Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий
Кафедра прикладной математики и компьютерной безопасности

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. А. Кытманов
подпись
«____» _____ 2018 г.

БАКАЛАВРСКАЯ РАБОТА

01.03.04 Прикладная математика

Компьютерная реализация элементарной версии метода Рунге
для кубических диофантовых уравнений

Руководитель _____ профессор, д.ф.-м.н. Н. Н. Осипов
подпись, дата

Выпускник _____ А. А. Муравлев
подпись, дата

Красноярск 2018

Реферат

Бакалаврская работа на тему «Компьютерная реализация элементарной версии метода Рунге для кубических диофантовых уравнений» содержит 42 страницы текста, 15 использованных источников и 3 таблицы.

ДИОФАНТОВЫ УРАВНЕНИЯ, МЕТОД РУНГЕ, АЛГОРИТМЫ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

Цель работы: компьютерная реализация пригодного для практического применения алгоритма решения уравнений кубических диофантовых уравнений, удовлетворяющих условию Рунге.

Актуальность работы выражается в том, что компьютерная реализация элементарной версии метода Рунге способствует усовершенствованию современных систем компьютерной алгебры и развивает алгоритмические подходы к решению диофантовых уравнений в теории чисел.

Содержание работы: рассмотрены алгоритмическая и компьютерная реализация элементарной версии метода Рунге для кубических диофантовых уравнений с двумя неизвестными.

Содержание

Введение

Метод Рунге является одним из самых простых общих методов решения диофантовых уравнений с двумя неизвестными (см., например, монографии [3, 9], а также статью [12]). Под *диофантовым уравнением* понимается уравнение вида

$$f(x, y) = 0,$$

где $f(x, y)$ — некоторый полином с целыми коэффициентами. Решения этого уравнения — это пары (x, y) целых чисел, при подстановки которых в уравнение получаются верные числовые равенства. Вообще говоря, найти все решения данного диофантова уравнения может представлять собой трудную математическую проблему. Тем не менее существуют классы диофантовых уравнений, для которых удается предложить эффективные методы их решения. К таким методам и относится упомянутый метод Рунге, предложенный немецким математиком Карлом Рунге еще в конце XIX века.

Оригинальная версия метода Рунге предполагает манипуляции со сложными математическими объектами (типа *рядов Пюизо*, частным случаем которых являются более известные *ряды Лорана*). Несмотря на теоретическую эффективность метода Рунге, при его практической реализации возникают определенные проблемы даже в случае диофантовых уравнений малой степени (3-й или 4-й), и это заставляет искать альтернативные пути.

Элементарная версия метода Рунге для решения кубических диофантовых уравнений с двумя неизвестными впервые была предложена в статье [1]. Там же было отмечено, что алгоритмическая реализация элементарного подхода нуждается в дальнейшей детализации и не представляется очевидной. Такая детализация была дана позже в статье [10]. Для диофантовых уравнений 4-й степени с двумя неизвестными также есть элементарная версия метода Рунге (см. статью [2]), где проблема компьютерной реализации будет, по-видимому, еще сложней.

В современных системах компьютерной алгебры (СКА) общего назначения (типа Maple [14], Mathematica [15] и т.п.) отсутствуют пакеты или модули, которые позволяли бы решать подобные диофантовы уравнения: обычно СКА или решает уравнение некорректно, или же просто отказывается его решать. Это связано с тем, что имеющиеся теоретические оценки (см., например, [13, 11]) для решений диофантовых уравнений, подпадающих под метод Рунге, оказываются практически непригодными, т. е. компьютерная реализация алгоритма решения диофантовых уравнений на основе этих оценок была бы бесполезной из-за слишком больших констант (подробности см., например, в статье [10]). Таким образом, актуальность настоящей работы выражается в том, что компьютерная реализация элементарной версии метода Рунге способствует усовершенствованию современных систем компьютерной алгебры и развивает алгоритмические подходы к решению диофантовых уравнений в теории чисел.

Для кубических диофантовых уравнений естественно поставить задачу об оптимизации алгоритма решения, учитывающей специфику уравнения. Основной целью работы является компьютерная реализация пригодного для практического применения алгоритма решения уравнений кубических диофантовых уравнений, удовлетворяющих условию Рунге.

Для достижения поставленной цели были сформулированы следующие задачи.

1. Предложить компьютерную реализацию на языке программирования Python элементарной версии метода Рунге для кубических уравнений вида

$$x(Ax^2 + Bxy + Cy^2) + a_1x^2 + a_2xy + 0 \cdot y^2 + a_4x + a_5y + a_6 = 0,$$

(Это фактически общий случай, поскольку с помощью линейной замены неизвестных к такому виду можно свести любое кубическое уравнение с двумя неизвестными.)

2. Оптимизировать предложенную компьютерную реализацию алгоритма

решения общего кубического уравнения.

3. Оптимизировать алгоритм решения для уравнений частного вида:

$$x(y^2 - 2x^2) + Hx + y + 1 = 0,$$
$$x(y^2 - 2x^2) + x + y + H = 0,$$
$$x(y^2 - 2x^2) + x + Hy + 1 = 0,$$

где H — целочисленный параметр. С помощью оптимизированной версии алгоритма получить статистику количества решений этих уравнений в достаточно широком диапазоне изменения параметра H.

Основные результаты работы: задачи 1 и 3 полностью решены. Попутно были получены некоторые дополнительные теоретические результаты относительно оценки величины и количества решений уравнений из п. 3.

1 Алгоритмизация метода Рунге для кубических уравнений

В работе рассматриваются кубические диофантовые уравнения вида

$$f(x,y) = f_3(x,y) + f_{\leqslant 2}(x,y) = 0. \tag{1.1}$$

В более подробном виде уравнение (1.1) можно записать как

$$Ax^3 + Bx^2y + Cxy^2 + Dy^3 + a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 = 0.$$

Таким образом, $f_3(x,y) = Ax^3 + Bx^2y + Cxy^2 + Dy^3$ — старшая (кубическая) однородная часть полинома $f(x,y)$.

1.1 Элементарная версия метода Рунге: общий алгоритм

Для кубических диофантовых уравнений (1.1) стандартное условие Рунге (см. [9], глава 28) состоит в том, что старшая однородная часть $f_3(x,y)$ полинома $f(x,y)$ разложима над $\mathbb{Z}$ в произведение *взаимно простых* полиномов:

$$f_3(x,y) = (\alpha x + \beta y)(Ax^2 + Bxy + Cy^2),$$

при этом сам $f(x,y)$ неразложим над $\mathbb{Z}$.

С помощью линейной замены неизвестных с целыми коэффициентами любое кубическое уравнение, удовлетворяющее условию Рунге, можно привести к виду

$$x(Ax^2 + Bxy + Cy^2) + a_1x^2 + a_2xy + 0 \cdot y^2 + a_4x + a_5y + a_6 = 0. \tag{1.2}$$

Говоря далее об алгоритме решения уравнения (1.2), будем предполагать $a_5 \neq 0$ (в противном случае алгоритм сводится к перебору всех делителей коэффициента a_6). Кроме того, можно считать $C > 0$.

Введем обозначение $\Delta = B^2 - 4AC$.

1. Случай $\Delta > 0$

Сначала рассмотрим наиболее сложный случай, когда $\Delta > 0$. Основная идея состоит в том, чтобы ввести дополнительный параметр $k \in \mathbb{Z}$ и затем исследовать систему уравнений:

$$\begin{cases} k = Ax^2 + Bxy + Cy^2 + a_1x + a_2y + a_4, \\ kx + a_5y + a_6 = 0. \end{cases} \tag{1.3}$$

Исключив из системы уравнений (1.3) неизвестное y, получим уравнение

$$\alpha x^2 + \beta x + \gamma = 0, \tag{1.4}$$

коэффициенты которого зависят от k следующим образом:

$$\begin{aligned} \alpha &= Ck^2 - Ba_5k + Aa_5^2, \\ \beta &= (2Ca_6 - a_2a_5)k + a_1a_5^2 - Ba_5a_6, \\ \gamma &= -a_5^2k + Ca_6^2 + a_4a_5^2 - a_2a_5a_6. \end{aligned}$$

В дальнейшем важную роль будет играть выражение

$$Q(m) = \frac{|Q_1|m + |Q_2|}{m^2 - \Delta a_5^2} + |a_5|\sqrt{\frac{|Q_3|m + |Q_4|}{m^2 - \Delta a_5^2} + \frac{|Q_5|m + |Q_6|}{(m^2 - \Delta a_5^2)^2}}, \tag{1.5}$$

где $Q_1, \ldots, Q_6$ определены равенствами

$$\begin{aligned} Q_1 &= a_2a_5 - 2Ca_6, \\ Q_2 &= a_5^2(Ba_2 - 2Ca_1), \\ Q_3 &= 2, \\ Q_4 &= 2Ba_5 - 4Ca_4 + a_2^2, \\ Q_5 &= 2(a_2a_5 - 2Ca_6)(Ba_2 - 2Ca_1), \\ Q_6 &= \Delta a_2^2a_5^2 + a_5^2(Ba_2 - 2Ca_1)^2 - 4\Delta Ca_2a_5a_6 + 4\Delta C^2a_6^2, \end{aligned} \tag{1.6}$$

а параметр m удовлетворяет условию

$$m > m_0 = |a_5|\sqrt{\Delta}. \tag{1.7}$$

Предлагаемый ниже алгоритм решения уравнения (1.2) в целых числах основан на следующей теореме, доказанной в [10].

Теорема 1.1. Пусть параметр m удовлетворяет условию (1.7). Если

$$\left|k - \frac{Ba_5}{2C}\right| > \frac{m}{2C} \quad \text{или} \quad \left|k - \frac{Ba_5}{2C}\right| < \frac{2m_0 - m}{2C},$$

то для корней уравнения (1.4) имеет место оценка $|x| < Q(m)$, где выражение $Q(m)$ определено равенством (1.5).

Доказательство. Положим

$$l = 2Ck - Ba_5.$$

Тогда $|l| > m$ или $|l| < 2m_0 - m$, а корни уравнения (1.4) можно записать как

$$x = \frac{Q_1 l + Q_2}{l^2 - \Delta a_5^2} \pm a_5 \sqrt{\frac{Q_3 l + Q_4}{l^2 - \Delta a_5^2} + \frac{Q_5 l + Q_6}{(l^2 - \Delta a_5^2)^2}}. \tag{1.8}$$

Следовательно,

$$|x| \leqslant \frac{|Q_1||l| + |Q_2|}{|l^2 - \Delta a_5^2|} + |a_5| \sqrt{\frac{|Q_3||l| + |Q_4|}{|l^2 - \Delta a_5^2|} + \frac{|Q_5||l| + |Q_6|}{(l^2 - \Delta a_5^2)^2}} < Q(m).$$

Последнее неравенство верно, так как на каждом из интервалов $t > m$ и $0 \leqslant t < 2m_0 - m$ обе величины

$$f_1(t) = \frac{t}{|t^2 - \Delta a_5^2|}, \quad f_2(t) = \frac{t}{(t^2 - \Delta a_5^2)^2}$$

являются монотонными функциями от t, при этом $f_i(m) > f_i(2m_0 - m)$ для $i = 1, 2$. □

Алгоритм решения уравнения (1.2)

1. Выберем параметр m так, чтобы выполнялось условие (1.7).

2. Для каждого $k \in \mathbb{Z}$, удовлетворяющего условию

$$\frac{2m_0 - m}{2C} \leqslant \left|k - \frac{Ba_5}{2C}\right| \leqslant \frac{m}{2C},$$

решим уравнение (1.4) относительно $x \in \mathbb{Z}$. Для каждой найденной такой пары $(x, k) \in \mathbb{Z}^2$ проверим, будет ли число

$$y = -\frac{kx + a_6}{a_5}$$

целым, и если да, то добавим пару $(x, y) \in \mathbb{Z}^2$ в множество решений.

3. Для каждого $x \in \mathbb{Z}$, удовлетворяющего условию $|x| < Q(m)$, решим уравнение (1.2) относительно $y \in \mathbb{Z}$. Найденные пары $(x, y) \in \mathbb{Z}^2$ добавим в множество решений.

Параметр m в предложенном алгоритме можно варьировать. Суммарное число решаемых квадратных уравнений определяется *функцией затрат*

$$\operatorname{cost}(m) = 2P(m) + 2Q(m),$$

где

$$P(m) = \begin{cases} \dfrac{2(m - m_0)}{2C}, & \text{если } m \leqslant 2m_0, \\ \dfrac{m}{2C}, & \text{если } m > 2m_0. \end{cases}$$

Для уменьшения объема вычислений в качестве параметра m можно взять точку минимума m^* функции $\operatorname{cost}(m)$ на интервале (m_0, ∞). Далее алгоритм, в котором $m = m^*$, будем называть *оптимизированным*.

К сожалению, для точки минимума m^* нет простого аналитического выражения, поэтому мы вынуждены ограничиться оценками величины $\operatorname{cost}(m^*)$. Положим

$$M = \max\{|A|, |B|, |C|\}, \quad H = \max\{|a_1|, |a_2|, |a_4|, |a_5|, |a_6|\}.$$

Теорема 1.2. Для оптимизированного алгоритма имеет место оценка

$$\operatorname{cost}(m^*) < C_1 M H, \tag{1.9}$$

где $C_1 > 0$ — абсолютная константа (можно взять, например, $C_1 = 20$).

Доказательство теоремы 1.2 см. в работе [10]. Отметим, что оценка (1.9) адекватно отражает только наихудший случай. Это означает, что в некоторых частных случаях данная оценка может быть существенно улучшена.

2. Остальные случаи

В случае $\Delta < 0$ можно предложить алгоритм, непосредственно основанный на оценке решений уравнения (1.2), предоставляемой следующей теоремой.

Теорема 1.3. При $\Delta < 0$ для решений (x, y) уравнения (1.2) имеет место оценка

$$|x| < C_2 M H,$$

где $C_2 > 0$ — абсолютная константа.

Доказательство теоремы 1.3 см. в работе [10].

В случае $\Delta = 0$ можно заметить, что здесь работает тот же алгоритм, что и в случае $\Delta < 0$, поскольку теорема 1.1 оказывается верной и при $\Delta = 0$.

1.2 Модификация общего алгоритма в частных случаях

Как уже было отмечено выше, выбор оптимального значения «управляющего» параметра m сопряжен с определенными трудностями. Однако в некоторых частных случаях эту проблему можно решить, при этом для минимального значения функции затрат $\mathrm{cost}\,(m)$ можно дать существенно лучшие оценки по сравнению с общим случаем (см. теорему 1.2).

Рассмотрим, к примеру, следующие семейства кубических диофантовых уравнений:

$$x(y^2 - 2x^2) + Hx + y + 1 = 0, \tag{1.10}$$

$$x(y^2 - 2x^2) + x + y + H = 0, \tag{1.11}$$

$$x(y^2 - 2x^2) + x + Hy + 1 = 0, \tag{1.12}$$

где параметр H принимает произвольные целые значения. Для каждого из этих уравнений ниже представлены алгоритмы их решения.

1. Уравнение (1.10)

Как и в общем случае, вводим дополнительный параметр $k \in \mathbb{Z}$ и получаем систему уравнений

$$\begin{cases} k = y^2 - 2x^2 + H, \\ kx + y + 1 = 0. \end{cases} \tag{1.13}$$

Исключив неизвестное y, получим уравнение

$$(k^2 - 2)x^2 + 2kx - k + H + 1 = 0. \tag{1.14}$$

Обозначим

$$Q(m) = \frac{m}{m^2 - 2} + \sqrt{\frac{m + |H|}{m^2 - 2} + \frac{2}{(m^2 - 2)^2}}. \tag{1.15}$$

Здесь параметр m удовлетворяет условию

$$m > \sqrt{2}. \tag{1.16}$$

Теорема, на которой основан алгоритм решения уравнения (1.10), формулируется следующим образом.

Теорема 1.4. Пусть параметр m удовлетворяет условию (1.16). Если

$$|k| > m,$$

то для всех решений уравнения (1.14) справедлива оценка $|x| < Q(m)$, где $Q(m)$ определено в (1.15).

Доказательство. Если воспользоваться стандартным неравенством «модуль суммы не превосходит суммы модулей», то из формулы для корней уравнения (1.14) можно вывести неравенство

$$|x| \leqslant \frac{|k|}{k^2 - 2} + \sqrt{\frac{|k| + |H|}{k^2 - 2} + \frac{2}{(k^2 - 2)^2}}.$$

Заметим, что функция $f(t) = t/(t^2-2)$ при $t > \sqrt{2}$ является убывающей (ее производная на этом промежутке отрицательна), поэтому при условии $|k| > m$ имеет место неравенство

$$\frac{|k|}{k^2-2} + \sqrt{\frac{|k|+|H|}{k^2-2} + \frac{2}{(k^2-2)^2}} < Q(m),$$

что и доказывает теорему. □

Исходя из теоремы 1.4, можно предложить следующий

Алгоритм решения уравнения (1.10)

1. Выберем параметр m так, чтобы выполнялось условие (1.16).

2. Для каждого целого числа k, удовлетворяющего условию $k \leqslant m$, решим уравнение (1.14) относительно неизвестного x в целых числах и добавим найденные таким образом пары $(x, y) = (x, -kx - 1)$ во множество решений.

3. Для каждого целого числа $x \neq 0$, удовлетворяющего условию $|x| < Q(m)$, решим уравнение (1.10) относительно неизвестного y в целых числах и добавим найденные таким образом пары (x, y) во множество решений. Кроме того, добавим во множество решений тривиальное решение $(0, 1)$.

2. Уравнение (1.11)

Данное уравнение решается аналогичным способом, как и уравнение (1.10). Вводим дополнительный параметр $k \in \mathbb{Z}$ и составляем систему уравнений:

$$\begin{cases} k = y^2 - 2x^2 + 1, \\ kx + y + H = 0. \end{cases} \tag{1.17}$$

Исключая неизвестное y, получаем уравнение

$$(k^2-2)x^2 + 2kxH - k + H^2 + 1 = 0. \tag{1.18}$$

Обозначим

$$Q(m) = \frac{Hm}{m^2-2} + \sqrt{\frac{m+1}{m^2-2} + \frac{2H^2}{(m^2-2)^2}}. \tag{1.19}$$

Параметр m по-прежнему удовлетворяет условию

$$m > \sqrt{2}. \tag{1.20}$$

Теорема, на которой основан алгоритм решения уравнения (1.11), формулируется следующим образом.

Теорема 1.5. Пусть параметр m удовлетворяет условию (1.20). Если

$$|k| > m,$$

то для всех решений уравнения (1.18) справедлива оценка $|x| < Q(m)$, где $Q(m)$ определено в (1.19).

Доказательство теоремы 1.5 практически идентично доказательству теоремы 1.4.

Алгоритм решения уравнения (1.11) имеет такую же структуру, как и алгоритм решения уравнения (1.10), поэтому мы опустим его описание.

3. Уравнение (1.12)

Как и выше, вводим дополнительный параметр $k \in \mathbb{Z}$ и составляем соответствующую систему уравнений:

$$\begin{cases} k = y^2 - 2x^2 + 1, \\ kx + Hy + 1 = 0. \end{cases} \tag{1.21}$$

Исключая неизвестное y, получаем уравнение

$$(k^2 - 2H^2)x^2 + 2kx - H^2(k-1) + 1 = 0. \tag{1.22}$$

Обозначим

$$Q(m) = \frac{m}{m^2 - m_0^2} + H\sqrt{\frac{m+1}{m^2 - m_0^2} + \frac{2}{(m^2 - m_0^2)^2}}. \tag{1.23}$$

Здесь параметр m удовлетворяет условию

$$m > m_0, \tag{1.24}$$

где $m_0 = H\sqrt{2}$.

Теорема, на которой основан алгоритм решения уравнения (1.12), формулируется следующим образом.

Теорема 1.6. Пусть параметр m удовлетворяет условию (1.24). Если

$$|k| > m \quad \text{или} \quad |k| < 2m_0 - m,$$

то для всех решений уравнения (1.12) справедлива оценка $|x| < Q(m)$, где $Q(m)$ определено в (1.23).

Доказательство. Из (1.22) следует, что

$$|x| < \frac{|k|}{|k^2 - m_0^2|} + H\sqrt{\frac{|k|+1}{|k^2 - m_0^2|} + \frac{2}{(k^2 - m_0^2)^2}}.$$

Функция $f(t) = t/|t^2 - m_0^2|$ при $t > m$ является убывающей, а при $0 < t < 2m_0 - m$ — возрастающей, при этом $f(2m_0 - m) < f(m)$. Поэтому если $|k| > m$ или $|k| < 2m_0 - m$, то

$$\frac{|k|}{|k^2 - m_0^2|} + H\sqrt{\frac{|k|+1}{|k^2 - m_0^2|} + \frac{2}{(k^2 - m_0^2)^2}} < Q(m),$$

что и доказывает теорему. □

Алгоритм решения уравнения (1.12)

1. Выберем параметр m так, чтобы выполнялось условие (1.24).

2. Для каждого целого числа k, удовлетворяющего условию $2m_0 - m \leqslant |k| \leqslant m$, решим уравнение (1.22) относительно неизвестного x в целых числах и добавим найденные таким образом пары $(x, y) = (x, -\frac{kx+1}{H})$ во множество решений.

3. Для каждого целого числа $x \neq 0$, удовлетворяющего условию $|x| < Q(m)$, решим уравнение (1.12) относительно неизвестного y в целых числах и добавим найденные таким образом пары (x, y) во множество решений.

Теперь осталось оптимизировать выбор «управляющего» параметра m.

4. Выбор оптимального значения параметра m

В каждом из рассмотренных уравнений присутствовал параметр m, который мы выбирали произвольным образом, исходя из соответствующих ограничений. Естественно выбрать значение m так, чтобы количество решаемых квадратных уравнений было как можно меньше.

В алгоритме решения уравнения (1.10) количество решаемых обычных квадратных уравнений можно определить функцией

$$\operatorname{cost}(m) = 2m + 2Q(m), \tag{1.25}$$

которую требуется минимизировать. Здесь $Q(m)$ определено равенством (1.15). Упростим формулу (1.25), заметив, что

$$Q(m) \approx \frac{\sqrt{|H|}}{m} = Q_0(m).$$

Теперь для функции затрат имеем

$$\operatorname{cost}(m) \approx 2m + 2Q_0(m). \tag{1.26}$$

Последнее выражение уже нетрудно минимизировать. Таким образом, для уравнения (1.10) оптимальное значение «управляющего» параметра $m^* \approx |H|^{1/4}$.

Для уравнения (1.11) рассуждения аналогичны. Количество решаемых обычных квадратных уравнений равно

$$\operatorname{cost}(m) = 2m + 2Q(m),$$

где $Q(m)$ определено равенством (1.19). Имеем

$$Q(m) \approx \frac{H}{m} + \frac{H\sqrt{2}}{m^2} \approx \frac{H}{m} = Q_0(m).$$

Следовательно,

$$\operatorname{cost}(m) \approx 2m + 2Q_0(m),$$

а последнее выражение будет иметь минимальное значение при $m = m^* \approx H^{1/2}$ — это и есть оптимальное значение «управляющего» параметра m.

Наконец, рассмотрим последнее уравнение (1.12). Здесь количество решаемых обычных квадратных уравнений равно

$$\operatorname{cost}(m) = 4m - 4m_0 + 2Q(m).$$

Положим $m = m_0 + aH^{2/3}$, где коэффициент $a > 0$ выберем позже. Тогда

$$\operatorname{cost}(m) = AH^{2/3} + o(H^{2/3}), \quad H \to \infty,$$

при этом для коэффициента A имеет место равенство

$$A = 4a + \sqrt{\frac{2}{a}}.$$

Нетрудно видеть, что коэффициент A будет минимальным при $a = 2^{-5/3}$. Таким образом, при данном значении коэффициента a количество решаемых квадратных уравнений будет минимальным и примерно равным $3 \cdot 2^{1/3} H^{2/3}$.

1.3 Дополнительные теоретические результаты

При анализе уравнений (1.10), (1.11) и (1.12) попутно были получены некоторые дополнительные теоретические результаты.

Аналитическое решение уравнения (1.10) **при** $H > 0$

Выяснилось, что данное уравнение возможно решить аналитически, не прибегая к алгоритму перебора. Справедлива следующая теорема.

Теорема 1.7. При $H \geqslant 4$ для каждого решения $(x, y) \neq (0, -1)$ уравнения (1.10) выполняется одно из данных равенств

$$x = -1 \pm \sqrt{H+3}, \quad x = \pm\frac{\sqrt{2H+2}}{2}, \quad x = 1 \pm \sqrt{H+1}. \tag{1.27}$$

Доказательство. Воспользуемся формулой

$$(k^2 - 2)x^2 + 2kx - k + H + 1 = 0, \tag{1.28}$$

выражающей неизвестное x через вспомогательный параметр k. Если $k \leqslant -2$, то подкоренное выражение в (1.27) будет отрицательно:

$$\frac{k-H}{k^2-2} + \frac{2}{(k^2-2)^2} \leqslant \frac{k-4}{k^2-2} + \frac{2}{(k^2-2)^2} < 0.$$

Если $k = 2$, то $x = -1 \pm \frac{\sqrt{6-2H}}{2}$, что при $H \geqslant 4$ невозможно. Аналогично, при $k = 3$ получим $x = -\frac{3}{7} \pm \frac{\sqrt{23-7H}}{2}$, что при $H \geqslant 4$ также невозможно. Пусть $k \geqslant 4$. Тогда

$$|x| \leqslant \frac{k-H}{k^2-2} + \sqrt{\frac{k-4}{k^2-2} + \frac{2}{(k^2-2)^2}} \leqslant \frac{7}{2} + \frac{1}{\sqrt{98}} < 1,$$

что опять невозможно, поскольку $x \neq 0$. Таким образом, остаются только три возможных значения $k = -1$, $k = 0$ и $k = 1$. Эти три значения k и приводят к равенствам (1.27). □

Как следствие, уравнение (1.10) при $H \geqslant 4$ также имеет не более, чем пять решений.

Оценка решений уравнения (1.10) **при** $H < 0$

Теорема 1.8. При $H < 0$ для решений (x, y) уравнения (1.10) справедлива оценка

$$x \leqslant 1 + \sqrt{\frac{|H|+3}{2}} \tag{1.29}$$

Доказательство. Как и при доказательстве теоремы 1.7, можно аккуратно проанализировать формулу (1.28) в зависимости от значений параметра k. Оценка (1.28) возникает при рассмотрении значения $k = 2$, при этом

$$x = -1 \pm \sqrt{\frac{3-H}{2}}.$$

При остальных значениях k неизвестное x по абсолютной величине оказывается меньше, чем правая часть в (1.29). □

Оценка решений уравнения (1.11) **при** $H < 0$

Теорема 1.9. Для решений (x, y) уравнения (1.11) справедлива оценка

$$|x| \leqslant H + \sqrt{2H^2 + 2}. \tag{1.30}$$

Доказательство. Проанализируем выражение для корня x уравнения (1.22) в зависимости от возможных значений параметра k. Нетрудно видеть, что если $k \geqslant 2$, то

$$|x| \leqslant H + \sqrt{\frac{H^2 + 1}{2}},$$

а если $k \leqslant -2$, то имеет строгое неравенство

$$|x| < H + \sqrt{\frac{H^2}{2}}.$$

Для оставшихся значений $k = 0$, $k = 1$ и $k = -1$ имеем соответственно

$$x = \pm\sqrt{\frac{H^2 + 1}{2}}, \quad x = H \pm \sqrt{2H^2}, \quad x = -H \pm \sqrt{2H^2 + 2}.$$

В итоге приходим к оценке (1.30). □

Заметим, что для решений (x, y) аналогичного уравнения

$$x(y^2 - 2x^2) + y + H = 0$$

известна оценка, аналогичная оценке (1.30) (см. статью [1], пример 2), однако способ ее доказательства не такой простой, как в теореме 1.9.

2 Компьютерная реализация на языке программирования Python

Среда разработки алгоритма была выбрана исходя из имеющихся знаний в области программирования и оптимизации скорости работы кода. Поэтому был выбран язык программирования Python, где впоследствии и проводились все исследования и сбор статистических данных.

2.1 Общий случай

Алгоритм решения уравнения (1.2) был запрограммирован. Код программы находится в приложении А.

Были реализованы следующие вспомогательные функции:

- *isqrt* — вычисление целочисленного квадратного корня
- *solve_int* — решение квадратного уравнения
- *runge* — алгоритм элементарной версии Рунге для решения кубических диофантовых уравнений

Также были импортированы некоторые функции из готовых библиотек:

- *math.floor* — округление числа в меньшую сторону
- *math.sqrt* — квадратный корень числа
- *time.clock* — время работы программы

2.2 Частные случаи: особенности реализации и статистика

Так как частные случаи были добавлены в работу для более детального изучения зависимости количества решений от размера коэффициента H,

появилась надобность сохранять полученные результаты.

Для хранения статистических данных было решено использовать нереляционную базу данных MongoDB, она отличается своей скоростью работы, не требовательная к различным типам данных и очень удобна в использовании в связке с Python, что значительно ускоряло работу.

Для подключения и «общения» с базой данных на Python, была использована библиотека *pymongo*, которая предоставляет API для управления MongoDB.

В процессе исследования, сохранялись следующие данные:

- значение коэффициента H,
- количество решений уравнения,
- массив данных с решениями уравнения.

Для того, чтобы собрать достаточное количество данных, пришлось прибегнуть к реализации *многопроцессности*, что позволило ускорить анализ уравнений.

Отталкиваясь от мощностей имеющейся вычислительной машины, было выбрано оптимальное количество одновременно запущенных процессов. Поэтому программы (см. приложения Б - Г) считали восемь уравнений с разными коэффициентами H одновременно, используя 100% мощности процессора.

Так как назначения программ для частного случая отличаются, были реализованы новые функции:

- *main* — функция отвечающая за распределение задач по процессам,
- *start_process* — функция отвечает за создание процессов, их запуск и остановку.

Новые импортированные функции из готовых библиотек:

- *multiprocessing.Process* — позволяет создать отдельный процесс

- *multiprocessing.cpu_count* — показывает количество поддерживаемых одновременных потоков

Статистика количества решений

Статистика количества решений уравнений приведена в таблицах ниже, где N — количество решений уравнения, а $\#H$ — количество уравнений с N решениями.

Таблица 1: Статистика для уравнения (1.10) в интервале $-10^7 \leqslant H \leqslant -1$.

N	1	2	3	4	5	6	7
$\#H$	9917061	71481	10999	356	99	3	1

Таблица 2: Статистика для уравнения (1.11)в интервале $1 \leqslant H \leqslant 10^6$.

N	1	2	3	4	5	6	7	8	9	10	11	12	13
$\#H$	952147	43431	3589	692	102	24	10	1	0	2	1	0	1

Таблица 3: Статистика для уравнения (1.12) в интервале $1 \leqslant H \leqslant 10^5$.

N	2	3	4	5	6	7	8	9
$\#H$	96204	3281	460	42	10	1	0	1

Заключение

В работе была рассмотрена алгоритмическая реализация элементарной версии метода Рунге для решения кубических диофантовых уравнений с двумя неизвестными в целых числах (см. раздел 1). Соответствующий алгоритм был запрограммирован на языке Python. К сожалению, оптимизация этого алгоритма была осуществлена только в некоторых частных случаях (см. раздел 1.2). Таким образом, цели работы достигнуты лишь частично. Тем не менее, удалось собрать интересную с теоретической точки зрения статистику количества решений некоторых частных семейств диофантовых уравнений. Отметим также, что попутно были получены дополнительные теоретические результаты о величине и количестве решений некоторых диофантовых уравнений, что представляет несомненный теоретический интерес.

Список использованных источников

[1] *Осипов Н.Н.* Элементарная версия метода Рунге для кубических уравнений // Математика в школе. 2012. № 1. С. 64 — 69.

[2] *Осипов Н.Н.* Метод Рунге для уравнений 4-й степени: элементарный подход // Математическое просвещение. Сер. 3. Вып. 19. М.: МЦНМО, 2015. С. 178 — 198.

[3] *Спринджук В.Г.* Классические диофантовы уравнения от двух неизвестных. М.: Наука, 1982.

[4] *Barbeau E.J.* Pell's equation. New York: Springer-Verlag, 2003.

[5] *Gorelick M., Ozsvald I.* High Performance Python // O'Reilly Media. 2014.

[6] *Hilliker D.L.* An algorithm for solving a certain class of diophantine equations. I // Math. Comput. 1982. V. 38. № 158. P. 611 — 626.

[7] *Hilliker D.L., Straus E.G.* Determination of bounds for the solutions to those binary diophantine equations that satisfy the hypotheses of Runge's theorem // Trans. Amer. Math. Soc. 1983. V. 280. P. 637 — 657.

[8] *Jones B., Beazley D.* Python Cookbook, 3rd Edition // O'Reilly Media. 2013.

[9] *Mordell L.J.* Diophantine equations. London: Academic Press Inc., 1969.

[10] *Osipov N.N., Gulnova B.V.* An Algorithmic Implementation of Runge's Method for Cubic Diophantine Equations // Journal of Siberian Federal University. 2018. V. 11(2). P. 137 — 147.

[11] *Poulakis D.* Polynomial bounds for the solutions of a class of diophantine equations // J. Number Theory. 1997. V. 66. P. 271 — 281.

[12] *Runge C.* Über ganzzahlige Lösungen von Gleichungen zwischen zwei Veränderlichen // J. reine und angew. Math. 1887. V. 100. P. 425—435.

[13] *Walsh P.G.* A quantitative version of Runge's theorem on diophantine equations // Acta Arithmetica. 1992. V. LXII. P. 157—172.

[14] `https://www.maplesoft.com/documentation_center/maple2017/UserManual.pdf`

[15] `https://www.wolfram.com/products/webmathematica/resources/userguide.pdf`

Приложение А. Программа для решения уравнения (1.2)

```
#Подключенные библиотеки

#округление числа в меньшую сторону и квадратный корень
from math import floor, sqrt
#счетчик времени
from time import clock

#Полный квадрат
def isqrt(x):
    n = int(x)
    if n == 0:
        return 0
    a, b = divmod(n.bit_length(), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y

#Решение квадратного уравнения
def solve_int(a,b,c):
    d = b*b-4*a*c
    s = set()
    if (d>=0):
```

```
        d2 = isqrt(d)
        if ((d2*d2)==d):
            if(divmod(-b+d2,2*a)[1]==0):
                tmp1 = (-b+d2)//(2*a)
                s.add(tmp1)
            if(divmod(-b-d2,2*a)[1]==0):
                tmp2 = (-b-d2)//(2*a)
                s.add(tmp2)
    return s

#Решение уравнения
def runge(A,B,C,a):
    S=set()
    Delta=B**2-4*A*C
    m0=abs(a[4])*sqrt(Delta)
    m=floor(m0+abs(a[4])**(2/3))
    Q1=a[1]*a[4]-2*C*a[5]
    Q2=a[4]**2*(B*a[1]-2*C*a[0])
    Q3=2
    Q4=2*B*a[4]-4*C*a[3]+a[1]**2
    Q5=2*(a[1]*a[4]-2*C*a[5])*(B*a[1]-2*C*a[0])
    Q6=Delta*a[1]**2*a[4]**2+a[4]**2*(B*a[1]-2*C*a[0])**2-\
            4*Delta*C*a[1]*a[4]*a[5]+4*Delta*C**2*a[5]**2

    QQ=floor((abs(Q1)*m+abs(Q2))/(m**2-Delta*a[4]**2)+\
            abs(a[4])*sqrt((abs(Q3)*m+abs(Q4))/\
            (m**2-Delta*a[4]**2)+\
            (abs(Q5)*m+abs(Q6))/(m**2-Delta*a[4]**2)**2))
```

```
if ((m**2)>=(4*(m0**2))):
    for k in range(divmod(B*a[4]-m,2*C)[0],\
                   divmod(B*a[4]+m,2*C)[0]+1):
        res=solve_int(C*k**2-B*a[4]*k+A*a[4]**2,\
        (2*C*a[5]-a[1]*a[4])*k+a[0]*a[4]**2-B*a[4]*a[5],\
        -a[4]**2*k+C*a[5]**2+a[3]*a[4]**2-a[1]*a[4]*a[5])

        for x in res:
            if (divmod(k*x+a[5],a[4])[1]==0):
                S.add((int(x),int(-(k*x+a[5])/a[4])))

if (m**2<4*m0**2):
    for k in range(floor((B*a[4]-m)/2/C),\
                   floor((B*a[4]+m-2*m0)/2/C)+1):
        res=solve_int(C*k**2-B*a[4]*k+A*a[4]**2,\
        (2*C*a[5]-a[1]*a[4])*k+a[0]*a[4]**2-B*a[4]*a[5],\
        -a[4]**2*k+C*a[5]**2+a[3]*a[4]**2-a[1]*a[4]*a[5])

        for x in res:
            if divmod(k*x+a[5],a[4])[1]==0:
                S.add((int(x),int(-(k*x+a[5])/a[4])))

    for k in range(floor((B*a[4]-m+2*m0)/2/C),\
                   floor((B*a[4]+m)/2/C)+1):
        res=solve_int(C*k**2-B*a[4]*k+A*a[4]**2,\
        (2*C*a[5]-a[1]*a[4])*k+a[0]*a[4]**2-B*a[4]*a[5],\
        -a[4]**2*k+C*a[5]**2+a[3]*a[4]**2-a[1]*a[4]*a[5])
```

```
                for x in res:
                    if divmod(k*x+a[5],a[4])[1]==0 :
                        S.add((int(x),int(-(k*x+a[5])/a[4])))

        if divmod(a[5],a[4])[1]==0:
            S.add((0,int(-a[5]/a[4])))

        for x in range(1, QQ+1):
            res=solve_int(C*x,B*x**2+x*a[1]+a[4],\
            A*x**3+a[0]*x**2+x*a[3]+a[5])
            for y in res:
                S.add((int(x),int(y)))

        for x in range(-QQ,0):
            res=solve_int(C*x,B*x**2+x*a[1]+a[4],\
            A*x**3+a[0]*x**2+x*a[3]+a[5])
            for y in res:
                S.add((int(x),int(y)))

        return S

#Начало программы
if __name__=="__main__":
            start = clock() #счетчик времени
            for H in range(1,10):
            #запуск расчетов
                    solve = runge(-1,1,1,[0,0,0,0,H,1])
```

```
            print(solve)
end = clock()
work_time = end-start
print("[+] Finished in %fs"%(work_time))
```

Приложение Б. Программа для решения уравнения (1.10)

```
#Подключенные библиотеки


#округление числа в меньшую сторону и квадратный корень
from math import floor, sqrt
#счетчик времени
from time import clock
#создание процесса
from multiprocessing import Process,cpu_count
#база данных MongoDB
from pymongo import MongoClient

#Полный квадрат
def isqrt(x):
    n = int(x)
    if n == 0:
        return 0
    a, b = divmod(n.bit_length(), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y

#Решение квадратного уравнения
```

```
def solve_int(a,b,c):
    d = b*b-4*a*c
    s = set()
    if (d>=0):
        d2 = isqrt(d)
        if ((d2*d2)==d):
            if(divmod(-b+d2,2*a)[1]==0):
                tmp1 = (-b+d2)//(2*a)
                s.add(tmp1)
            if(divmod(-b-d2,2*a)[1]==0):
                tmp2 = (-b-d2)//(2*a)
                s.add(tmp2)
    return s


#Решение уравнения
def runge(H):
    s = set()
    m0 = H*sqrt(2)
    delta_m = 0.3149802625*(H**(2/3))
    m = m0 + delta_m
    tmp1 = m**2-m0**2
    Q = floor(m/(tmp1) + \
            H*sqrt((m+1)/(tmp1) + (2/(tmp1)**2)))
    endof1 = floor(m0-delta_m)
    endof2 = floor(m0+delta_m)
    for k in range(endof1 -1 ,endof2 + 2):
        s = s.union(map(lambda x:(x,-(k*x+1)/H)\
         if divmod(k*x+1,H)[1]==0 else None,\
```

```
            solve_int(k**2-2*H**2,2*k,-H**2*(k-1)+1)))

    for k in range(-endof2 -1 ,-endof1 + 2):
        s = s.union(map(lambda x:(x,-(k*x+1)/H)\
         if divmod(k*x+1,H)[1]==0 else None,\
          solve_int(k**2-2*H**2,2*k,-H**2*(k-1)+1)))

    for x in range(1,Q+1):
        s = s.union(map(lambda y:(x,y),\
         solve_int(x,H,-2*x**3+x+1)))
    for x in range(-Q,0):
        s = s.union(map(lambda y:(x,y),\
         solve_int(x,H,-2*x**3+x+1)))
    return s

#Главная функция, для запуска решения уравнения
def main(start,step):
        client = MongoClient()
        db = client.runge
        collection = db.solved_equation
        bulk = collection.initialize_ordered_bulk_op()
        for H in range(start,count+1,step):
                solve = runge(H)
                length = len(solve)
                bulk.insert({"h":H,"length":length,\
                                "out":list(solve)})
        bulk.execute()
```

```
#Распределение функции main по процессам
def start_process(func):
    tasks = [Process(target = func,\
                     args=(i+1,total_process,))\
                     for i in range(total_process)]
    print("[+] Process create")
    for i in tasks:
        i.start()
    print("[+] Process start")
    for i in tasks:
        i.join()
    print("[+] Process stop")

#Начало программы
if __name__=="__main__":
        count = 1000 #количесво уравнений
        total_process = cpu_count() #количество процессов
    start = clock() #счетчик времени
    start_process(main) #запуск расчетов
    end = clock()
    work_time = end-start
    print("[+] Finished in %fs"%(work_time))
```

Приложение В. Программа для решения уравнения (1.11)

```
#Подключенные библиотеки

#округление числа в меньшую сторону и квадратный корень
from math import floor, sqrt
#счетчик времени
from time import clock
#создание процесса
from multiprocessing import Process,cpu_count
#база данных MongoDB
from pymongo import MongoClient

#Полный квадрат
def isqrt(x):
    n = int(x)
    if n == 0:
        return 0
    a, b = divmod(n.bit_length(), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y

#Решение квадратного уравнения
```

```
def solve_int(a,b,c):
    d = b*b-4*a*c
    s = set()
    if (d>=0):
        d2 = isqrt(d)
        if ((d2*d2)==d):
            if(divmod(-b+d2,2*a)[1]==0):
                tmp1 = (-b+d2)//(2*a)
                s.add(tmp1)
            if(divmod(-b-d2,2*a)[1]==0):
                tmp2 = (-b-d2)//(2*a)
                s.add(tmp2)
    return s

#Решение уравнения
def runge(H):
        s = set()
        s.add((0,-H))
        m = floor(sqrt(H)) + 2
        Q = floor((m*H)/(m**2-2) + \
            sqrt((m+1)/(m**2-2)+(2*H**2)/(m**2-2)**2))
        for k in range(-m,m+1):
            s = s.union(map(lambda x:(x,-k*x-H),\
             solve_int(k**2-2,2*H*k,-k+1+H**2)))
        for x in range(1,Q+1):
            s = s.union(map(lambda y:(x,y),\
            solve_int(x,1,-2*x**3+x+H)))
        for x in range(-Q,0):
```

```
                s = s.union(map(lambda y:(x,y),\
            solve_int(x,1,-2*x**3+x+H)))
        return s

#Главная функция, для запуска решения уравнения
def main(start,step):
        client = MongoClient()
        db = client.runge
        collection = db.solved_equation
        bulk = collection.initialize_ordered_bulk_op()
        for H in range(start,count+1,step):
                solve = runge(H)
                length = len(solve)
                bulk.insert({"h":H,"length":length,\
                        "out":list(solve)})
        bulk.execute()

#Распределение функции main по процессам
def start_process(func):
    tasks = [Process(target = func,\
                args=(i+1,total_process,))\
                 for i in range(total_process)]
    print("[+] Process create")
    for i in tasks:
        i.start()
    print("[+] Process start")
    for i in tasks:
        i.join()
```

```
        print("[+] Process stop")


#Начало программы
if __name__=="__main__":
    count = 10000 #количесво уравнений
    total_process = cpu_count() #количество процессов
        start = clock() #счетчик времени
        start_process(main) #запуск расчетов
        end = clock()
        work_time = end-start
        print("[+] Finished in %fs"%(work_time))
```

Приложение Г. Программа для решения уравнения (1.12)

```
#Подключенные библиотеки

#округление числа в меньшую сторону и квадратный корень
from math import floor, sqrt
#счетчик времени
from time import clock
#создание процесса
from multiprocessing import Process,cpu_count
#база данных MongoDB
from pymongo import MongoClient

#Полный квадрат
def isqrt(x):
    n = int(x)
    if n == 0:
        return 0
    a, b = divmod(n.bit_length(), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y

#Решение квадратного уравнения
```

```
def solve_int(a,b,c):
    d = b*b-4*a*c
    s = set()
    if (d>=0):
        d2 = isqrt(d)
        if ((d2*d2)==d):
            if(divmod(-b+d2,2*a)[1]==0):
                tmp1 = (-b+d2)//(2*a)
                s.add(tmp1)
            if(divmod(-b-d2,2*a)[1]==0):
                tmp2 = (-b-d2)//(2*a)
                s.add(tmp2)
    return s

#Решение уравнения
def runge(H):
    s = set()
    m0 = floor(H*sqrt(2))
    delta_m = floor(0.5*(H**(2/3)))
    m = m0 + delta_m + 1
    tmp1 = m**2-2*H**2
    Q = floor(m/(tmp1) +\
            H*sqrt((m+1)/(tmp1) + (2/(tmp1)**2)))
    endof1 = m0-delta_m
    endof2 = m0+delta_m
    for k in range(endof1 -1 ,endof2 + 2):
        s = s.union(map(lambda x:(x,-k*x-H),\
                solve_int(k**2-2*H**2,2*k,-H*(k-1)+1)))
```

```
    for k in range(-endof2 -1 ,-endof1 + 2):
        s = s.union(map(lambda x:(x,-k*x-H),\
                solve_int(k**2-2*H**2,2*k,-H*(k-1)+1)))
    for x in range(1,Q+1):
        s = s.union(map(lambda y:(x,y),\
                solve_int(x,H,-2*x**3+x+1)))
    for x in range(-Q,0):
        s = s.union(map(lambda y:(x,y),\
                solve_int(x,H,-2*x**3+x+1)))
    return s

#Главная функция, для запуска решения уравнения
def main(start,step):
        client = MongoClient()
        db = client.runge
        collection = db.solved_equation
        bulk = collection.initialize_ordered_bulk_op()
        for H in range(start,count+1,step):
                solve = runge(H)
                length = len(solve)
                bulk.insert({"h":H,"length":length,\
                        "out":list(solve)})
        bulk.execute()

#Распредение функции main по процессам
def start_process(func):
    tasks = [Process(target = func,\
                args=(i+1,total_process,))\
```

```
                    for i in range(total_process)]
    print("[+] Process create")
    for i in tasks:
        i.start()
    print("[+] Process start")
    for i in tasks:
        i.join()
    print("[+] Process stop")

#Начало программы
if __name__=="__main__":
    count = 1000 #количесво уравнений
    total_process = cpu_count() #количество процессов
    start = clock() #счетчик времени
    start_process(main) #запуск расчетов
    end = clock()
    work_time = end-start
    print("[+] Finished in %fs"%(work_time))
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий
Кафедра прикладной математики и компьютерной безопасности

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. А. Кытманов
подпись
«18» 06 2018 г.

БАКАЛАВРСКАЯ РАБОТА

01.03.04 Прикладная математика

Компьютерная реализация элементарной версии метода Рунге
для кубических диофантовых уравнений

Руководитель 16.06.18 профессор, д.ф.-м.н. Н. Н. Осипов
подпись, дата

Выпускник 12.06.18 А. А. Муравлев
подпись, дата

Красноярск 2018