Итеративное восстановление точки в n-мерном пространстве по $n + 1$ точкам и недостоверным расстояниям

hk@r4in.tk
mns@r4in.tk

15 сентября 2020 г.

1. Введение

Проблема восстановления координат точки в пространстве в целом проблемой не является. Формулировка задачи выглядит следующим образом: в гладком эвклидовом пространстве $\mathbb{R}^n$ существует точка $\mathbf{x} = \left\{x_i\right\}_{i=0}^{n-1}$, координаты которой неизвестны, но известны координаты $n + 1$ других точек $\mathbf{C} = \left\{\mathbf{c}_j\right\}_{j=0}^{n}$, не лежащих в одной гиперплоскости, а также расстояния $R = \left\{r_j\right\}_{j=0}^{n}$ от них до точки $\mathbf{x}$. Очевидно, что восстановление координат точки $\mathbf{x}$ осуществляется решением системы уравнений:

$$\begin{cases} r_0^2 = \sum\limits_{k=0}^{n-1} \left(x_k - c_{0,k}\right)^2 \\ r_1^2 = \sum\limits_{k=0}^{n-1} \left(x_k - c_{1,k}\right)^2 \\ \qquad\qquad \dots \\ r_n^2 = \sum\limits_{k=0}^{n-1} \left(x_k - c_{n,k}\right)^2 \end{cases} \tag{1}$$

Решение такой системы при $n = 2$ не представляет проблемы. При $n = 3$ оно уже сложнее. При дальнейшем увеличении n сложность решения такой системы уравнений растёт очень быстро[1]. Учитывая, что в настоящее время размерность может достигать весьма больших значений от сотен и более,

[1] Точная оценка вычислительной сложности решения таких систем уравнений в зависимости от n не приводится, так как зависит от конкретного алгоритма.

а кроме того может определяться самой структурой данных и меняться в ходе их обработки, представляется целесообразным выработать общий алгоритм восстановления точки, с приемлемым (и желательно контролируемым) временем выполнения для больших n.

Ситуация осложняется тем, что в прикладных задачах точность измерения расстояния может быть разной. Так, например, четыре радиоприёмника, оценивая расстояние до источника по силе сигнала, могут давать значения радиусов, которые не имеют единой точки пересечения, в результате чего решение такой системы уравнений будет комплексным, а следовательно неподходящим для практического использования. В подобных случаях необходимо не точное решение системы уравнений (1), а некое, желательно наилучшее, приближение. Одна из таких ситуаций приведена на рис.1. Представляется справедливым, что искомая точка вероятнее всего принадлежит области a, ограниченной 1-сферами с центрами в $\mathbf{c}_i$ и радиусами r_i соответственно. Данная гипотеза основана на том, что для любой точки внутри области a сумма расстояний до поверхностей сфер меньше, чем вне области a. В идеальном случае, если радиусы сфер абсолютно точны, то расстояния от искомой точки до поверхностей сфер равны 0 в соответствии со смыслом системы уравнений (1). Уточнения и раскрытие данного утверждения приведены ниже с рассмотрением случаев, когда область a может быть неограниченной, включая случай, когда ни одна пара сфер не имеет общих точек.

Также нельзя не отметить, что в прикладных задачах часто есть возможность получить данные не с $n + 1$ датчиков, дающих неточные значения, а с большего их (датчиков) количества. С одной стороны может показаться, что решение системы (1) при этом усложнится и это так, но если отталкиваться от иных подходов, то в прикладной сфере дополнительная информация может (и должна) улучшать приближение искомой точки, что будет показано в разделе 3.4.2.

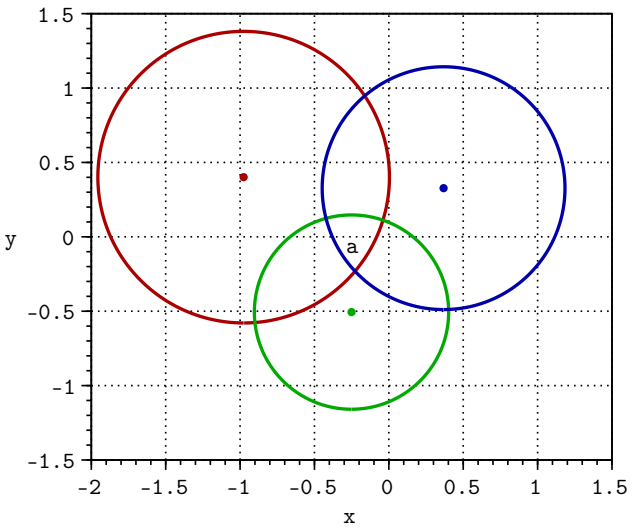

Рис. 1: Ситуация при $n = 2$,
$\mathbf{c}_0 = \{-0.976, 0.401\}$,
$\mathbf{c}_1 = \{-0.251, -0.506\}$,
$\mathbf{c}_2 = \{0.368, 0.327\}$,
$R = \{0.98, 0.653, 0.816\}$.

Здесь и далее иллюстрации для наглядности приводятся для случаев $n = 2$, однако, подразумевается, что все изложенное справедливо для любых $n \in \mathbb{N}$ и, соответственно, $(n - 1)$-сфер.

Таким образом, целью данного документа является описание алго-

ритма, который получает в качестве аргументов:

1. Размерность пространства – $n \in \mathbb{N}$;
2. Координаты m точек ($m \in \mathbb{N}$ и $m > n$)[2] в данном пространстве – $\mathbf{C} = \{\mathbf{c}_i \in \mathbb{R}^n\}_{i=0}^{m-1}$;
3. Недостоверные расстояния от точек $\mathbf{C}$ до искомой точки – $R = \{r_i \in \mathbb{R}\}_{i=0}^{m-1}$.

Результат работы алгоритма это точка $\mathbf{x} = \{x_i\}_{i=0}^{n-1}$, являющаяся наилучшим приближением искомой точки. Это расплывчатое определение дано в связи с тем, что при описании алгоритма умышленно не будет использоваться «настоящая» искомая точка, так как в практических задачах, для решения которых вырабатывается алгоритм, искомая точка всегда неизвестна даже в случае полностью контролируемых условий оценки алгоритма. В разделе 3.4 при оценке прототипа будет использована «настоящая» искомая точка для вычисления статистики по ошибке приближения, но при этом необходимо понимать, что хороший алгоритм должен демонстрировать прямую корреляцию качества своей работы и точности датчиков выдающих расстояния R, а в таком случае оценка расстояния между алгоритмически вычисленной и искомой точкой будет не оценкой алгоритма, а оценкой точности датчиков, причём косвенной. В общем же случае лучше говорить, что алгоритм рассчитывает координаты точки, где наиболее вероятно нахождение искомой точки; либо полученное приближение достаточно точно для принятия решения и дальнейших действий. Грубо говоря, если поставить целью «уничтожение» искомой точки бомбой, которая при детонации сохраняет разрушающее воздействие в радиусе 100 метров от эпицентра взрыва, то достаточно того, чтобы алгоритмически рассчитанная точка была менее чем в ста метрах от искомой.

2. Алгоритм

2.1. Предварительные утверждения

В данном разделе приведены самоочевидные или доказуемые утверждения, постулаты, леммы, отношение которых к описываемому алгоритму на данном этапе изложения может быть неясно. Поэтому вы можете при первом чтении пропустить этот раздел и перейти к общему описанию алгоритма (разд. 2.2) и в ходе его чтения по ссылкам вернуться в этот раздел при необходимости.

[2]Ограничение $m > n$ очевидно и продиктовано тем, что при $m < n + 1$ существует сфера (множество точек) такая, что искомая точка равновероятно находится в любой точке принадлежащей поверхности такой сферы.

2.1.1. Кратчайшее расстояние от точки до поверхности сферы в гладком эвклидовом пространстве

Определим функцию $\delta(\mathbf{a}, \mathbf{b})$ как эвклидово расстояние:

$$\delta(\mathbf{a}, \mathbf{b}) :\Leftrightarrow \sqrt[2]{\sum_{i=0}^{n-1} \left(a_i - b_i\right)^2} \tag{2}$$

Тогда минимальное расстояние μ от точки $\mathbf{a}$ до поверхности $(n-1)$-сферы с центром в $\mathbf{c}$ и радиусом r:

$$\begin{gathered}
\mathbf{S} :\Leftrightarrow \{\mathbf{s} \in \mathbb{R}^n \mid \delta(\mathbf{c}, \mathbf{s}) = r\} \\
\mathbf{s}_\mu = \arg\min_{\mathbf{s} \in \mathbf{S}} \delta(\mathbf{a}, \mathbf{s}) \\
\mu = \delta\left(\mathbf{a}, \mathbf{s}_\mu\right) \\
\mu(\mathbf{a}, \mathbf{c}, r) :\Leftrightarrow |\delta(\mathbf{a}, \mathbf{c}) - r|
\end{gathered} \tag{3}$$

Формула (3) это удобное следствие из определения сферы, позволяющее алгоритмически эффективно рассчитывать μ без вычисления точки $\mathbf{s}_\mu$. Также необходимо помнить, что при смене метрики на иную формула (3) может перестать работать. Особенно надо быть осторожным при смене метрики на непринадлежащую метрикам Минковского или даже на некоммутирующую функцию расстояния, и учитывать, что при смене функции расстояния должна меняться не только формула, но и определение сферы. Это связано с тем, что формула (3) в данном случае следует из определения сферы через метрику, или, иными словами, в данном случае формула (3) справедлива потому, что δ и r косвенно определены одна через другую (circulus vitiosus!).

В худших случаях смена функции расстояния может привести к тому, что $n + 1$ сфер могут иметь более одной общей точки и/или что точный способ нахождения кратчайшего расстояния от точки до сферы может быть неизвестен, сложен для машинного вычисления, или реализовано только приблизительное его вычисление через поиск $\arg\min_{\mathbf{s}} \delta(\mathbf{a}, \mathbf{s})$.

В дальнейшем в данном документе используется и подразумевается эвклидова метрика.

2.1.2. Уравнение координат пересечения $(n-1)$-сферы с прямой, проходящей через центр $(n-1)$-сферы

Для нахождения координат точки пересечения $(n-1)$-сферы в пространстве $\mathbb{R}^n$ с прямой необходимо решить систему уравнений:

$$\begin{cases} \frac{x_0-p_0}{v_0} = \frac{x_1-p_1}{v_1} \\ \frac{x_0-p_0}{v_0} = \frac{x_2-p_2}{v_2} \\ \quad\cdots \\ \frac{x_0-p_0}{v_0} = \frac{x_{n-1}-p_{n-1}}{v_{n-1}} \\ r^2 = \sum\limits_{i=0}^{n-1}\left(x_i - c_i\right)^2 \end{cases}$$

, где

$\mathbf{x} = \{x_i\}_{i=0}^{n-1}$ — точка пересечения;

$\mathbf{p} = \{p_i\}_{i=0}^{n-1}$ — точка принадлежащая прямой;

$\vec{v} = \{v_i\}_{i=0}^{n-1}$ — вектор коллинеарный прямой;

$\mathbf{c} = \{c_i\}_{i=0}^{n-1}$ — центр сферы;

r — радиус сферы;

n — размерность пространства.

(4)

Полагаю излишним приводить решение данной системы уравнений, но необходимо отметить, что в период первоначального составления документа математические программы актуальных версий (Wolfram Mathematica, Mathcad Prime) при $n > 3$ дают неверные, но приближенные решения системы уравнений (4). Так в ходе работы над алгоритмом было проведено ручное решение и декомпозиция системы (4) вплоть до $n = 5$, что и обнаружило неточность машинных решений системы. Поэтому в случае необходимости модификации алгоритма и решения подобных систем уравнений осторожно относитесь к результатам машинного решения уравнений, если нет полной уверенности в точности алгоритма решения. Далее приведена окончательная формула искомой точки с декомпозицией (внимательно отнеситесь к индексам!):

$$\mathbf{x} = \left\{ x_i = p_i + \frac{v_i\left(l \pm \sqrt[2]{e^2 - 4df}\right)}{2dv_0} \right\}_{i=0}^{n-1}, \tag{5}$$

где

$$g = \sum_{i=1}^{n-1} v_i^2 \qquad d = g + v_0^2$$

$$h = \sum_{i=1}^{n-1} \left(c_i - p_i\right)^2 \qquad e = 2\left(p_0 g + v_0\left(c_0 v_0 + k\right)\right)$$

$$k = \sum_{i=1}^{n-1} v_i\left(c_i - p_i\right) \qquad f = v_0^2\left(c_0^2 - r^2 + h\right) + p_0\left(p_0 g + 2v_0 k\right)$$

$$l = 2v_0\left(k + v_0\left(c_0 - p_0\right)\right)$$

Из формулы[3] (5) и геометрического смысла системы (4) очевидно,

[3]Обратите внимание, что d это дополнение суммы g до полной размерности n вектора $\vec{v}$; а l это произведение $2v_0$ на дополнение суммы k до полной размерности n вектора $\vec{v}$ и точек **p**, **c**. Данное замечание может быть полезно при реализации алгоритма.

что любая прямая проходящая через центр сферы с $r > 0$ в гладком эвклидовом пространстве имеет две точки пересечения с поверхностью сферы. Поэтому определим функцию $\chi\left(\mathbf{p}, \vec{v}, \mathbf{c}, r\right)$ как

$$\chi\left(\mathbf{p}, \vec{v}, \mathbf{c}, r\right) :\Leftrightarrow \left\{\mathbf{x}_0, \mathbf{x}_1\right\}, \tag{6}$$

где

$$\mathbf{x}_0 = \left\{x_i = p_i + \frac{v_i\left(l + \sqrt[2]{e^2 - 4df}\right)}{2dv_0}\right\}_{i=0}^{n-1} \quad \text{и} \quad \mathbf{x}_1 = \left\{x_i = p_i + \frac{v_i\left(l - \sqrt[2]{e^2 - 4df}\right)}{2dv_0}\right\}_{i=0}^{n-1}$$

Замечание. Так как ожидается, что $\mathbf{x} \in \mathbb{R}^n$, то формула (5) влечёт явные ограничения. Так $\mathbf{x} \notin \mathbb{R}^n$ в следующих случаях:

(a) $v_0 = 0$, т. е первая компонента вектора $\vec{v}$ системы (4) равна нолю. В геометрическом смысле вектор $\vec{v}$ ортогонален абсциссе.

(b) $\|\vec{v}\| = 0 \implies d = 0$, т. е норма вектора $\vec{v}$ и следовательно сумма d квадратов компонент вектора $\vec{v}$ системы (4) в формуле (5) равны нолю. В геометрическом смысле вектор $\vec{v}$ имеет нулевую длину, в связи с чем его направление не определено.

(c) $e^2 - 4df < 0$, т. е подкоренное выражение формулы (5) отрицательно. В геометрическом смысле это значит, что поверхность сферы не имеет общих точек с прямой в гладком эвклидовом пространстве. Может показаться, что такое невозможно раз уж прямая проходит через центр сферы, но как всегда есть нюансы: во-первых это утверждение верно не для всех сочетаний размерностей; а, во-вторых, особенности обработки чисел с плавающей запятой в практических реализациях алгоритма могут давать результаты не соответствующие абстрактным математическим ожиданиям.

2.2. Общее описание алгоритма

В целом алгоритм является вариантом градиентного спуска с дополнительной информацией. Стоит напомнить, что согласно разделу 1 аргументами алгоритма являются:

- размерность пространства $n \in \mathbb{N}$;
- координаты m точек $\mathbf{C} = \left\{\mathbf{c}_i \in \mathbb{R}^n\right\}_{i=0}^{m-1}$, $m \in \mathbb{N}$ и $m > n$;
- недостоверные расстояния $R = \left\{r_i\right\}_{i=0}^{m-1}$.

Пары «точка-расстояние» рассматриваются как описание множества сфер $\mathfrak{S} = \left\{\left\{\mathbf{c}_i, r_i\right\}\right\}_{i=0}^{m-1}$ («центр-радиус»), то есть постулируется взаимно однозначное соответствие по индексу известной точки и расстояния от неё до искомой точки.

Далее приводится пошаговая схема алгоритма:

1. Методом ϕ (разд. 2.3.1 (8)) выбирается начальное приближение искомой точки.

$$\mathbf{p} = \phi\left(\mathbf{C}\right)$$

2. Выбирается сфера $\left\{\mathbf{c}_j, r_j\right\}$, поверхность которой наиболее удалена по μ (разд. 2.1.1 (3)) от $\mathbf{p}$.

$$\forall \left\{\mathbf{c}, r\right\} \in \mathfrak{S} \;:\; \bigwedge_{i=0}^{m-1} \mu\left(\mathbf{p}, \mathbf{c}_j, r_j\right) \geqslant \mu\left(\mathbf{p}, \mathbf{c}_i, r_i\right)$$

3. Методом χ (разд. 2.1.2 (6)) рассчитывается пара точек $\mathbf{Q} = \left\{\mathbf{q}_0, \mathbf{q}_1\right\}$, принадлежащих сфере $\left\{\mathbf{c}_j, r_j\right\}$ и прямой проходящей через точки $\mathbf{c}$ и $\mathbf{p}$.

$$\mathbf{Q} = \left\{\mathbf{q}_0, \mathbf{q}_1\right\} = \chi\left(\mathbf{p}, \vec{v}, \mathbf{c}_j, r_j\right)$$

4. К результатам χ применяется процедура Λ, которая выбирает из пары $\mathbf{Q}$ точку, для которой значение λ (разд. 2.3.2 (9)) больше, или, иными словами, рассчитывается показатель качества приближения и выбирается наилучшее приближение искомой точки из пары $\left\{\mathbf{q}_0, \mathbf{q}_1\right\}$.

$$\Lambda\left(\mathbf{Q}, \mathbf{C}, R\right) :\Leftrightarrow \left(\mathbf{t} \in \mathbf{Q} \mid \forall \mathbf{q} \in \mathbf{Q} \;:\; \bigwedge_{i=0}^{|\mathbf{Q}|-1} \lambda\left(\mathbf{t}, \mathbf{C}, R\right) \geqslant \lambda\left(\mathbf{q}_i, \mathbf{C}, R\right)\right) \qquad (7)$$

5. $\mathbf{p}$ присваивается значение Λ.

$$\mathbf{p} = \Lambda\left(\mathbf{Q}, \mathbf{C}, R\right)$$

6. Проверяется утверждение Z (разд. 2.3.3), если Z истинно, осуществляется переход к шагу 8.

7. Переход к шагу 2.

8. Методом ψ (разд. 2.3.4 (11)) осуществляется уточнение приближения искомой точки.

$$\mathbf{x} = \psi\left(\mathbf{p}, \mathbf{C}, R\right)$$

2.3. Предположения

В данном разделе приведены неочевидные предположения, доказуемость которых неустановлена, но они обоснованы рядом аргументов или проверены эмпирическим путём. При этом надо понимать, что результат эмпирической проверки может быть статистической флуктуацией.

2.3.1. Выбор начального приближения искомой точки

В принципе строгих ограничений выбора начального приближения искомой точки не выявлено. В случае если расстояния до искомой точки известны точно и являются абсолютно достоверными, то никаких ограничений на выбор начального приближения нет и можно брать случайную точку. Но если расстояния до искомой точки недостоверны, то в ряде случаев при «неудачном» выборе начального приближения, результат работы алгоритма может не быть «лучшим», а только «хорошим» приближением. Это связано с тем, что в данном ряде случаев может быть не одна, а несколько областей, в которых вероятно присутствие искомой точки. Если говорить в терминах градиентного спуска, то может быть более одного глобального экстремума функции λ и/или ещё и несколько локальных экстремумов, в один из которых и приведёт алгоритм, в случае «плохого» выбора начального приближения. В связи с этим были изучены несколько подходов к выбору начального приближения при прочих равных, и проверены эмпирически на нерепрезентативной выборке:

1. Использовать в качестве начального приближения один из центров $\mathbf{C}$.
2. Использовать в качестве начального приближения одну из вершин параллелограмма (естественно, n-мерного), грани которого являются касательными к сферам в различных точках.
3. Использовать в качестве приближения усреднение по точкам $\mathbf{C}$.

В данном документе не приводится статистика проверок предложенных методов, в связи с тем, что (1) данный список не исчерпывающий, был проверен примерно десяток подходов, но записи о них производились на черновиках и утрачены; (2) выборка при проверке была нерепрезентативна и при принятии решения бесполезна; (3) ни один из проверенных способов не показал полного избегания локальных экстремумов, то есть независимо от выбора подхода возникают ситуации, при которых алгоритм завершается не в глобальном, а в локальном экстремуме функции λ.

Лучший эмпирический результат показал и наиболее обоснованным представляется метод 3, в связи с чем функция ϕ определена как

$$\phi\left(\mathbf{C}\right) :\Leftrightarrow \left\{ \sum_{i=0}^{m-1} c_{i,j} \middle/ m \right\}_{j=0}^{n-1} \tag{8}$$

В формуле (8) используется среднее арифметическое, но допустимо поэкспериментировать с другими степенными средними, а при достаточно больших значениях n можно испытать и нестепенные средние. По крайней мере, на момент написания документа, не было сформулировано аргументов против использования других средних.

В практической реализации алгоритма, если время принятия решения позволяет или есть возможность параллельных вычислений, можно снизить вероятность «плохого» выбора начального приближения путём вычисления алгоритма из нескольких начальных приближений подобранных разными методами, а затем выбрать из нескольких результатов приближение с наилучшим качеством (см. разд. 2.3.2). Так, например, в прототипе алгоритма в качестве начального приближения берутся общее усреднение по известным точкам и попарные усреднения со сдвигом, в последствии выбирается наилучшее конечное приближение.

2.3.2. Критерий качества точки

Предполагается, что в искомой точке расстояния до всех сфер равны нолю, а для наилучшего приближения минимальны, в связи с чем критерий качества точки может быть выражен очень разными способами, дающими, в общем, сходные результаты. Поэтому в качестве критерия был выбран самый простой, а именно отрицательная сумма расстояний от точки до поверхности всех сфер.

$$\lambda\left(\mathbf{p}, \mathbf{C}, R\right) :\Leftrightarrow -\sum_{i=0}^{m-1} \mu\left(\mathbf{p}, \mathbf{c}_i, r_i\right) \tag{9}$$

Отрицательная сумма взята исключительно из эстетических соображений, чтобы показатель качества рос с приближением к точке пересечения сфер, в которой он очевидно обращается в ноль, а если расстояния недостоверны, то просто максимизируется.

Также как и выбор начального приближения точки, данная функция качества была проверена эмпирически среди других подходов, показала хороший результат, имеет достаточно очевидную и твёрдую аргументацию под собой, но не доказательство. Её можно сменить, например, на средние расстояний или иным образом, исходя из иных предположений о качестве точки.

2.3.3. Условие последней итерации

Условием последней итерации является истинность утверждения Z. Само утверждение Z сильно зависит от решаемой задачи и обстановки исполнения алгоритма. Поэтому будет рассмотрено три варианта утверждения Z, в зависимости от надёжности значений R и требований обстановки.

Утверждение Z **при абсолютно точных значениях** R**.** Так, в случае если расстояния от известных точек до искомой достоверны и абсолютно точны[4], то последней считается итерация, после которой сумма расстояний от найденного приближения до поверхности всех сфер равна нолю или машинному ε (эпсилон).

$$\mathrm{Z} \Leftrightarrow \sum_{i=0}^{m-1} \mu\left(\mathbf{p}, \mathbf{c}_i, r_i\right) = 0$$

При истинности такого утверждения шаг 8 алгоритма можно пропустить, так как уточнять приближение не требуется, да и невозможно. Но вероятность возникновения такой ситуации в реальности стремиться к нолю, поэтому рассмотрим другое утверждение Z при ухудшении обстановки.

Утверждение Z **при недостоверных значениях** R **и известной допустимой ошибке приближения** Δ**.** Предположим, множество R содержит заведомо неточные расстояния, но для принятия решения, необходимо приближение, расположенное не далее, чем в Δ от искомой точки. В таком случае, учитывая, что на шаге 5 алгоритма $\exists \mathbf{S}_i \in \mathfrak{S} : \mathbf{p} \in \mathbf{S}_i \Rightarrow \mu\left(\mathbf{p}, \mathbf{c}_i, r_i\right) = 0$, то если расстояния от данного приближения до всех из $m-1$ приближений, которые могут быть получены относительно других сфер, меньше Δ, утверждение Z истинно.

$$\mathrm{Z} \Leftrightarrow \bigwedge_{i=0}^{m-1} \mu\left(\mathbf{p}, \mathbf{c}_i, r_i\right) < \Delta$$

В зависимости от задачи и обстановки можно использовать строгое и нестрогое сравнение.

Необходимо понимать, что использование такого утверждения возможно только тогда, когда погрешность измерения в датчике-источнике значений R существенно меньше Δ. Иными словами, если известно матожидание погрешности датчиков-источников R, то допустимо использовать эту величину с запасом (например, $2 \times 3\sigma$) в качестве Δ. Если же Δ меньше погрешности датчиков-источников значений R, возникнет ситуация, при которой состояние «Z *истинно*» недостижимо, в связи с чем рассмотрим наихудшую обстановку.

Утверждение Z **при недостоверных значениях** R **и отсутствии информации о допустимой ошибке приближения.** Наихудшая ситуация, но и наиболее частая на практике: значения **C** и R недостоверны и информации о величине возможной ошибки нет и не предвидится. Для этого было эмпирически выработано утверждение Z, которое может показаться избыточным, однако, соответствуют неким ситуациям, возникшим в ходе эксплуатации реализации алгоритма. В связи с вышеизложенным, если нет уверен-

[4] В контексте документа абсолютно точными можно считать значения R при абстрактном анализе алгоритма без использования актуальных значений, либо ситуацию, когда ошибка измерения в датчике-источнике значений R существенно меньше машинного ε (эпсилон).

ности в минимальной стабильности обстановки работы алгоритма, рекомендуется использовать данное или ещё более строгое относительно данного утверждение Z.

$$\mathrm{Z} \Leftrightarrow \left(\forall \lambda \in L \, : \bigwedge_{i=\omega-m^2}^{\omega} \lambda_{max} \geqslant \lambda\left(\mathbf{p}_i, \mathbf{C}, R\right) \right) \vee \omega \geqslant (wm), \tag{10}$$

где

ω — номер текущей итерации алгоритма, увеличение ω на единицу происходит на шаге 7 алгоритма;

$L = \left\{\lambda\left(\mathbf{p}_i, \mathbf{C}, R\right)\right\}_{i=\omega-m^2}^{\omega}$ — множество значений функции λ для m^2 последних итераций, соответственно $\mathbf{p}_i$ – приближение i-той итерации;

$\lambda_{max} \mid \forall \lambda \in \left\{\lambda\left(\mathbf{p}_i, \mathbf{C}, R\right)\right\}_{i=0}^{\omega-1} : \bigwedge_{i=0}^{\omega-1} \lambda_{max} \geqslant \lambda\left(\mathbf{p}_i, \mathbf{C}, R\right)$ — максимальное значение функции λ среди всех итераций алгоритма, кроме последней;

w — предопределённая константа, которая определяет максимально возможное количество итераций из расчёта на количество известных сфер, иными словами, если прошло (wm) итераций, то производим уточнение ψ и возвращаем результат.

w выбирается исходя из ограничений по времени и в прототипе определено как 100 на сферу (см. раздел 3.2.2), а значит прототип всегда завершается после $100m$ итераций и переходит к уточнению ψ. Очевидно, что w должно выбираться так, что $w > m$, ведь в ином случае становится бессмысленным сравнение с λ_{max} для последних m^2 итераций, либо должен быть снижен критерий количества итераций без улучшения под m^2.

Данное условие обосновано только эмпирически и может быть изменено в соответствии с иными представлениями об обстоятельствах влекущих завершение работы алгоритма. Дополнительные соображения по выбору w в условиях строгих ограничений на время исполнения алгоритма приведены в замечании к разделу 3.2.2.

2.3.4. Уточнение приближения последней итерации

Учитывая, что все точки полученные как значение функции χ принадлежат поверхности одной из известных сфер, а при неточных значениях расстояний R сферы вероятнее всего не имеют единой точки пересечения, следовательно любое приближение принадлежащее одной из сфер будет иметь ненулевое расстояние до одной или более других сфер. В связи с

этим имеет смысл уточнять приближение полученное после установления истинности высказывания Z таким образом, чтобы в идеальном случае (когда расстояние от приближения до всех сфер равно нолю) уточнение не перемещало точку **p**. В числе прочих для этого подходит усреднение по m точкам, которые являются результатом выбора лучшего приближения для каждой из известных сфер от текущего приближения, или в символах:

$$\psi\left(\mathbf{p}, \mathbf{C}, R\right) :\Leftrightarrow \left\{ \sum_{i=0}^{m-1} t_{i,j} \middle/ m \right\}_{j=0}^{n-1}, \tag{11}$$

где

$$\mathbf{T} = \left\{ \mathbf{t}_i = \Lambda\left(\chi\left(\mathbf{p}, \vec{v}, \mathbf{c}_i, r_i\right), \mathbf{C}, R\right) \right\}_{i=0}^{m-1}$$

и Λ — процедура (7), определённая для шага 4 общего описания алгоритма в разделе 2.2.

Данный подход выбирался исходя из ситуации полной неопределённости погрешности датчиков-источников R и даже более сильном условии, состоящим в том, что сама искомая точка принципиально не определяема точно. Поэтому, исходя из особенностей частной обстановки применения алгоритма, вполне возможно, что и среднее арифметическое, и метод Λ, и процедура уточнения в целом могут быть изменены на релевантные специфичным условиям применения алгоритма.

3. Прототип

Прототип представляет собой реализацию вышеописанного алгоритма на языке C. Прототип не является строго оптимальной реализацией алгоритма и предназначен для проверки возможных изменений алгоритма и сравнения результатов (регресс-оценки). Но в целом, если к программному обеспечению не предъявляется строгих требований по времени и пространству исполнения, то можно использовать прототип «как есть».

3.1. Состав прототипа

Получить копию прототипа можно по адресу:
https://ggs.void.r4in.tk/hk/iterative_point_recovery/archive/master.tar.gz
после чего распаковать полученный архив. Кроме того, при наличии установленной СКВ Git (https://git-scm.com) можно получить копию прототипа командой

```
git clone https://ggs.void.r4in.tk/hk/iterative_point_recovery.git
```

В состав прототипа входят следующие файлы и каталоги:

- `iterative_point_recovery/`
  - `documentation/` — исходный код и сопутствующие файлы данного документа, см. раздел 3.5.
  - `examples/` — исходный код с примерами использования прототипа, см. раздел 3.3.
  - `src/` — исходный код собственно прототипа, см. раздел 3.2.
  - `test_suite/` — исходный код проверочного комплекта прототипа, см. раздел 3.4.
  - `visualisation/` — набор сценариев для визуализации результатов тестирования, см. раздел 3.4.
  - `Makefile` — сценарий сборки прототипа, см. раздел 3.2.

Далее рассмотрены составляющие прототипа, даны их краткие описания и руководства по использованию.

3.2. Прототип. Сборка и описание

Основной блок прототипа представляет собой функцию на языке C, которая может быть собрана в библиотеку или использована в виде исходного кода «как есть». К основному блоку прототипа относятся:

- `iterative_point_recovery/`
  - …
  - `src/`
    - `include_hd/`
      - `iterative_point_recovery.clh` — заголовочный (header) файл, содержащий объявления функций и структур прототипа, см. раздел 3.2.3.
      - `ptrc_hd_settings.clh` — заголовочный (header) файл, содержащий настройки прототипа, см. раздел 3.2.2.
    - `iterative_point_recovery.clc` — файл, содержащий определения функций прототипа, см. раздел 3.2.3.
  - …
  - `Makefile` — сценарий сборки прототипа в библиотеку, см. раздел 3.2.1.

3.2.1. Сборка и установка прототипа

Сборка. Для сборки по умолчанию необходимо перейти в каталог `iterative_point_recovery` и выполнить команду

```
make
```

Если команда завершена без ошибок, то в каталоге `iterative_point_recovery/build` появится файл `libIterPntRcv.so.`*I*`.`*J* и несколько несколько служебных файлов. В имени файла *I* – основная версия библиотеки, а *J* – дополнительная.

При необходимости возможна сборка для отладки (debug) командой

```
make debug
```

Если при сборке указать переменную `GPGPU_DEV_IDX=`*`N`*, например так

```
make GPGPU_DEV_IDX=0
```

то, при наличии установленного пакета OpenCL_helpers (https://ggs.void.r4in.tk/hk/OpenCL_helpers), кроме обычной библиотеки будет собрана OpenCL-библиотека с прототипом для GPGPU-устройства с индексом *N* в системе. Если пакет OpenCL_helpers был установлен не в путь по умолчанию, то надо указать актуальный путь в переменной `OCLH_PATH=`*`путь_установки`*. После сборки с указанием переменной `GPGPU_DEV_IDX`, если сборка завершена без ошибок, в каталоге `iterative_point_recovery/build` появится появится дополнительный файл `libIterPntRcv-`*`имя_устройства_GPGPU`*`.clso`.

Установка. Установка осуществляется командой

```
make install
```

В результате будет создан каталог `~/opt/iterative_point_recovery`, куда в подкаталоги `lib`, `include_hd` будут скопированы файлы библиотеки и заголовочные файлы соответственно. Затем целесообразно добавить каталог `~/opt/iterative_point_recovery/lib` в переменную окружения `LD_LIBRARY_PATH`.

Можно изменить целевой путь если задать команду

```
make PRFX_PATH=путь_установки install
```

Удаление. Удаление производится командой

```
make uninstall
```

либо

```
make PRFX_PATH=путь_установки uninstall
```

если установка производилась не в каталог по умолчанию.

3.2.2. Настройки прототипа

Настройка прототипа производится изменением макроопределений в файле `src/include_hd/ptrc_hd_settings.clh`:

`_PTRC_VAL_T`

тип значений с плавающей точкой компонент точек и векторов. Допустимые значения `half`, `float`, `double`, `long double`. Учитывайте, поддерживается ли данный тип компилятором и математическими библиотеками или, иными словами, перегружены ли для данного типа операторы `+`, `-`, `*`, `/` и функции `sqrt()`, `abs()`. Также учитывайте, что на момент написания данного документа тип `long double` и более длинные чаще всего обрабатываются не аппаратно, а программно,

что влечёт значительное увеличение времени расчётов. В прототипе данное макроопределение установлено в значение `flt32_t`.

`_PTRC_MAX_ITERATIONS_ON_SPHERE`
: значение w условия `Z` (см. разд. 2.3.3 (10)). В прототипе данное макроопределение установлено в значение `100`.

> **Важное замечание:** ведение данной настройки может показаться бессмысленным, так как вероятность срабатывания условия $\omega \geqslant (wm)$ (разд. 2.3.3 (10)) очень мала, но ненулевая. Для RTOS в условиях строгого ограничения времени эта настройка становится актуальной, поэтому рекомендуется использовать следующие подходы для выбора w:
>
> $$w = \left\lfloor \frac{\texttt{CutOff} / \texttt{ExecT}}{m} \right\rfloor,$$
>
> где
>
> m — количество известных сфер (датчиков-источников расстояний);
>
> `CutOff` — интервал времени отсечки восстановления точки (по истечении этого интервала результат восстановления теряет смысл). Может измеряться в секундах (чаще нано- или микро-) для гибридных архитектур процессоров или в тактах для традиционных архитектур;
>
> `ExecT` — интервал времени выполнения (в тех же единицах, что и `CutOff`) функции `_ptrc_recoverPoint()` (см. раздел 3.2.3) при `_PTRC_MAX_ITERATIONS_ON_SPHERE` определённом как `1`, аргументе `u64NofRuns` равном `1` и точном выходе по условию $\omega \geqslant (wm)$.
>
> Оценка `ExecT` для традиционных архитектур процессоров должна показывать постоянное время отличающееся от запуска к запуску менее, чем на время одного такта процессора и может оцениваться по количеству машинных команд. Если такого не наблюдается или используется процессор гибридной архитектуры, то рекомендуется брать максимальное значение `ExecT` не менее чем от 10000 замеров, так как практические оценки показывают, что кривая времени для гибридных архитектур достаточно сглаживается только на 10-ти тысячах и более замеров.
>
> При замерах необходимо учитывать контур размещения внутренних часов процессора, по возможности использовать замеры времени по часам на внутреннем контуре. Необходимо учитывать количество регистров часов и возможность множественных переполнений.

Данный подход сформулирован исходя из пессимистичных соображений оценки времени и тактов исполнения, и даёт время/такты «с запасом», что идеологически верно, однако при уверенности в стабильности работы архитектуры и константных оценках времени/тактов можно добавить оптимизма и, проведя поблочную оценку времени исполнения одной итерации алгоритма в прототипе, уточнить оценку приемлемого значения w.

`_PTRC_NONRECONSTRUCTABLE_PNT_ERR`
: значение возвращаемое функцией в случае когда нет возможности сформировать приближение искомой точки или, иными словами это код ошибки «точка не восстанавливаема». Такая ошибка возвращается в случае если $m < n + 1$ (см. сноску 2 на стр. 3); в случаях из замечания на стр. 6; и если указано нулевое количество проверяемых начальных приближений (см. аргумент `u64NofRuns` основной функции прототипа в разделе 3.2.3). В прототипе данное макроопределение установлено в значение `73`.

`_PTRC_MEMALLOC_ERR`
: значение функции, возвращаемое в случае ошибки выделения памяти. В прототипе данное макроопределение установлено в значение `-150`.

3.2.3. Функции и структуры прототипа

Почти все структуры и функции объявлены и определены таким образом, что прототип без изменений компилируется как компилятором соответствующим стандарту C11 (ISO/IEC 9899:2011), так и компиляторами OpenCL-диалекта C версии 1.2 и выше. Это достигается за счёт широкого использования препроцессора языка C и макроопределением, указывающим, что необходимо осуществлять сборку для OpenCL-диалекта C, является макроопределение `_OCLH_OCL_COMPILER_` – если оно определено, то препроцессор сгенерирует код OpenCL-диалекта C, в ином случае будет сгенерирован код соответствующий стандарту C11. Самостоятельно определять `_OCLH_OCL_COMPILER_` нет необходимости, если вы используете пакет OpenCL_helpers, который автоматически определяет данный макрос при компиляции. Кроме того, в связи с особенностями выделения памяти на GPGPU и вычислительных акселераторах, код OpenCL-диалекта C использует заголовочный файл `oclh_d_mem_alloc.clh` пакета OpenCL_helpers.

> **Важное замечание:** Пакет OpenCL_helpers реализует единый механизм выделения памяти для вычислительных акселераторов как без прямого доступа к оперативной памяти несущего компьютера, так и с таким доступом. Но для акселераторов с доступом данный механизм может работать медленнее, чем реализации `malloc()/free()` от их производителей. Для использования выделения, проверки и освобождения памяти релевантных диалекту OpenCL-диалекта C конкретного вычислительного акселератора необходимо переопределить макропределения `__PTRC_VAL_T_alloc_and_check` и `__PTRC_VAL_T_free` в файле `src/include_hd/iterative_point_recovery.clh`.

Структура, описывающая исходные данные. Исходные данные хранятся и передаются в структуре

```
typedef struct _PTRC_INPUT_DATA {
              uint64_t     u64D;
    __private _PTRC_VAL_T* pfCnt;
    __private _PTRC_VAL_T* pfRds;
              uint64_t     u64NofS;
} _PTRC_INDAT;
```

Модификаторы `__private` не имеют смысла для языка C стандарта C11 и будут проигнорированы в соответствии с макроопределением. Для OpenCL-диалекта C эти модификаторы обозначают, что указатели указывают на собственную память нити исполнения.

`u64D`
: размерность пространства.

`pfCnt`
: указатель на координаты `u64NofS` центров известных сфер, по `u64D` значений для каждого центра.

`pfRds`
: указатель на `u64NofS` неточных расстояний от центров известных сфер до искомой точки.

`u64NofS`
: количество известных сфер.

Основная функция прототипа. Основной функцией прототипа является `_ptrc_recoverPoint()` и для стандарта C11 она объявлена как

```
int32_t _ptrc_recoverPoint(      _PTRC_VAL_T* const pfDst,
                                 _PTRC_VAL_T* const pfQlt,
                           const _PTRC_INDAT        in,
                           const uint64_t           u64NofRuns)
```

а для OpenCL-диалекта C как

```
int32_t _ptrc_recoverPoint(__private       _PTRC_VAL_T* const pfDst,
                           __private       _PTRC_VAL_T* const pfQlt,
                           __private const _PTRC_INDAT        in,
                           __private const uint64_t           u64NofRuns,
                           _GDM_heap_PROTO(__private))
```

Функция возвращает 0 в случае нахождения приближения, _PTRC_MEMALLOC_ERR – в случае ошибки выделения памяти и _PTRC_NONRECONSTRUCTABLE_PNT_ERR – в случае если восстановление точки не представилось возможным (коды ошибок см. в разд. 3.2.2).

in
: исходные данные в виде структуры _PTRC_INDAT.

pfDst
: указатель на память размером не менее чем in.u64D размеров _PTRC_VAL_T, где будет сохранено лучшее найденное приближение.

pfQlt
: указатель на память размером не менее чем один размер _PTRC_VAL_T, где будет сохранен показатель качества λ (см. раздел 2.3.2) лучшего найденного приближения.

u64NofRuns
: количество начальных приближений, от которых будет произведён поиск наилучшего приближения. Первое начальное приближение является усреднением по центрам известных сфер, последующие начальные приближения являются попарными усреднениями известных центров сфер со сдвигом к лучшему найденному приближению. Если u64NofRuns<1, то функция вернёт _PTRC_NONRECONSTRUCTABLE_PNT_ERR.

_GDM_heap_PROTO(__private)
: макроопределение-прототип собственной «кучи» нити исполнения OpenCL для выделения памяти. При вызове функции в качестве данного аргумента передаётся макроопределение _GDM_heap_ARG(__private). Перед использованием макроопределения _GDM_heap_ARG(__private) собственная «куча» нити исполнения должна быть инициализирована макроопределением _GDM___private_heap_init().

Внутренние функции и структуры прототипа. Далее обзорно описаны функции, вызываемые при выполнении _ptrc_recoverPoint(). Модификаторы __private не имеют смысла для языка C стандарта C11 и будут проигнорированы в соответствии с макроопределением. Для

OpenCL-диалекта C эти модификаторы обозначают, что указатели указывают на собственную память нити исполнения.

```
int32_t __ptrc_Phi_meanOfPoints(
                          __private       _PTRC_VAL_T* const pfDstPnt,
                          __private const _PTRC_VAL_T* const pfSrcPnt,
                          __private const uint64_t           u64D,
                          __private const uint64_t         u64NofSrcPnts)

int32_t __ptrc_meanOfPointsBiased(
                          __private       _PTRC_VAL_T* const pfDstPnt,
                          __private const _PTRC_VAL_T* const pfSrcPnt,
                          __private const _PTRC_VAL_T* const pfBiasPnt,
                          __private const uint64_t           u64D,
                          __private const uint64_t       u64NofSrcPnts)
```

Функции являются реализациями функции ϕ (разд. 2.3.1) и осуществляют выбор начального приближения. Функция `__ptrc_Phi_meanOfPoints()` сохраняет в `pfDstPnt` усреднение по точкам `pfSrcPnt`. Функция `__ptrc_meanOfPointsBiased()` сохраняет в `pfDstPnt` усреднение по точкам `pfSrcPnt` со сдвигом к `pfBiasPnt`. `u64D` и `u64NofSrcPnts` — размерность пространства и количество исходных точек соответственно.

```
int32_t __ptrc_onePointRoutine(__private       _PTRC_VAL_T* const pfPnt,
                               __private const _PTRC_INDAT        in,
                               _GDM_heap_PROTO(__private))
```

Функция является реализацией шагов 2–8 алгоритма (разд. 2.2) для исходных данных `in` и одного начального приближения `pfPnt`, по этому же указателю будет сохранено найденное приближение. Для вызова функции в программах стандарта C11 аргумент `_GDM_heap_PROTO(__private)` не указывается; для вызова функции в программах OpenCL-диалекта C в качестве данного аргумента передаётся макроопределение `_GDM_heap_ARG(__private)`. Перед использованием макроопределения `_GDM_heap_ARG(__private)` собственная «куча» нити исполнения должна быть инициализирована макроопределением `_GDM___private_heap_init()`.

```
_PTRC_VAL_T __ptrc_Lambda_quality(
                          __private const _PTRC_VAL_T* const pfPnt,
                          __private const _PTRC_INDAT        in)
```

Функция является реализацией функции λ (разд. 2.3.2 (9)) и возвращает значение качества приближения `pfPnt` для исходных данных `in`.

```
int32_t __ptrc_copyVec(__private       _PTRC_VAL_T* const pfDst,
                       __private const _PTRC_VAL_T* const pfSrc,
                       __private       uint64_t           u64D)
```

Функция копирования вектора/точки `pfSrc` размерности `u64D` в вектор/точку `pfDst`.

```
uint64_t __ptrc_idxOfFarestSphere(
                          __private const _PTRC_VAL_T* const pfPnt,
                          __private const _PTRC_INDAT        in)
```

Функция выбора наиболее удалённой сферы реализует шаг 2 алгоритма (разд. 2.2) и функцию μ (разд. 2.1.1 (3)). Возвращает индекс сферы из исходных данных `in`, поверхность которой наиболее удалена от точки `pfPnt`.

```
int32_t __ptrc_nextPntAndQuality(
                          __private       _PTRC_VAL_T* const pfDst,
                          __private const _PTRC_VAL_T* const pfPnt,
                          __private const _PTRC_INDAT        in,
                          __private const uint64_t    u64IdxOfSphere,
                          __private      _PTRC_VAL_T* const pfQuality,
                          _GDM_heap_PROTO(__private))
```

Функция реализует шаги 3–4 алгоритма (разд. 2.2). От приближения `pfPnt` рассчитывается новое приближение к сфере из `in` с индексом `u64IdxOfSphere`. Новое приближение сохраняется по указателю `pfDst`; показатель качества нового приближения сохраняется по указателю `pfQuality`. Допустимо передавать в качестве `pfDst` и `pfPnt` один адрес. Для вызова функции в программах стандарта C11 аргумент `_GDM_heap_PROTO(__private)` не указывается; для вызова функции в программах OpenCL-диалекта C в качестве данного аргумента передаётся макроопределение `_GDM_heap_ARG(__private)`. Перед использованием макроопределения `_GDM_heap_ARG(__private)` собственная «куча» нити исполнения должна быть инициализирована макроопределением `_GDM___private_heap_init()`.

```
int32_t __ptrc_Psi_refineApproximation(
                          __private       _PTRC_VAL_T* const pfPnt,
                          __private const _PTRC_INDAT        in,
                          _GDM_heap_PROTO(__private))
```

Функция реализует процедуру ψ (разд. 2.3.4 (11)). Для приближения `pfPnt` и исходных данных `in` рассчитывается уточнение приближения, которое

сохраняется по указателю pfPnt. Для вызова функции в программах стандарта C11 аргумент _GDM_heap_PROTO(__private) не указывается; для вызова функции в программах OpenCL-диалекта C в качестве данного аргумента передаётся макроопределение _GDM_heap_ARG(__private). Перед использованием макроопределения _GDM_heap_ARG(__private) собственная «куча» нити исполнения должна быть инициализирована макроопределением _GDM___private_heap_init().

```
_PTRC_VAL_T __ptrc_EuclDist(__private const _PTRC_VAL_T* const pfA,
                            __private const _PTRC_VAL_T* const pfB,
                            __private       uint64_t           u64D)
```

Функция расчёта расстояния в эвклидовой метрике, реализует функцию δ (разд. 2.1.1 (2)). Возвращает значение расстояния между точками pfA и pfB в пространстве размерности u64D.

```
int32_t __ptrc_Chi_intersections(__private       _PTRC_VAL_T* const pfDst,
                                 __private const _PTRC_VAL_T* const pfP,
                                 __private const _PTRC_VAL_T* const pfV,
                                 __private const _PTRC_VAL_T* const pfC,
                                 __private const _PTRC_VAL_T        fR,
                                 __private const uint64_t           u64D)
```

Функция реализует функцию χ (разд. 2.1.2 (5) и (6)). Функция в пространстве размерности u64D рассчитывает пару точек, в которых прямая, проходящая через точки pfP и pfC, пересекает поверхность сферы с центром в pfC и радиусом fR. Результат (пара точек) сохраняется по указателю pfDst.

```
_PTRC_DEFL_VARS __ptrc_deflFunc(__private const _PTRC_VAL_T* const P,
                                __private const _PTRC_VAL_T* const V,
                                __private const _PTRC_VAL_T* const C,
                                __private const uint64_t           u64D,
                                __private const _PTRC_VAL_T        r)
_PTRC_GHK_VARS __ptrc_ghkFunc(__private const _PTRC_VAL_T* const P,
                              __private const _PTRC_VAL_T* const V,
                              __private const _PTRC_VAL_T* const C,
                              __private const uint64_t           u64D)
```

Функции рассчитывают значения d, e, f, g, h, k, l формулы (5) раздела 2.1.2. Возвращают структуры, содержащие эти значения:

```
typedef struct _PTRC_G_H_K_VARS {
    _PTRC_VAL_T g; _PTRC_VAL_T h; _PTRC_VAL_T k;
} _PTRC_GHK_VARS;

typedef struct _PTRC_D_E_F_L_VARS {
    _PTRC_VAL_T d; _PTRC_VAL_T e; _PTRC_VAL_T f; _PTRC_VAL_T l;
} _PTRC_DEFL_VARS;
```

3.3. Примеры использования прототипа в приложениях

К примерам использования прототипа в приложениях относятся следующие файлы:

```
iterative_point_recovery/
├── …
├── examples/
│   ├── Makefile ..... сценарий сборки примеров.
│   ├── point_recovery_kernel.clc .....
│   │                  файл, содержащий определение ядерной GPGPU-функции,
│   │                  вызывающей _ptrc_recoverPoint().
│   ├── point_recovery-C11.c .....
│   │                  файл, содержащий исходный код консольного приложения,
│   │                  вызывающего функцию _ptrc_recoverPoint().
│   └── point_recovery-OpenCL.c .....
│                      файл, содержащий исходный код консольного прило-
│                      жения, разворачивающего инфраструктуру OpenCL
│                      и запускающего ядерную GPGPU-функцию из файла
│                      point_recovery_kernel.clc.
└── …
```

Для сборки примера на языке C стандарта C11 необходимо, находясь в каталоге `iterative_point_recovery/examples`, выполнить команду

```
make c11
```

Если прототип установлен не в каталог по умолчанию, то при сборке необходимо указывать актуальный путь (префикс) в переменной окружения `IPR_PATH`. Если команда `make` завершена без ошибок, то в каталоге `iterative_point_recovery/examples/build` появится файл `ptrc_example-c11`. Запуск данного файла приведёт к выводу входных данных и результирующего приближения, рассчитанного функцией `_ptrc_recoverPoint()`. При этом в переменной окружения `LD_LIBRARY_PATH` должен быть путь к файлу `libIterPntRcv.so.0.0` (по умолчанию `~/opt/iterative_point_recovery/lib`).

Несколько иначе осуществляется сборка прототипа для OpenCL-диалекта C. Предполагается, что установлен пакет OpenCL_helpers, и прототип собран (см. разд. 3.2.1) с указанием индекса устройства GPGPU. Тогда выполнение команды

```
make GPGPU_DEV_IDX=N opencl
```

(где *N* — тот же индекс устройства GPGPU, который был указан при сборке прототипа) приведёт к появлению в каталоге `iterative_point_recovery/examples/build` файла `ptrc_example-ocl-`*`имя_устройства_GPGPU`*. Запуск данного файла приведёт к выводу входных данных и результирующего приближения, рассчитанного функцией `_ptrc_recoverPoint()`. При этом в переменной окружения `LD_LIBRARY_PATH` должен быть путь к файлу

liboclh.so.0.0 (по умолчанию ~/opt/oclh/lib). В ходе выполнения файла ptrc_example-ocl-*имя_устройства_GPGPU* ведётся журнал в файле ipr_oclh.log, где отображаются события и действия связанные с OpenCL инфраструктурой.

Сами файлы содержащие примеры использования, не превышают 50-ти строк кода достаточно прозрачного для дальнейшей интеграции прототипа в приложения.

3.4. Проверка и оценка прототипа. Визуализация результатов

3.4.1. Проверочный комплект

Проверочный комплект состоит из следующих файлов:

```
iterative_point_recovery/
├── …
├── test_suite/
│   ├── Makefile ..... сценарий сборки проверочного комплекта.
│   ├── ptrc_test_settings.h ..... файл, содержащий настройки проверочного комплекта.
│   └── ptrc_test.c ... файл, содержащий исходный код проверочного комплекта.
└── …
```

Проверочный комплект реализован только для проверки алгоритма на языке C стандарта C11, исходя из предположения, что алгоритм на OpenCL-диалекте C идентичен за исключением округлений и случаев оптимизации обработки числе с плавающей запятой.

Для сборки примера на языке C стандарта C11 необходимо, находясь в каталоге iterative_point_recovery/test_suite, выполнить команду

```
make stable
```

или

```
make debug
```

если необходима сборка с отладочной информацией. Если команда make завершена без ошибок, то в каталоге iterative_point_recovery/test_suite/build появится файл ptrc_test. Запуск данного файла приведёт к выполнению процедуры проверки в соответствии с настройками из файла ptrc_test_settings.h, которые описаны ниже.

При проверке одного случая проверочный комплект выбирает псевдослучайные точки в пространстве $\mathbb{R}^n$, затем рассчитывает расстояния от точек до последней из них и вносит псевдослучайную погрешность в рассчитанные расстояния. Таким образом формируются данные, где первые точки – центры сфер, расстояния с погрешностью – радиусы сфер, а последняя точка – искомая. Этот блок данных передаётся функции восстановле-

ния точки `_ptrc_recoverPoint()`, после чего рассчитывается расстояние от полученного приближения до последней псевдослучайной точки, что интерпретируется как ошибка приближения. Такие параметры как размерность пространства, интервал выбора псевдослучайных точек, количество известных сфер и максимальная погрешность датчика-источника расстояния задаются в настройках проверочного комплекта.

Сводка настроек проверочного комплекта. Настройки проверочного комплекта хранятся в файле `ptrc_test_settings.h` и представляют собой накроопределения на языке C, в связи с чем после изменения настроек необходима пересборка проверочного комплекта в порядке описанном в разделе 3.4.1.

`_PTRC_TS_DIMENSIONALITY`
: размерность пространства.

`_PTRC_TS_NUM_OF_SPHERES`
: количество известных сфер (пар «центр-радиус»).

`_PTRC_TS_NUM_OF_RUNS`
: количество начальных приближений, от которых будет выполняться алгоритм. Соответственно, данное количество точек равно количеству запусков алгоритма. Рекомендуется использовать размерность пространства, увеличенную на единицу.

`_PTRC_TS_RANGE_MULTIPLIER`
: множитель интервала, из которого выбираются проверочные точки. Так, если `_PTRC_TS_RANGE_MULTIPLIER` определён как `1.0f`, то точки принадлежат интервалу $[-1; 1]$ по каждой оси координат. При изменении множителя концы интервала умножаются на данное число.

`_PTRC_TS_MIN_SENSOR_UNCERTAINTY`
: начальная относительная случайная инструментальная погрешность датчика-источника расстояний, которая будет использована при проверке.

`_PTRC_TS_MAX_SENSOR_UNCERTAINTY`
: конечная относительная случайная инструментальная погрешность датчика-источника расстояний, которая будет использована при проверке .

`_PTRC_TS_SENSOR_UNCERTAINTY_INCREMENT`
: приращение погрешности датчика-источника расстояний.

`_PTRC_TS_NUM_OF_CASES_FOR_ONE_INCREMENT`
: количество частных случаев, которые будут проверены для одного значения погрешности датчика-источника расстояний.

`_PTRC_MAX_ITERATIONS_ON_SPHERE`
: переопределение значения w условия `Z` (см. разд. 2.3.3 (10), а также разд. 3.2.2).

`_PTRC_TS_STATISTICS_FILENAME`
: имя файла, в который будет сохранена итоговая статистика результатов проверки. Данный файл используется для визуализации результатов (см. разд. 3.4.2).

`_PTRC_TS_STATISTICS_DISCRETISATION_STEP`
: частота дискретизации значений ошибки для расчёта статистики. Так, если частота дискретизации определена как `0.001f`, тогда ошибка приближения 0,00345 ... рассматривается в статистике как 0,003.

`_PTRC_TS_FLOAT_OUTFORM`
: формат вывода чисел с плавающей запятой. Соответствует преобразованиям функции `printf()` из стандартной библиотеки языка C.

`_PTRC_TS_DATA_OUTPUT_MODE`
: значение данного макроопределения несущественно. В случае если данное макроопределение определено, будет производится машиночитаемый вывод следующих значений разделённых пробельными символами (значения указаны в порядке вывода):

  - погрешность датчика-источника расстояний,
  - значение w (см. разд. 2.3.3 (10) и 3.2.2),
  - размерность пространства,
  - количество известных сфер,
  - количество проверяемых начальных приближений,
  - координаты центров известных сфер,
  - неточные радиусы известных сфер (расстояния до искомой точки);

  затем выводится блок записей итераций, каждая запись состоит из следующих значений разделённых пробельными символами:

  - номер итерации,
  - координаты нового приближения,
  - качество нового приближения λ (см. разд. 2.3.2).

Две последние положительные записи блока итераций отличаются номером итерации на 2, и это значит, что последняя запись с положительным номером итерации есть результат уточнения приближения ψ (см. разд. 2.3.4) и его качество λ; предпоследняя запись в блоке итераций имеет номер итерации -1 и является окончательным приближением, выбранным из приближений полученных из разных начальных точек; последняя запись в блоке итераций имеет номер итерации -2 и содержит координаты «настоящей» искомой точки, а последнее число это эвклидово расстояние до наилучшего выбранного приближения (т. н. ошибка). Данные значения в таком порядке машиночитаемы и могут быть использованы для визуализации шагов алгоритма с помощью сценариев из раздела 3.4.3.

`_PTRC_TS_VERBOSE_MODE`

значение данного макроопределения несущественно. В случае если данное макроопределение определено, будет производится человекочитаемый вывод о действиях алгоритма при работе функции `_ptrc_recoverPoint()`, что позволяет убедиться в соответствии функции `_ptrc_recoverPoint()` описанию алгоритма. Данная настройка сделана исключительно в демонстрационных целях. Рекомендуется использовать её только в случае проверки единственного частного случая. Не рекомендуется определять данное макроопределение в любых других случаях, так как это приводит к объёмному избыточному выводу.

3.4.2. Регресс-оценка прототипа

Визуализация регресс-оценки прототипа реализована в следующих файлах:

```
iterative_point_recovery/
├── …
├── visualisation/
│   ├── Scilab/
│   │   ├── …
│   │   ├── ptrc_statistics.sce ..... сценарий Scilab для визуализации результата работы проверочного комплекта для регресс-оценки алгоритма.
│   │   └── …
│   ├── Wolfram_Mathematica/
│   │   ├── …
│   │   ├── ptrc_statistics.wls ..... сценарий Wolfram Mathematica для визуализации результата работы проверочного комплекта для регресс-оценки алгоритма.
│   │   └── …
│   ├── example_stat_report.txt ..... демонстрационные данные.
│   └── example_stat_report0.txt ..... демонстрационные данные.
└── …
```

Визуализация регресс-оценки реализована для пакетов Scilab и Wolfram Mathematica, описание приводится для пакета Scilab, однако всё написанное справедливо и для сценария Wolfram Mathematica, за исключением прямо оговорённых отличий.

Регресс-оценка предназначения для общей статистической оценки изменения корректности сформированного алгоритмом приближения после внесения изменений в параметры алгоритма или изменения его отдельных частей. Для регресс-оценки необходимо, используя проверочный комплект, сформировать два файла: первый – содержит статистические данные для текущей версии алгоритма, а второй – для изменённой.

Рекомендуемые настройки проверочного комплекта для регресс-оценки:

```
#define _PTRC_TS_DIMENSIONALITY                   по условиям испытаний
#define _PTRC_TS_NUM_OF_SPHERES                   по условиям испытаний
#define _PTRC_TS_NUM_OF_RUNS                      по условиям испытаний
#define _PTRC_TS_RANGE_MULTIPLIER                 1.0f
#define _PTRC_TS_MIN_SENSOR_UNCERTAINTY           0.0f
#define _PTRC_TS_MAX_SENSOR_UNCERTAINTY           0.5f
#define _PTRC_TS_SENSOR_UNCERTAINTY_INCREMENT     0.01f
#define _PTRC_TS_NUM_OF_CASES_FOR_ONE_INCREMENT   10000ul
#define _PTRC_MAX_ITERATIONS_ON_SPHERE            100ul
#define _PTRC_TS_STATISTICS_FILENAME              ''stat_report.txt''
#define _PTRC_TS_STATISTICS_DISCRETISATION_STEP   0.001f
#define _PTRC_TS_FLOAT_OUTFORM                    ''%.7f''
// #define _PTRC_TS_DATA_OUTPUT_MODE
// #define _PTRC_TS_VERBOSE_MODE
```

После установки настроек проверочного комплекта, необходимо осуществить его сборку в порядке указанном в разделе 3.4.1 и запустить полученный исполняемый файл. В результате будет сформирован файл статистики `stat_report.txt`, который сохраняется отдельно как данные о текущей версии алгоритма. В демонстрационных целях для размерности 5 и количества известных сфер 6 сформирован такой файл и сохранён как

`iterative_point_recovery/visualisation/example_stat_report0.txt`

Затем необходимо внести желаемые изменения в алгоритм или его настройки, пересобрать проверочный комплект и снова запустить. Так, в разделе 1 был выражен тезис «*...в прикладных задачах часто есть возможность получить данные не с n* + 1 *датчиков, дающих неточные значения, а с большего их (датчиков) количества ... дополнительная информация может (и должна) улучшать приближение искомой точки...*». Для демонстрации справедливости этого тезиса для размерности 5 и количества известных сфер 9 сформирован файл статистики и сохранён как

`iterative_point_recovery/visualisation/example_stat_report1.txt`

Иными словами, далее в примерах сравнивается точность работы алгоритма при поиске точки в пятимерном пространстве при 6-ти и 9-ти известных сферах (парах «точка-расстояние»).

На данном этапе доступны два файла: первый – со статистической оценкой исходной версии прототипа (`example_stat_report0.txt`); второй – со статистической оценкой изменённой версии прототипа (`example_stat_report1.txt`). Теперь в файле (для сценария Scilab):

`visualisation/Scilab/ptrc_statistics.sce`

либо (для сценария Wolfram Mathematica):

`visualisation/Wolfram_Mathematica/ptrc_statistics.wls`

необходимо изменить значения переменных

```
dataFile0=''../example_stat_report0.txt'';
dataFile1=''../example_stat_report1.txt'';
```

на актуальные. Здесь `dataFile0` – путь к первому файлу с данными текущей версии прототипа, и, соответственно, `dataFile1` – путь к второму файлу с данными изменённого прототипа. После чего сохраните файл и выполните его командой (для сценария Scilab):

```
scilab -f ptrc_statistics.sce -quit
```

либо (для сценария Wolfram Mathematica):

```
wolframscript -f ptrc_statistics.wls
```

Для сценария Scilab можно не указывать ключ `-quit`, а в дальнейшем перезапускать сценарий из открытой оболочки Scilab. Для сценария Wolfram Mathematica, чтобы сохранить сценарий открытым в оболочке с возможностью перезапуска откройте его командой:

```
Mathematica ptrc_statistics.wls
```

В результате выполнения данного сценария будут сохранены файлы `errors.png`, `errors.svg` и `solves.png`, `solves.svg`.

Визуализация регресс-оценки по ошибкам приближений. Рассмотрим для начала файл `errors.png` (или `svg` в векторном представлении). Содержимое файла выглядит примерно как на рис. 2. Сплошная синяя линия описывает математическое ожидание величины ошибки для условий описанных в легенде к графику, в приведённом примере это: размерность 5, известных сфер 6, множитель интервала выбора точек 1.0, количество проверенных начальных приближений 6; погрешность датчика при оценке изменялась от 0 до 0.5 с шагом приращения 0.01, для каждой погрешности было испытано 10000 случаев, частота дискретизации значений ошибки 0.001. Как и было указано в разделе 1 «*алгоритм должен демонстрировать прямую корреляцию качества своей работы и точности датчиков выдающих расстояния R, а в таком случае оценка расстояния между алгоритмически вычисленной и искомой точкой будет не оценкой алгоритма, а оценкой точности датчиков, причём косвенной*», что и можно наблюдать в данном примере, так как видно, что математическое ожидание ошибки растёт с ростом погрешности датчика, однако очевидно, чем алгоритм лучше, тем математической ожидание ошибки при оценке должно быть ниже. Светло-синяя пунктирная линия это, согласно легенде, сумма математического ожидания с одним среднеквадратичным (стандартным) отклонением, чем она ближе к математическому ожиданию, тем алгоритм лучше. Светло-синие точечные линии это минимумы и максимумы, соответственно. С одной стороны, чем ближе минимумы и максимумы к математическому ожиданию, тем лучше, но с другой стороны большинство таких максимизированных выбросов обусловлены не качеством алгоритма как такового, а ситуациями, когда выбранные для оценки точки находятся близко к одной гиперплоскости, что приводит к возникновению нескольких областей с примерно

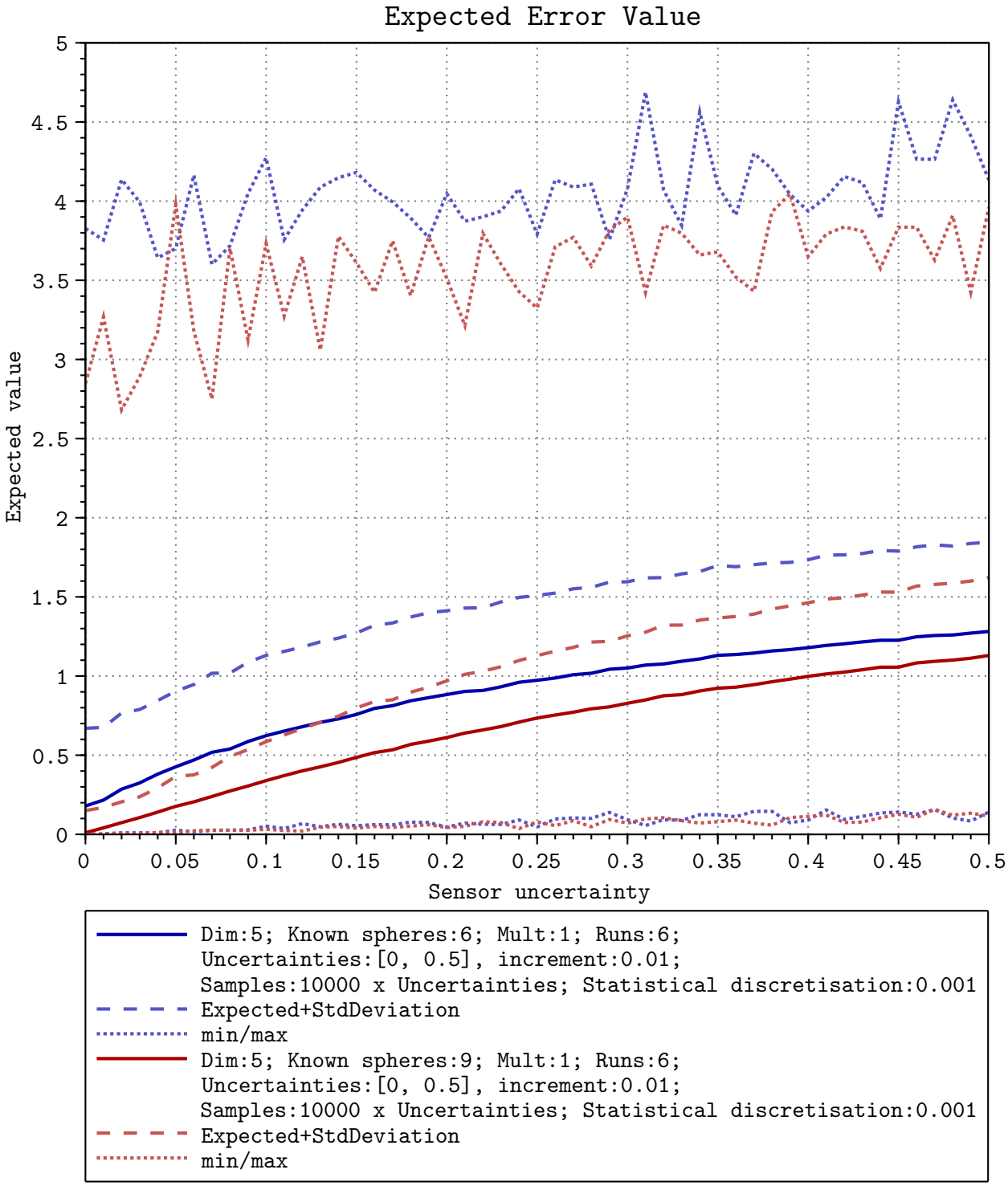

Рис. 2: Статистика ошибок приближений.

одинаковыми качествами приближений и неопределённости в выборе (подобный пример будет рассмотрен в разделе 3.4.3). Представляется очевидным, что, в случае если известные точки принадлежат одной гиперплоскости, местонахождение искомой точки становится полностью неопределённым, поэтому при решении практических задач необходимо следить, чтобы датчики-источники расстояний по возможности не принадлежали и не находились близко к одной гиперплоскости.

Сплошная красная, пунктирная и точечная светло-красные линии отображают, соответственно, математическое ожидание, сумму математического ожидания и среднеквадратичного отклонения, минимумы и максимумы для испытаний второго варианта алгоритма или условий. Так, из легенды к графику можно видеть, что условия для испытаний, при которых получены красные показатели, отличаются только количеством известных сфер. По сравнению с 6-ю известными сферами 9 известных сфер снижают математическое ожидание ошибки и уменьшают среднеквадратичное отклонение ошибки.

Увеличение точности приближений с ростом количества известных сфер ожидаемо, но не бесплатно, стоимость такого увеличения точности рассматривается в следующем параграфе.

Визуализация регресс-оценки по количеству произведённых действий. Для данного варианта алгоритма самым затратным повторяющимся шагом является расчёт формулы χ (разд. 2.1.2 (5) и (6)) с последующим вычислением качества λ (разд. 2.3.2 (9)) для двух приближений при осуществлении выбора. Так как для каждого вычисления χ производится вычисление λ, в проверочном комплекте в глобальной переменной `fSolves` осуществляется подсчёт таких шагов для каждого испытанного случая. Полученная статистика отображается на графике в файле `solves.png` (или `svg` в векторном представлении). Содержимое файла выглядит примерно как на рис. 3. Цветовые обозначения и формы линий аналогичны таковым при визуализации статистики ошибок приближений. Так линии синего и светло-синего цветов соответствуют данным первого набора испытаний (эталонного), а линии красного и светло-красного цвета соответствуют данным второго набора испытаний (для изменённого алгоритма или условий). Сплошная линия в контексте данного графика обозначает математическое ожидание количества шагов (вычислений χ) к погрешности датчика-источника расстояний, пунктирная линия показывает сумму математического ожидания и стандартного отклонения количества шагов, точечные линии отображают минимумы и максимумы соответственно их расположению.

Из представленного примера видно, что при изменении условий исполнения алгоритма от 6-и к 9-и известным сферам количество расчётов увеличилось. Иными словами, изменение количества известных сфер делает найденное приближение точнее, но увеличивает время выполнения алгоритма для одного случая. Необходимо учитывать данное обстоятельство при формировании входящих данных для алгоритма, так, например, при ограничении времени и наличии нескольких датчиков-источников расстояний имеет смысл не использовать все известные сферы, а выбирать, только $n + 1$ тех из них, центры которых, которые наиболее удалены от одной гиперплоскости. Окончательный выбор входных данных должен производится исходя из отмеченного обстоятельства зависимости точности и вре-

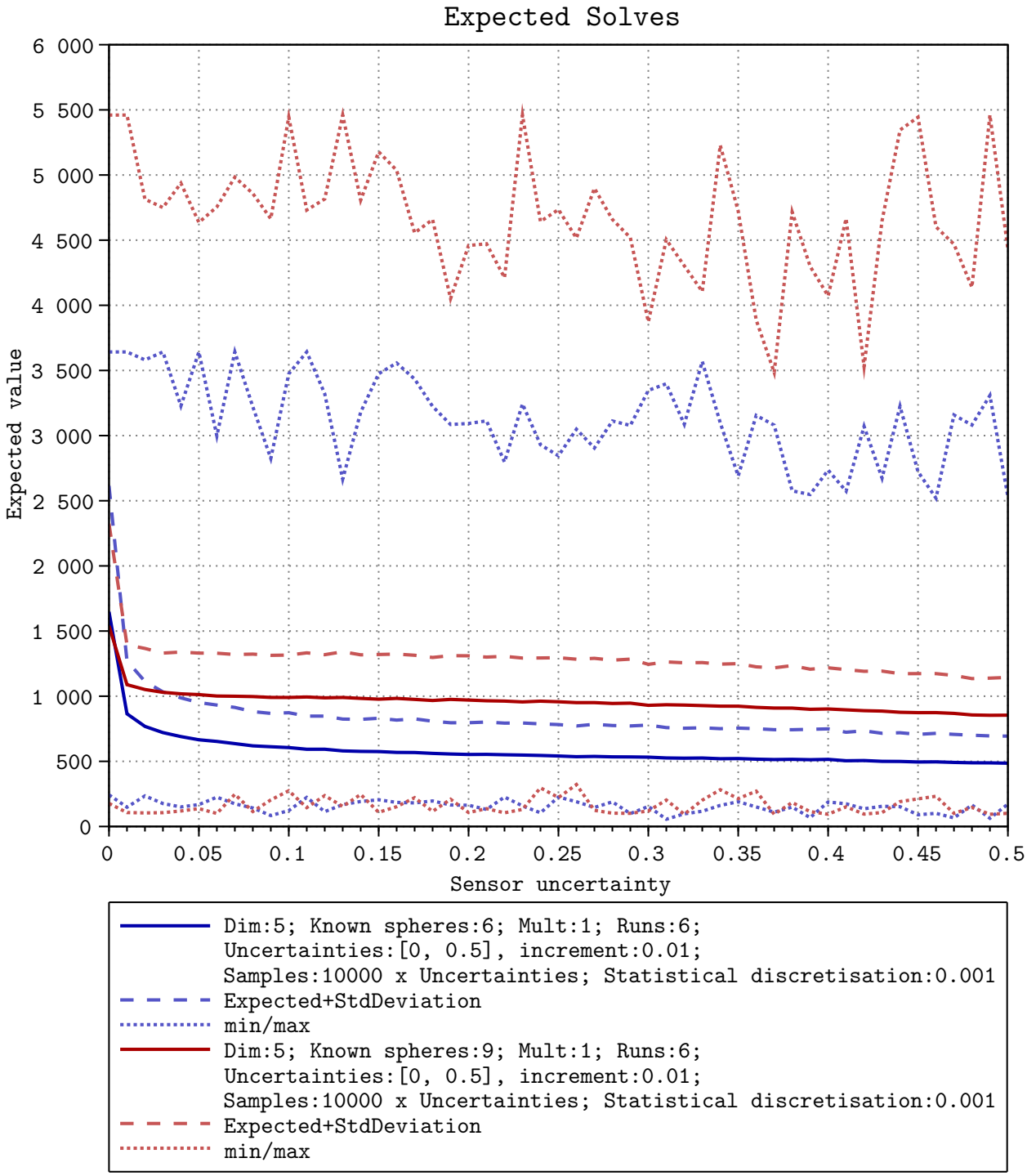

Рис. 3: Статистика количеств расчётов формулы χ.

мени исполнения, объективных ограничений и рисков обстановки исполнения алгоритма.

Данные по каждому испытанию. Конкретные предполагаемые условия эксплуатации алгоритма могут потребовать дополнительной обработки и более тонких манипуляций со статистикой испытаний, поэтому в файле со статистикой сохраняются значения ошибок и количество шагов для каждого испытанного случая.

Значения ошибок представлены в виде:

```
Sensor_uncertainty Raw_Deltas
0.0000000 0.0000018 0.0000024 0.0000140 ...
0.0100000 0.0380290 0.0628870 0.0381434 ...
0.0200000 0.1352121 0.2248735 0.0489258 ...
...       ...       ...       ...
```

Здесь число в первом столбце это значение погрешности датчика-источника расстояний, далее до конца строки значения ошибок для каждого испытанного случая. При этом надо иметь в виду, что могут возникать ситуации, когда количество значений ошибки меньше, чем `_PTRC_TS_NUM_OF_CASES_FOR_ONE_INCREMENT`, так как не все точки могут быть восстановлены (см. замечание на стр. 6).

Значения количеств шагов представлены в виде:

```
Sensor_uncertainty Raw_Solves
0.0000000 2566 2885 710 ...
0.0100000 866  546  994 ...
0.0200000 997  935  570 ...
0.0300000 1231 2668 463 ...
...       ...  ...  ...
```

Здесь число в первом столбце это значение погрешности датчика-источника расстояний, далее до конца строки значения количеств предпринятых шагов для каждого испытанного случая. При этом надо иметь в виду, что при малых значениях w (см. разд. 2.3.3 (10) и замечание к разделу 3.2.2) большинство значений не будет превышать (wm) + const.

Исходя их этих данных можно строить иные показатели качества алгоритма, например, композитный показатель отношения точности к количеству шагов или иные согласно предполагаемым условиям эксплуатации алгоритма.

3.4.3. Визуализация одного частного случая

Визуализация частного случая работы алгоритма реализована в следующих файлах:

- `iterative_point_recovery/`
  - …
  - `visualisation/`
    - `Scilab/`
      - …
      - `ptrc_one_sample_vis-2d-anim.sce` — сценарий Scilab для анимированной визуализации двухмерного случая работы алгоритма.
      - `ptrc_one_sample_vis-2d.sce` — сценарий Scilab для визуализации двухмерного случая работы алгоритма.
      - `ptrc_one_sample_vis-3d.sce` — сценарий Scilab для визуализации трёхмерного случая работы алгоритма.
      - `sample_data_parser.sce` — сценарий Scilab для чтения и разбора данных частного случая, сформированных проверочным комплектом.
      - …
    - `Wolfram_Mathematica/`
      - …
      - `ptrc_one_sample_vis-2d-anim.wls` — сценарий Wolfram Mathematica для анимированной визуализации двухмерного случая оценки алгоритма.
      - `ptrc_one_sample_vis-2d.wls` — сценарий Wolfram Mathematica для визуализации двухмерного случая оценки алгоритма.
      - `ptrc_one_sample_vis-3d.wls` — сценарий Wolfram Mathematica для визуализации трёхмерного случая оценки алгоритма.
      - `sample_data_parser.wl` — сценарий Wolfram Mathematica для чтения и разбора данных частного случая, сформированных проверочным комплектом.
      - …
    - `one_case_data-2d_#.txt` — демонстрационные данные.
    - `one_case_data-3d_#.txt` — демонстрационные данные.
  - …

Визуализация регресс-оценки реализована для пакетов Scilab и Wolfram Mathematica, описание приводится для пакета Scilab, однако всё написанное справедливо и для сценария Wolfram Mathematica, за исключением прямо оговорённых отличий.

Визуализация одного частного случая не имеет значительной практической пользы и сделана в разъяснительных целях для понимания алгоритма лицами, склонными к пространственно-геометрическому мышлению больше, нежели к синтаксическому. Учитывая, что человек способен достаточно полно воспринимать только двухмерное пространство, визуализация частного случая сделана только для двухмерного и ограниченно для трёхмерного случая. Теоретически можно сделать визуализации случаев и больших размерностей, разбивая их на проекции, но так как в целом практической пользы от такой визуализации нет или крайне мало, для случаев с размерностью выше трёх сценарии визуализации не создавались.

Визуализация двухмерного частного случая работы алгоритма. Рекомендуемые настройки проверочного комплекта для визуализации одного двухмерного частного случая работы алгоритма:

```
#define _PTRC_TS_DIMENSIONALITY                  2ul
#define _PTRC_TS_NUM_OF_SPHERES                  по условиям испытаний
#define _PTRC_TS_NUM_OF_RUNS                     3ul
#define _PTRC_TS_RANGE_MULTIPLIER                1.0f
#define _PTRC_TS_MIN_SENSOR_UNCERTAINTY          по условиям испытаний
#define _PTRC_TS_MAX_SENSOR_UNCERTAINTY     равно предыдущему значению
#define _PTRC_TS_SENSOR_UNCERTAINTY_INCREMENT    0.01f
#define _PTRC_TS_NUM_OF_CASES_FOR_ONE_INCREMENT  1ul
#define _PTRC_MAX_ITERATIONS_ON_SPHERE           100ul
#define _PTRC_TS_STATISTICS_FILENAME             ''stat_report.txt''
#define _PTRC_TS_STATISTICS_DISCRETISATION_STEP  0.001f
#define _PTRC_TS_FLOAT_OUTFORM                   ''%.7f''
#define _PTRC_TS_DATA_OUTPUT_MODE
//#define _PTRC_TS_VERBOSE_MODE
```

После установки настроек проверочного комплекта, необходимо осуществить его сборку в порядке указанном в разделе 3.4.1. Затем из выполните сценарий Scilab командой:

```
scilab -f ptrc_one_sample_vis-2d.sce
```

либо сценарий Wolfram Mathematica командой:

```
Mathematica -f ptrc_one_sample_vis-2d.wls
```

После выполнения сценария останется открытая оболочка и графические окна с визуализацией. В данном случае открытые оболочки остаются специально, чтобы можно было в интерактивных графических окнах масштабировать изображение, так как часть визуализации достаточно мала и неразборчива на общем плане. В сценарии первые строки представляют собой присвоение значений двум переменным

```
fileName=''one_case_data-2d.txt'';
generateNewCase=1;
```

Первая их них `fileName` определяет путь к и имя файла, куда будут сохранены результаты запуска проверочного комплекта, выведенные в стандартный вывод; вторая `generateNewCase` – указывает сценарию производить ли новый запуск проверочного комплекта (значение 1) или работать с файлом `fileName` без его изменения (значение 0).

Для начала рассмотрим случай, когда начальные условия достаточно корректны. Так на рис. 4 показан случай, когда в результате ошибок датчиков-источников расстояний образуется закрытая область. Визуализация строится на фоне тепловой карты значений функ-

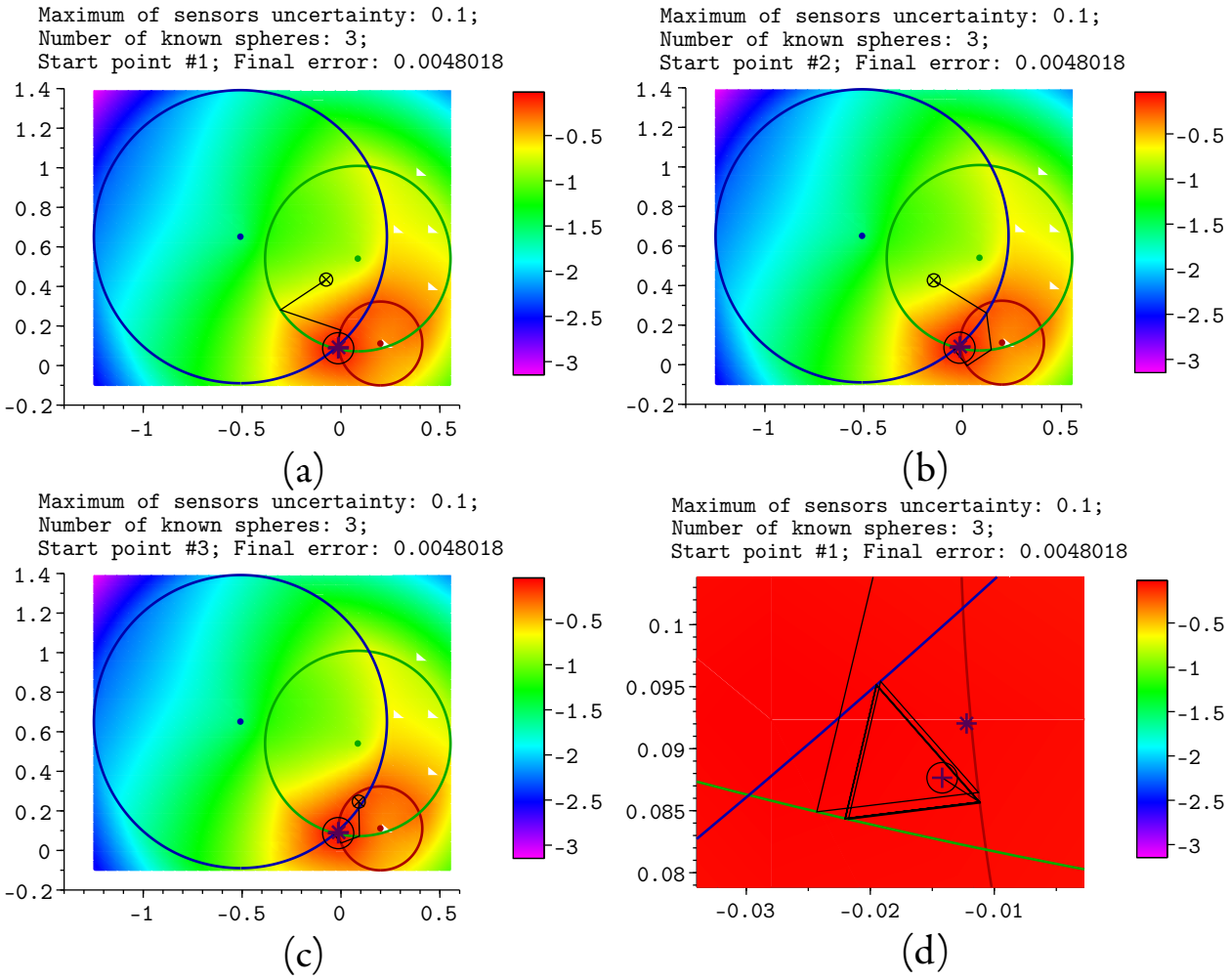

Рис. 4: Визуализация работы алгоритма в случае закрытой области.

ции λ (разд. 2.3.2 (9)). Известные окружности и их центры (расположение датчиков-источников расстояний) обозначены оттенками цветов (в данном случае тёмно-красным, тёмно-зелёным и тёмно-синим). Символом $\otimes$ обозначается начальное приближение. Чёрные линии показывают переход от приближения к приближению в ходе выполнения алгоритма. Последнее приближение для данного начального обведено чёрной окружностью. Фиолетовым символом + обозначается наилучшее приближение выбранное среди всех результатов работы из разных начальных приближений. Фиолетовым символом $*$ обозначается «настоящая» искомая точка. На рис. 4

изображение (a) показывает «путь» улучшения приближений для первого начального приближения и последующие два изображения (b) и (c) для 2-ого и 3-его соответственно. Последнее изображение (d) это увеличенный фрагмент первого изображения, где продемонстрирована замкнутая область, образованная окружностями построенными из неточных расстояний. Можно видеть, как на четвёртом изображении алгоритм попадает в замкнутую область и завершается. В результате, как написано в заголовках изображений, расстояние от найденного приближения до искомой точки составляет 0.0048018.

Аналогично на рис. 5 продемонстрирован случай, когда по расстояниям с ошибками построены сферы образующие незамкнутую (открытую область), где алгоритм находит наилучшее приближение. Завершение работы алгоритма можно видеть на рис. 5 изображение (d).

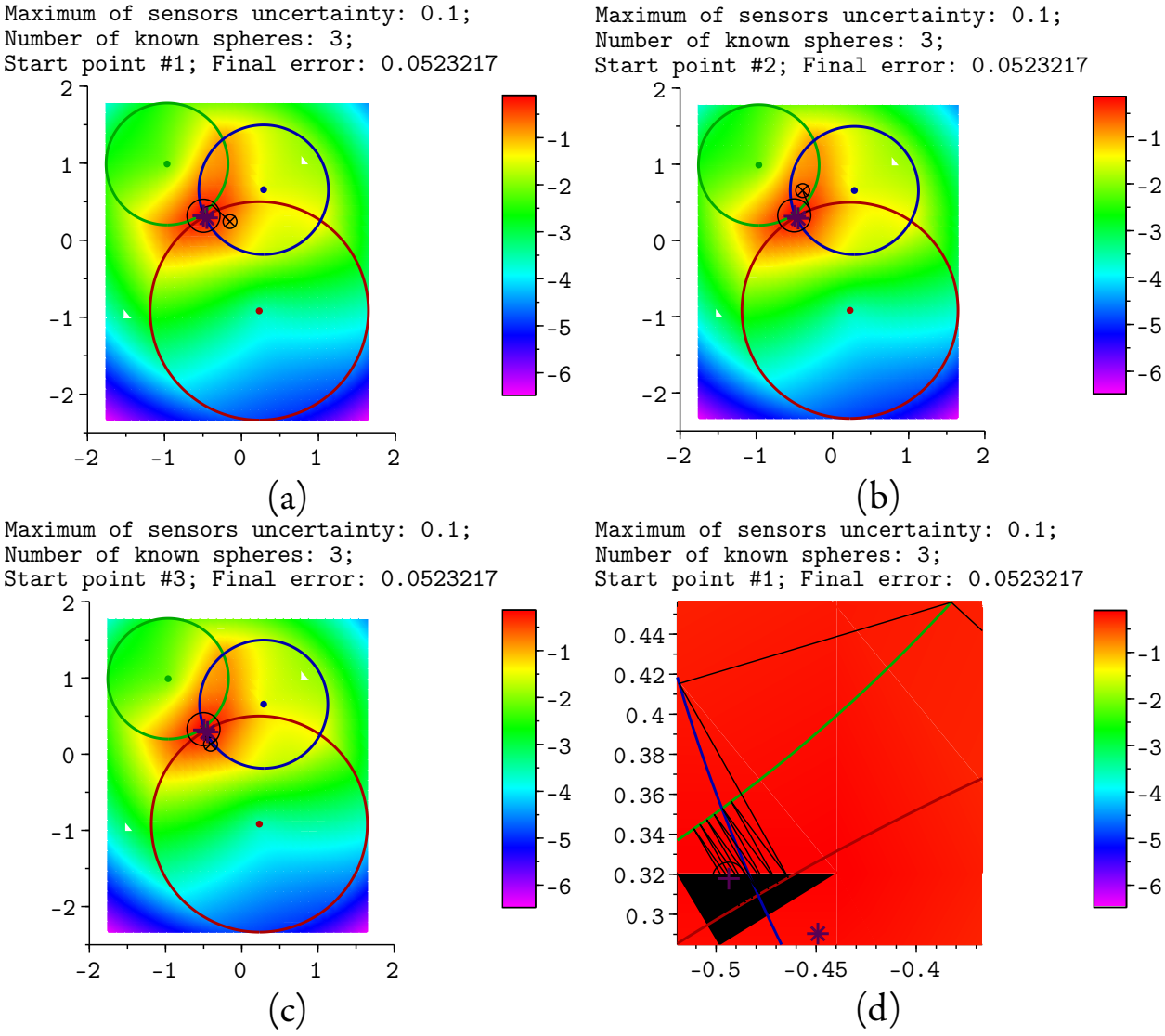

Рис. 5: Визуализация работы алгоритма в случае открытой области.

Можно и далее приводить различные случаи и классифицировать их, включая случаи, когда есть полузакрытая область или случаи, когда окруж-

ности не пересекаются, но в целом в большинстве таких случаев алгоритм ведёт себя правильно и даёт хорошее приближение. Интереснее рассмотреть случаи, когда условия выполнения алгоритма действительно плохие и на тепловой карте явно образованы 2 (иногда больше) экстремумов функции λ, что, как было сказано выше, происходит, если датчики-источники расстояний расположены на одной или близко к одной гиперплоскости. На таких случаях становятся видны преимущества нескольких запусков алгоритма из разных начальных приближений. Так на рис. 6 можно видеть, что датчики-источники расстояний (центры сфер) расположены почти на одной гиперплоскости (прямой для двухмерного случая), что приводит к появлению двух примерно равных экстремумов на севере и юго-востоке изображения, причём искомая точка расположена на юго-востоке. Можно

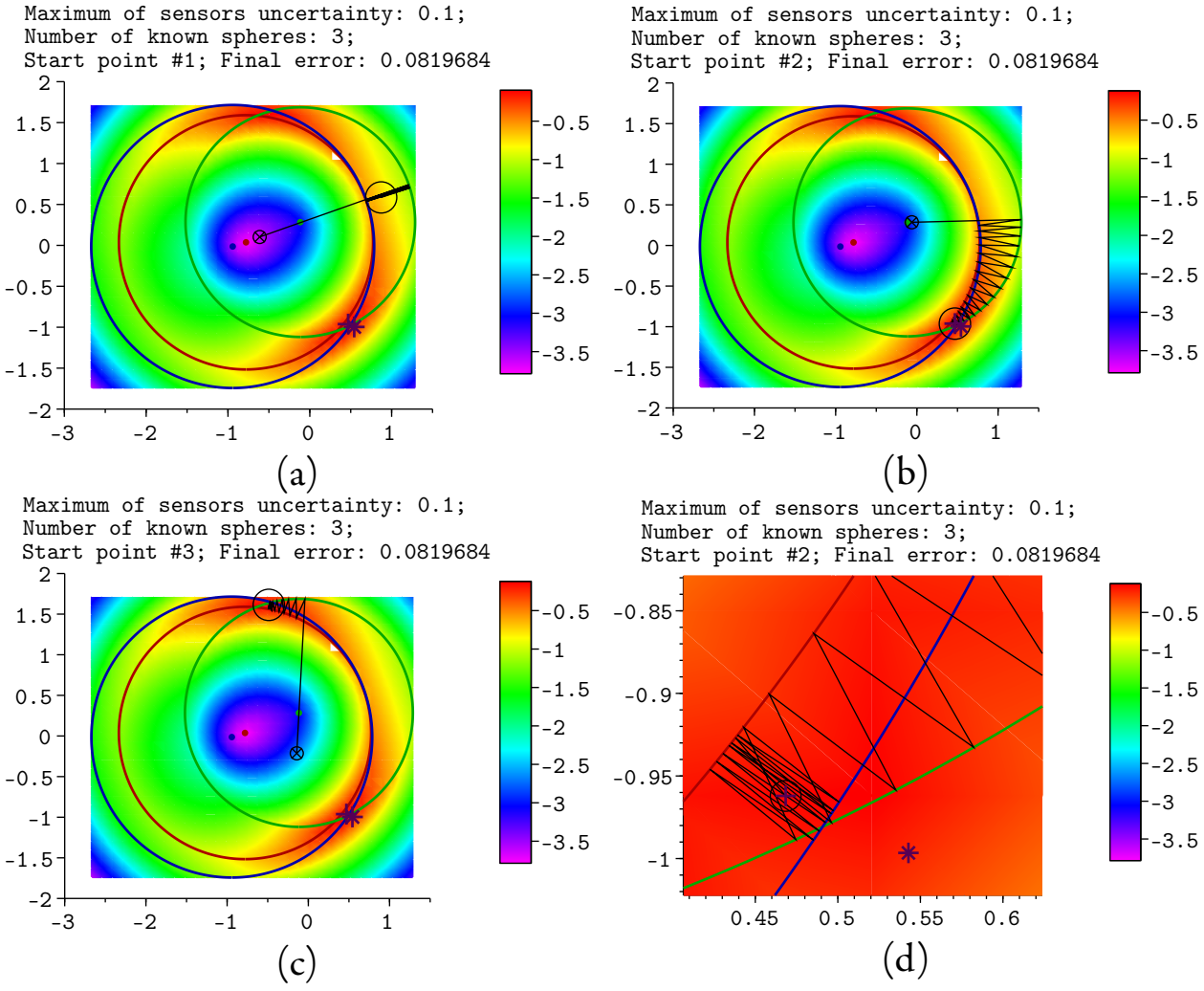

Рис. 6: Визуализация работы алгоритма в случае двух экстремумов функции λ и нахождении «хорошего» приближения.

видеть, что на изображении (a) рис. 6 из первого начального приближения алгоритм приходит в область между двумя экстремумами и в отсутствии улучшения функции λ (разд. 2.3.2 (9)) прекращает поиск, возвращая плохое приближение. Но при проверке второго начального приближения на изоб-

ражении (b) рис. 6, алгоритм приходит в «правильный» экстремум функции λ и возвращает «хорошее» приближение искомой точки, завершение чего можно видеть на масштабированном изображении (d) рис. 6. И при проверке третьего начального приближения алгоритм уходит к «неправильному» северному экстремуму функции λ, где и завершается. И только выбор из этих трёх финальных приближений позволяет выбрать наилучшее, соответствующее изображению (b).

В отдельных случаях, когда датчики-источники расстояний расположены на одной или близко к одной гиперплоскости может образоваться область, в которой значение функции λ выше, чем в области с искомой точкой. В таком случае алгоритм вернёт очень «плохое» приближение. Подобный пример приведён на рис. 7. Интересным в приведённом примере является

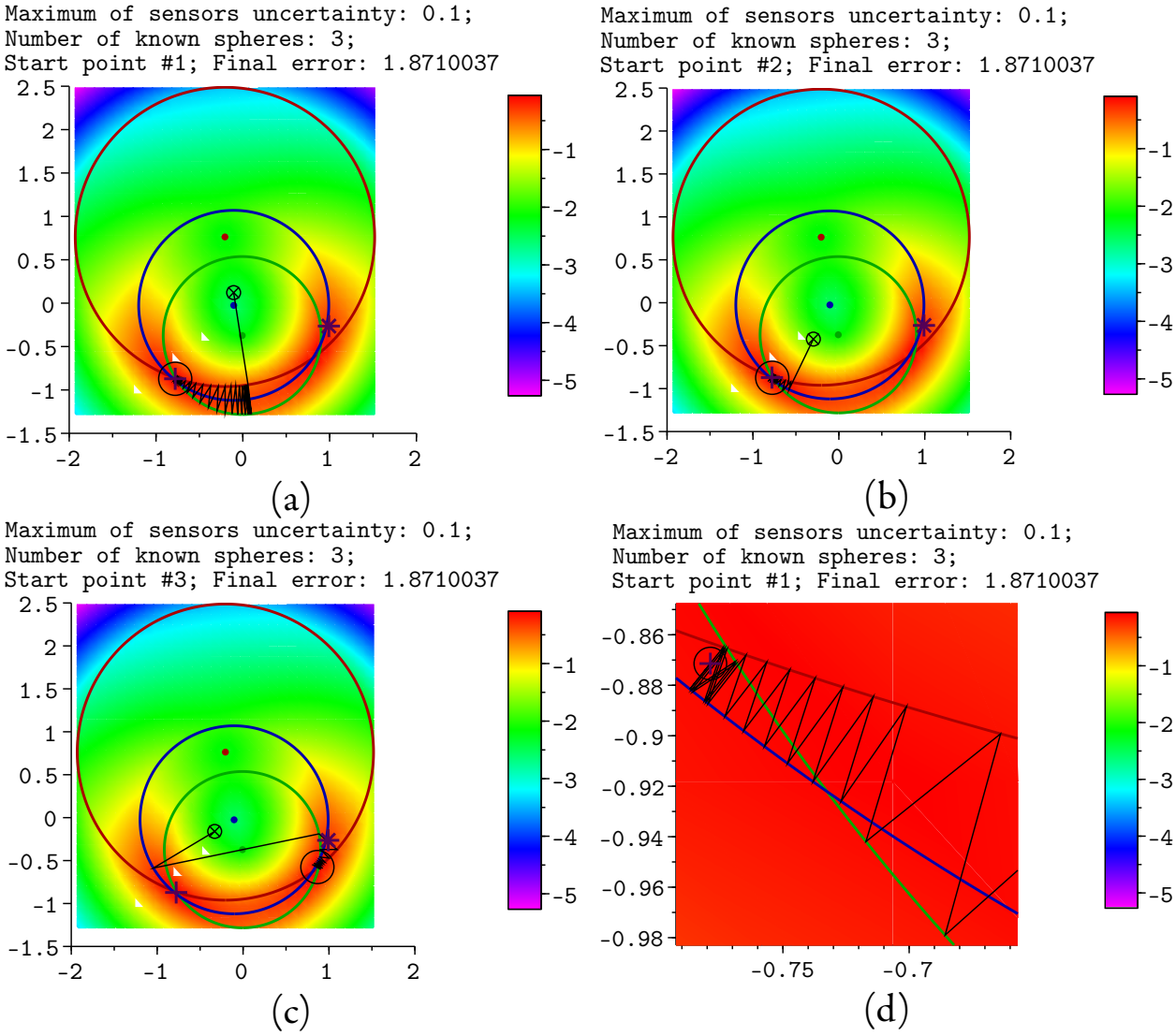

Рис. 7: Визуализация работы алгоритма в случае двух экстремумов функции λ и нахождении ошибочного приближения.

то, что при исполнении алгоритма от третьего начального приближения на изображении (c) видно как алгоритм пришёл в область близкую к искомой

точке, но после сравнения качества финальных первого и третьего приближений было выбрано первое, то есть значение функции λ оказалось выше в экстремуме противоположном области с искомой точкой. Как можно попытаться избегать подобных ситуаций, будет описано в разделе 4.

Анимированная визуализация двухмерного частного случая работы алгоритма. Анимированная визуализация двухмерного частного случая осуществляется с теми же настройками, что и статическая визуализация, описанная в предыдущем параграфе. Сценарий анимированной визуализации использует сценарий статической визуализации, поэтому настройки переменных `fileName` и `generateNewCase` необходимо менять в соответствующих файлах `ptrc_one_sample_vis-2d.sce` или `.wls`. Имя файла с анимацией определено в сценарии `ptrc_one_sample_vis-2d-anim.sce` для Scilab или `.wls` для Wolfram Mathematica переменной

```
animationFileName=''animation.gif'';
```

Сценарий Scilab, создающий анимированный gif-файл, запускается командой:

```
scilab -f ptrc_one_sample_vis-2d-anim.sce -quit
```

а сценарий Wolfram Mathematica командой:

```
Mathematica ptrc_one_sample_vis-2d-anim.wls
```

В результате выполнения данной команды в каталоге появится файл `animation.gif`, где и будет анимированная визуализация одного двухмерного случая.

Сценарий Scilab формирует gif-файл с помощью пакета ImageMagic (https://imagemagick.org), который должен быть установлен и доступен.

Сценарий Wolfram Mathematica формирует gif-файл собственными средствами, но есть нюанс: так как формирование gif-файла производится модулем на Java, при большом количестве кадров сценарий может использовать очень много оперативной памяти вплоть до 200 гигабайт. В связи с изложенным рекомендуется запускать сценарий с ограничением по оперативной памяти, например, в ОС Linux это можно сделать с помощью контрольных групп командами:

```
sudo cgcreate -t user:user -a user:user -g memory:mem2G
echo 2048M > /sys/fs/cgroup/memory/mem2G/memory.limit_in_bytes
```

А затем запустить сценарий при данных ограничениях:

```
cgexec -g memory:mem2G Mathematica ptrc_one_sample_vis-2d-anim.wls
```

Визуализация трёхмерного частного случая работы алгоритма. Рекомендуемые настройки проверочного комплекта для визуализации одного трёхмерного частного случая работы алгоритма:

```
#define _PTRC_TS_DIMENSIONALITY                  3ul
#define _PTRC_TS_NUM_OF_SPHERES                  по условиям испытаний
#define _PTRC_TS_NUM_OF_RUNS                     3ul
#define _PTRC_TS_RANGE_MULTIPLIER                1.0f
#define _PTRC_TS_MIN_SENSOR_UNCERTAINTY          по условиям испытаний
#define _PTRC_TS_MAX_SENSOR_UNCERTAINTY     равно предыдущему значению
#define _PTRC_TS_SENSOR_UNCERTAINTY_INCREMENT    0.01f
#define _PTRC_TS_NUM_OF_CASES_FOR_ONE_INCREMENT  1ul
#define _PTRC_MAX_ITERATIONS_ON_SPHERE           100ul
#define _PTRC_TS_STATISTICS_FILENAME             ''stat_report.txt''
#define _PTRC_TS_STATISTICS_DISCRETISATION_STEP  0.001f
#define _PTRC_TS_FLOAT_OUTFORM                   ''%.7f''
#define _PTRC_TS_DATA_OUTPUT_MODE
//#define _PTRC_TS_VERBOSE_MODE
```

После установки настроек проверочного комплекта, необходимо осуществить его сборку в порядке указанном в разделе 3.4.1. Затем из выполните сценарий Scilab командой:

```
scilab -f ptrc_one_sample_vis-3d.sce
```

либо сценарий Wolfram Mathematica командой:

```
Mathematica -f ptrc_one_sample_vis-3d.wls
```

После выполнения сценария останется открытая оболочка и графические окна с визуализацией. В данном случае открытые оболочки остаются специально, чтобы можно было в интерактивных графических окнах масштабировать изображение, так как часть визуализации достаточно мала и неразборчива на общем плане. В сценарии первые строки представляют собой присвоение значений двум переменным

```
fileName=''one_case_data-3d.txt'';
generateNewCase=1;
```

Первая их них `fileName` определяет путь к и имя файла, куда будут сохранены результаты запуска проверочного комплекта, выведенные в стандартный вывод; вторая `generateNewCase` – указывает сценарию производить ли новый запуск проверочного комплекта (значение `1`) или работать с файлом `fileName` без его изменения (значение `0`).

Для визуализации трёхмерного случая тепловая карта функции λ не строится. В целом визуализация трёхмерного случая весьма неинформативна и после построения требуется посмотреть с нескольких точек обзора, чтобы понять, как происходило улучшение приближений. В сценарии визуализации для Scilab сферы отображаются с помощью сетки, частоту которой регулирует переменная `meshSize`, а в сценарии визуализации для

Wolfram Mathematica поверхности сфер отображаются полностью, но введён регулятор прозрачности поверхности сфер.

Сложность восприятия визуализации трёхмерного случая можно понять из рис. 8.

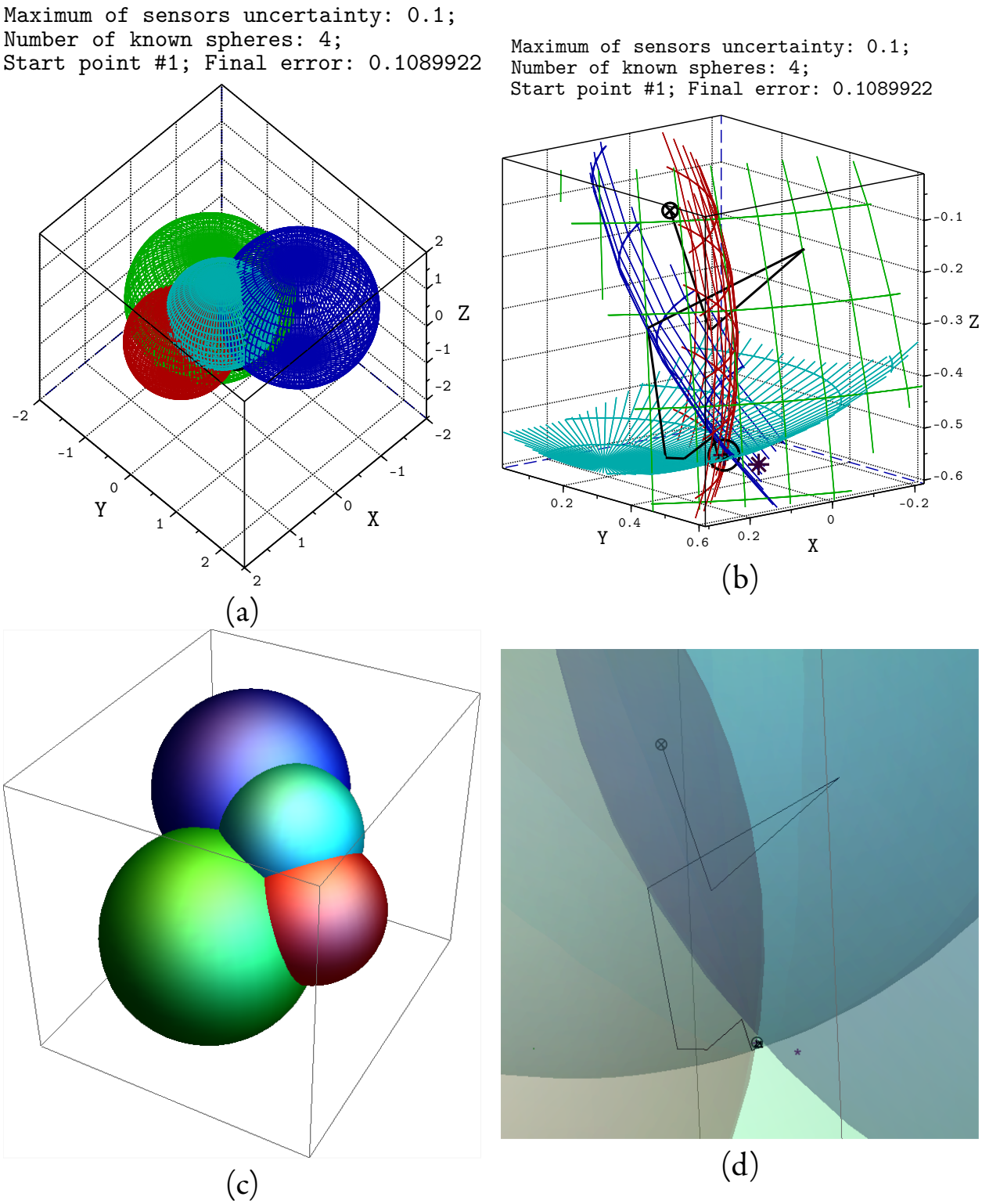

Рис. 8: Визуализация работы алгоритма в трёхмерном случае.
(a), (b) – Scilab. (c), (d) – Wolfram Mathematica.

3.5. Исходный код документации и сопутствующие файлы

Данная документация предоставляется в виде исходных кодов в следующих файлах:

```
iterative_point_recovery/
├── …
├── documentation/
│   ├── fonts/ ..... каталог, содержащий шрифты и их настройки.
│   ├── images/ ..... каталог, содержащий иллюстрации документации и исходные файлы для построения иллюстраций.
│   ├── build_script .. сценарий сборки документации.
│   ├── ipr_doc.cls ... файл с определениями стиля документации.
│   └── iterative_point_recovery_documentation-russian.tex ..... исходный код документации.
└── …
```

Документация собирается отдельно. Для её сборки необходима система вёрстки XeTeX/XeLaTeX или иная TeX/LaTeX-система. Система XeTeX и сопутствующие пакеты поставляются в составе дистрибутива TeX Live (https://www.tug.org/texlive/). Использование системы отличной от XeTeX может потребовать внесения изменений в исходный код документации.

Сама сборка документации осуществляется из каталога `iterative_point_recovery/documentation` запуском сценария сборки

```
./build_script
```

Если в ходе исполнения данного сценария не возникло ошибок, то в каталоге `iterative_point_recovery/documentation/build` появится файл

```
iterative_point_recovery_documentation-russian.pdf
```

с документацией.

При сборке будут использованы шрифты семейства IBM Plex, но можно вернуться к базовому семейству Computer Modern просто раскомментировав строку

```
\input{fonts/font_settings-Computer_Modern.tex}
```

в преамбуле исходного кода документации.

4. Основные направления возможных доработок

При реализации алгоритма для конкретных условий работы прежде всего необходима ревизия предположений из раздела 2.3, так как приведённые предположения могут не соответствовать известным условиям работы алгоритма и требовать пересмотра. В настоящее время они сформулированы достаточно общо, что оставляет маневр для уточнения в конкретных реализациях. При изменении подходов, описанных в разделе 2.3,

рекомендуется проводить регресс-оценку на случайном или специфичном задачам проверочном множестве.

Для избегания ситуаций неправильного выбора приближения в случаях когда, экстремум функции λ (разд. 2.3.2 (9)) находится на удалении от искомой точки, а возможности удовлетворительного выбора датчиков-источников расстояний нет, рассматривается возможность сохранения и выбора из нескольких приближений, исходя из дополнительной информации, например, о предыдущем положении искомой точки.

Содержание