MATHEMATICS OF COMPUTATION
VOLUME 61, NUMBER 203
JULY 1993, PAGES 235–244

ON SOLVING THE DIOPHANTINE EQUATION $x^3 + y^3 + z^3 = k$ ON A VECTOR COMPUTER

D. R. HEATH-BROWN, W. M. LIOEN, AND H. J. J. TE RIELE

Dedicated to the memory of D. H. Lehmer

ABSTRACT. Recently, the first author has proposed a new algorithm for solving the Diophantine equation $x^3 + y^3 + z^3 = k$, where k is a given nonzero integer. In this paper we present the detailed versions of this algorithm for some values of k given below, and we describe how we have optimized and run the algorithm on a Cyber 205 vector computer. A vectorized version of the Euclidean algorithm was written, which is capable of solving the equations $w_i x_i \equiv 1 \bmod n_i$, $i = 1, 2, \ldots$, at vector speed. Moreover, the basic double-precision arithmetic operations (+, −, ×, /) were vectorized. The following cases were implemented and run: $k = 3, 30, 2, 20, 39$, and 42. For $k = 3$ a range was searched which includes the cube $|x|, |y|, |z| \le 10^8$; this considerably extends an earlier search in the cube $|x|, |y|, |z| \le 2^{16}$. No solutions were found apart from the known ones $(1, 1, 1)$ and $(4, 4, -5)$. For $k = 30$, which probably is, with $k = 3$, the case which has attracted most attention in the literature, no solution was found. It is the smallest case for which no solution is known and for which one has not been able to find a proof that no solution exists. For $k = 2$ a parametric form solution is known, but we also found one which does not belong to this parametric form, viz., $(1214928, 3480205, -3528875)$. For $k = 20$, several new large solutions were found in addition to several known ones; this case served as a (partial) check of the correctness of our program. Finally, for $k = 39$ we found the first solution: $(134476, 117367, -159380)$. Hence, this case can be dropped from the list of values of k ($k = 30, 33, 39, 42, \ldots$) for which no solution is known (yet).

1. INTRODUCTION

Recently [3], the first author has presented a new algorithm to find solutions of the Diophantine equation

$$x^3 + y^3 + z^3 = k, \tag{1}$$

in which k is a fixed positive integer, and the x, y, z can be any integers, positive, negative, or zero. In order to find solutions with $|x|, |y|, |z| \le N$, this algorithm takes $\mathcal{O}_k(N \log N)$ steps, where the implied constant depends on k. In [3] this algorithm is given explicitly for the case $k = 3$, but significant changes have to be made for other values of k, depending mainly on the class number of $\mathbb{Q}(\sqrt[3]{k})$.

Received by the editor March 24, 1992 and, in revised form, December 14, 1992.
1991 *Mathematics Subject Classification*. Primary 11D25, 11Y50.
Key words and phrases. Cubic Diophantine equation, vector computer, Euclidean algorithm.

For $k = 3$, the idea of the new algorithm is as follows. If $k \equiv 3 \bmod 9$ then $x \equiv y \equiv z \equiv 1 \bmod 3$. If x, y, and z all have the same sign, then $x = y = z = 1$. Otherwise, let x and y have the same sign, and z the other; then we have $|x+y| \geq |z| \geq 1$. Now let $n := x+y$ and solve the equation $z^3 \equiv 3 \bmod n$ with z and n having different sign and $1 \leq |z| \leq |n|$. In [3] it is derived by factoring in $\mathbb{Q}(\sqrt[3]{3})$ (which has class number equal to 1) that $(n, 3) = 1$ and that

$$n = a^3 + 3b^3 + 9c^3 - 9abc$$

for some integers a, b, c such that

$$z \equiv (3c^2 - ab)(b^2 - ac)^{-1} \bmod n$$

(with z and n having different sign and $(b^2 - ac, n) = 1$). This gives a unique value of z. We can then solve the equations $x^3 + y^3 + z^3 = 3$ and $x + y = n$ to find x and y. This yields

$$x = \frac{n+d}{2}, \qquad y = \frac{n-d}{2}$$

$$\text{with } d = \sqrt{D} \text{ and } D = \frac{1}{3}\left[4\left(\frac{3-z^3}{n}\right) - n^2\right].$$

Here, D should be the square of an integer to yield integral x and y. If we choose $a = -1$, $b = 0$ and $c = 1$, we get $n = 8$, $z = -5$, $D = 0$ and $x = y = 4$ ($(1, 1, 1)$ and $(4, 4, -5)$ are the only known solutions for $k = 3$).

In [6] and [2] solutions of (1) were computed by means of a straightforward algorithm, which for given z and k checks whether any of the possible combinations of values of x and y in a chosen range satisfies (1). The range chosen in [2] (which includes the one chosen in [6]) was:

$$0 \leq x \leq y \leq 2^{16},$$
$$0 < N \leq 2^{16}, \qquad N = z - x,$$
$$0 < |k| \leq 999.$$

This algorithm takes $\mathcal{O}(N^2)$ steps, but it finds solutions of (1) for a *range* of values of k. The implied $\mathcal{O}$-constant depends on that range.

It is easily seen that equation (1) has no solution at all if $k \equiv \pm 4 \bmod 9$. There is no known reason for excluding any other values of k, although there are still many values of k for which no solution at all is known. Those below 100 (and $\not\equiv \pm 4 \bmod 9$) are:

(2) $\qquad k = 30, 33, 39, 42, 52, 74, 75, \text{ and } 84.$

For some values of k infinitely many solutions are known. For example, we have

$$(9t^4)^3 + (-9t^4 + 3t)^3 + (-9t^3 + 1)^3 = 1,$$

and

$$(6t^3 + 1)^3 + (-6t^3 + 1)^3 + (-6t^2)^3 = 2.$$

These relations give a solution of (1) for each $t \in \mathbb{Z}$. For $k = 1$ many solutions are known which do *not* satisfy the above parametric form (e.g., $(64, 94, -103)$). For more parametric solutions, see [6, 7, and 5].

It is possible to implement the new algorithm on an arbitrary vector computer. In particular, the Euclidean algorithm for the computation of $(b^2-ac)^{-1} \bmod n$ can be vectorized using standard Fortran. In this paper we present the results of optimizing and running this algorithm on a Cyber 205 vector computer, for $k = 3, 30, 2, 20, 39$, and 42. The cases $k = 3$ and $k = 30$ probably are the most intensively studied ones (cf. [2, 6, and 8]). For $k = 2$ the above parametric solution is known, but we wanted to check whether other solutions exist. For $k = 20$ the density of adèlic points is rather high, and relatively many integer points are known. This case was used as a (partial) check of the correctness of our program. The smallest value of $k > 30$ for which no solution is known is $k = 33$. However, the fundamental unit of $\mathbb{Q}(\sqrt[3]{33})$ is enormous, and in this case the algorithm becomes very inefficient. Therefore, we selected the next two cases $k = 39$ and $k = 42$.

In §2 we give a precise description of the algorithms for the various chosen values of k. Section 3 presents some details of how we have implemented the algorithms on the Cyber 205 vector computer and the results obtained. In particular, we describe how we have vectorized the computation of $(b^2-ac)^{-1} \bmod n$ (§3.1). The double-precision arithmetic operations which were necessary because of the size of the numbers we wanted to handle, were vectorized along the lines of [9]. The results of our computations are listed and discussed in §3.2. No (new) solutions were found for $k = 3$, 30, and 42. For $k = 2$ the first solution was found which is *not* of the parametric form given above. For $k = 20$ eight new solutions were found and, finally, for $k = 39$ we found one solution, so this case can be removed from the list of values of k for which no solution is known.

2. The algorithms for $k = 2, 3, 20, 30, 39$, and 42

For $k = 3$ the algorithm is derived and presented in [3]. The other cases can be derived in a similar way, but with significant changes caused by the facts that prime factors of k may occur in n $(= x + y)$, and that $\mathbb{Q}(\sqrt[3]{k})$ may not have unique factorization. We first present the algorithms for all the values of k listed above, except for $k = 20$: this case is given separately.

For $k \neq 20$, the algorithms are organized in such a way that all the solutions of (1) are found for which

$$1 \le |z| \le |x+y| \le |\epsilon| K^3, \tag{3}$$

where $|\epsilon|^{-1} > 1$ is the fundamental unit of $\mathbb{Q}(\sqrt[3]{k})$. This includes the cube $|x|, |y|, |z| \le \frac{1}{2}|\epsilon|K^3$.

The algorithm for $k = 2, 3, 30, 39, 42$. Let $\theta := \sqrt[3]{k}$; for the combination of values of r, a, b, and c, given in Table 1 (next page):

1. Let a, b, c run over the integers in the ranges

$$|a|,\ \theta|b|,\ \theta^2|c| \le Kr^{1/3},$$

for suitably chosen K (depending on the available computing resources).

2. Let $n := (a^3 + kb^3 + k^2c^3 - 3kabc)/r$, $w := b^2 - ac$ and $v := kc^2 - ab$ (r, a, b, and c are such that n is integral).

TABLE 1. Values of k, r, a, b, and c

k	r	restrictions on a, b, c
2	1	$a+2b+4c \equiv 1$ or $2 \bmod 6$ and either $2 \nmid a$ or $2 \mid a$, $4 \nmid a$, $b \equiv 1 \bmod 4$ or $4 \mid a$, $b+2c \equiv 1 \bmod 4$
3	1	$a \equiv 2 \bmod 3$
30	1	$a \equiv 2 \bmod 3$
	2	$a \equiv 4 \bmod 6$
	5	$a \equiv 10 \bmod 15$
39	1	$a \equiv 2 \bmod 3$
	2	$a \equiv 1 \bmod 3$, $a+b+c \equiv 0 \bmod 2$
	3	$a \equiv 0$, $b \equiv 2 \bmod 3$
	6	$a \equiv 0$, $b \equiv 1 \bmod 3$, $a+b+c \equiv 0 \bmod 2$
	9	$a \equiv 0$, $b \equiv 0$, $c \equiv 2 \bmod 3$
	18	$a \equiv 0$, $b \equiv 0$, $c \equiv 1 \bmod 3$, $a+b+c \equiv 0 \bmod 2$
42	1	$a \equiv 1 \bmod 3$
	3	$a \equiv 0$, $b \equiv 2 \bmod 3$
	9	$a \equiv 0$, $b \equiv 0$, $c \equiv 1 \bmod 3$

3a. ($k = 2, 3, 39, 42$) Use the Euclidean algorithm to find $\overline{w} := w^{-1} \bmod n$ (provided that w and n are coprime; if not, reject this quadruple (r, a, b, c)).

3b. ($k = 30$) Take $n' = n$ if $r \nmid b$, and $n' = n/r$ if $r \mid b$; use the Euclidean algorithm to find $\overline{w} := w^{-1} \bmod n'$ (provided that w and n' are coprime; if not, reject this quadruple (r, a, b, c)).

4. Compute $z \equiv v \cdot \overline{w} \bmod n$ with z in the range $1 \le |z| \le |n|$ and having opposite sign to n.

5. Compute $D = \frac{1}{3}[4(\frac{k-z^3}{n}) - n^2]$. If D is not the square of an integer, reject this quadruple (r, a, b, c).

6. Find the solution $(x, y, z) = (\frac{n+\sqrt{D}}{2}, \frac{n-\sqrt{D}}{2}, z)$.

The algorithm for $k = 20$.

1. Let a, b, c run over the integers in the ranges

$$|a|,\ \sqrt[3]{20}|b|,\ \sqrt[3]{50}|c| \le K,$$

for suitably chosen K (depending on the available computing resources).

2a. $2 \nmid a,\ 3 \nmid a - (b + c)$:

$$n' := a^3 + 20b^3 + 50c^3 - 30abc,\quad w := 2b^2 - ac,\quad v := 10c^2 - 2ab.$$

2b. $2 \nmid b,\ 3 \nmid c - (a + b)$:

$$n' := 2a^3 + 5b^3 + 100c^3 - 30abc,\quad w := b^2 - ac,\quad v := 20c^2 - 2ab.$$

2c. $2 \nmid c,\ 3 \nmid a + b + c$:

$$n' := 4a^3 + 10b^3 + 25c^3 - 30abc,\quad w := b^2 - ac,\quad v := 5c^2 - 2ab.$$

3. Use the Euclidean algorithm to find $\overline{w} := w^{-1} \bmod n'$ (provided that $(w, n') = 1$; if not, reject the triple (a, b, c)).

4a. Compute

$$z \equiv v \cdot \overline{w} \bmod n'; \tag{4}$$

4b1. $n := n',\ 1 \le |z| \le |n|,\ n,\ z$ of opposite sign.

4b2. $n := 4n',\ 1 \le |z| \le |n|,\ n,\ z$ of opposite sign; there will be four solutions of (4); we require further that $z + n \equiv 2 \bmod 6$.

5. and 6. Similar to steps 5 and 6 of the previous algorithm.

3. Implementation on the Cyber 205

We have implemented the algorithms of §2 in terms of long vectors, in order to reach optimal performance on the Cyber 205 vector computer.

A vector version of the Euclidean algorithm (needed in Step 3) was formulated which has input vectors **n** and **w** with components n_i and w_i, respectively, and which computes a vector **u** with components u_i such that $w_i u_i \equiv 1 \bmod n_i$. This is described in §3.1.

In Step 4 of the algorithm the product $v \cdot \overline{w}$ becomes too large for the (48 bits-) integer capacity of the Cyber 205. Therefore, we have written a vectorized double-precision version of the modular multiplication, which returns an integer result vector.

For the arithmetic operations involved in Steps 5 and 6 we also have written vectorized double-precision routines on the Cyber 205 (namely, for vector addition, subtraction, multiplication, and division, and conversion from integer to double precision and vice versa). These routines are based on ideas of Schlichting for double-precision BLAS (Basic Linear Algebra Subroutines) on the Cyber 205 [9], to which we refer for details. It should be noticed that Schlichting had to use the standard Fortran convention for storage of double-precision floating-point numbers, i.e., the upper and lower part of a double-precision number are stored in *consecutive* array locations. This has the disadvantage of yielding a stride two in vector operations on the double-precision numbers. In order to avoid these strides in our implementation, we have stored the upper and lower parts in two *separate* arrays.

In §3.2 we present the results of running our algorithms.

3.1. **Solving the equation** $wx \equiv 1 \bmod n$. For given w and n, the scalar equation $wx \equiv 1 \bmod n$, where $\gcd(w, n) = 1$ and $1 \le w < n$, can be solved as follows. Consider the regular continued fraction (abbreviated: c.f.) expansion of the rational number w/n. If c/d is the penultimate convergent of this c.f., then we have $wd - nc = \pm 1$, so that $wd \equiv \pm 1 \bmod n$. Here, the proper sign depends on the parity of the number of convergents of the c.f. of w/n. So we need to compute the *denominators* of the convergents of the c.f. of w/n. To that end we just follow the Euclidean algorithm for computing $\gcd(w, n)$, and we update the denominator of the convergent at each step (with the denominators from the previous two steps). The resulting algorithm looks as follows.

Scalar algorithm to compute $u = w^{-1} \bmod n$.

```
      sign = 1; $d0 = 0$; $d1 = 1$
      $a = w$; $b = n$
   10 $q = \lfloor b/a \rfloor$; $r = b - q \times a$
      if $r = 0$ then
c now $a$ contains gcd($w$, $n$)
          if $a = 1$ then
              $u$ = sign $\times d1$
          else
              $u = 0$
          endif
          return
      endif
      $h = q \times d1 + d0$; $d0 = d1$; $d1 = h$
      $b = a$; $a = r$; sign = − sign
      goto 10
```

For the *vectorization* of this algorithm, we store the pairs (w_i, n_i) ($i = 1, \dots, l$, where in our program we take $l = 10{,}000$) into the vectors **w** and **n**. All the scalar variables of the algorithm, except sign, are turned now into vectors of length l. Since not all u_i's will be completed in the same number of Euclidean algorithm steps, we introduce a *bit* vector **mask** (the components of which can only be '0' or '1') for 'compressing out' completed (w_i, n_i)-pairs. To keep track of the original locations, we maintain an index vector **ind**, initially [1: l], and compress this simultaneously with **w** and **n**. The vector algorithm will terminate as soon as the length of the compressed vectors becomes zero.

In the vectorized algorithm given below the statement **mask** = $(\mathbf{r} \neq \mathbf{0})$ means that **mask**(i)=1 if **r**(i)$\neq 0$ and **mask**(i)=0 if **r**(i)=0. The statement compress(**a**, **b**, **c**), where **b** is a bit vector, means that only those elements of the vector **a** are stored in (consecutive locations of) **c** for which the corresponding elements of **b** are 1. The statement scatter(**a**, **d**, **e**) means that the elements of **a** are stored in a location of **e**, the index of which is determined by the value of the corresponding element of **d**. (For simplicity, we assume here that all the gcd's of corresponding components of the input vectors **n** and **w** are 1. In our program we use a second bit vector to handle gcd's > 1.)

Vector algorithm to compute $\mathbf{u} = \mathbf{w}^{-1} \bmod \mathbf{n}$.

```
        ind = [1 : l]
        sign = 1 ; d0 = 0; d0 = 0
        a = w ; b = n
     10 q = ⌊b/a⌋ ; r = b − q × a
  c generate the bit vector mask
        mask = (r ≠ 0)
  c store those components of d1 for which the corresponding components
    of r are 0,
  c into the proper place in u
        compress(ind, ¬ mask, order)
        compress(d1, ¬ mask, t)
        if (sign = −1) t = −t
        scatter(t, order, u)
  c compress ind, q, r, a, d0, and d1 for next step
        compress(ind, mask, ind)
        compress(q, mask, q)
        compress(r, mask, r)
        compress(a, mask, a)
        compress(d0, mask, d0)
        compress(d1, mask, d1)
        if (length(ind) = 0) return
        h = q × d1 + d0 ; d0 = d1 ; d1 = h
        b = a; a = r; sign = − sign
        goto 10
```

It should be noted that most of the vector movements in the two lines before the "goto 10"-line can be accomplished by operating on *pointers to the vectors* rather than on the vectors themselves (on the Cyber 205, these pointers are called *descriptors*).

3.2. **Results.** We have run our program for solving (1) for the values of k given in §2, for various values of K. The results are listed in Table 2 (next page). This table also gives, for $k = 2, 3, 30, 39, 42$, the size of the largest (x, y, z)-cube which is contained in the range of searched (a, b, c)-values. This number equals the bound $\frac{1}{2}|\epsilon|K^3$ (truncated to three decimal places) which is given below (3) in §2. The fundamental units ϵ were taken from [1, Table 2 on p. 270]. Unfortunately, the case $k = 20$ is slightly different, and we have not attempted to derive such an upper bound in this case. However, when we inspect the size of the solutions found for $k = 20$, it seems that the largest cube covered in this case is comparable with the largest cubes covered in the

TABLE 2. New solutions of (1)

k	$\epsilon(\theta = \sqrt[3]{k})$	K	$\lvert x\rvert, \lvert y\rvert, \lvert z\rvert \le$	x	y	z
2	$\theta - 1$	1000	1.29×10^8	1214928	3480205	−3528875
3	$\theta^2 - 2$	1500	1.35×10^8		none	
20	$-\frac{1}{2}\theta^2 + \theta + 1$	1000		3049	8427	−8558
				99637	607191	−608084
				136912	264145	−275877
				−305081	−523091	555618
				−378203	−555737	608880
				−2006066	−3431087	3645939
				−3633722	−9161277	9348001
				15670213	40439559	−41209136
				−89598233	−374850480	376549093
30	$-3\theta^2 + 9\theta + 1$	2000	1.64×10^6		none	
39	$2\theta^2 - 23$	1000	3.15×10^5	134476	117367	−159380
42	$12\theta^2 - 42\theta + 1$	1000	1.57×10^4		none	

cases $k = 2$ and $k = 3$. We only present the *new* solutions found and not those which were already given in [6 and 2] (with one exception: the smallest solution we list for $k = 20$ was not explicitly given in [2], but in an accompanying table which was deposited in the UMT-file of *Mathematics of Computation*).

The total amount of CPU-time for the computation of Table 2 was about 30 hours. To give an impression of the speed of our program: the job for $k = 30$, $r = 1$, $K = 2000$, consumed 3934 seconds CPU-time on the Cyber 205, 65% of which was spent on the solution of the equation $wx \equiv 1 \bmod n'$ (step 3b of the algorithm given in §2). The total number of (a, b, c)-triples treated in this job was about 7.12×10^8.

All new solutions were found several times, for different combinations of a, b, and c. Of course, this corresponds to using different associates in $\mathbb{Q}(\sqrt[3]{k})$. For example, the solution for $k = 20$: $(x, y, z) = (136912, 264145, -275877)$ was found three times, viz., for $(a, b, c) = (-47, 8, 18)$, $(53, -129, 81)$, and $(443, 170, 121)$. For $k = 2$, the solution $(x, y, z) = (1214928, 3480205, -3528875)$ was found for $(a, b, c) = (165, -12, 16)$ and for five other (a, b, c)-triples.

In order to see how fast the length of the vectors in the algorithm for the computation of $\mathbf{w}^{-1}$ **mod n** tends to zero, we have counted the number of steps of this algorithm for $k = 3$, $K = 255$ (which is representative for all our experiments) so that $|a| \le 255$, $|b| \le 176$, and $|c| \le 122$. Since $a \equiv 2 \bmod 3$, the number of (a, b, c)-triples to be handled is

$$\lceil (2 \times 255 + 1)/3 \rceil \times (2 \times 176 + 1) \times (2 \times 122 + 1) = 14{,}788{,}935.$$

Among these, there is no case with $n = 0$, there are 851 cases with $w \bmod n = 0$ (most of these have $w = 0$), and there are 1,671 cases with $w \bmod n = 1$, so that $w^{-1} \bmod n = 1$. For the remaining 14,786,413 cases, the continued fraction algorithm found 10,390,393 cases with gcd $= 1$ (70.3%) and 4,396,020 cases (29.7%) with gcd > 1. In about 63% of the 10,392,064 cases for which $w^{-1} \bmod n$ could be computed, we found $D < 0$ in step 5 of our algorithm. In only six cases, D was the square of an integer, yielding the solution $(x, y, z) = (4, 4, -5)$ six times. Table 3 gives the distribution of the numbers of steps in the continued fraction algorithm to find the gcd, including percentages, and

TABLE 3. Number of cases with i steps in the continued fraction expansions, for $i = 1, \dots, 20$

i	$\#(i)$	percentage	cum. percentage
1	16985	0.1	0.1
2	54291	0.4	0.5
3	190366	1.3	1.8
4	442659	3.0	4.8
5	831022	5.6	10.4
6	1412828	9.6	19.9
7	1924764	13.0	33.0
8	2302600	15.6	48.5
9	2336986	15.8	64.3
10	2023481	13.7	78.0
11	1496212	10.1	88.1
12	934761	6.3	94.5
13	494258	3.3	97.8
14	216375	1.5	99.3
15	78800	0.5	99.8
16	23285	0.2	100.0
17	5541	0.0	100.0
18	1018	0.0	100.0
19	163	0.0	100.0
20	18	0.0	100.0

cumulative percentages. The average number of steps is 8.59. If c_i is the cumulative percentage entry in the row numbered i, then the vector length after i steps of the algorithm is given by $100 * (100 - c_i)$ (the original length is 10,000).

We would like to compare our results with the theoretical results which are known about the number of steps needed in the Euclidean algorithm [4, §§4.5.2 and 4.5.3].

If w and n are random numbers, the probability that they are relatively prime is $6/\pi^2$ (≈ 0.608) [4, §4.5.2, Theorem C]. However, since $a \equiv 2 \bmod 3$, we have $n \equiv 2 \bmod 3$, so $\gcd(w, n)$ cannot be a multiple of 3. Following [4, §4.5.2, Exercise 13], we find that, given two random positive integers which do not have a common divisor 3, the probability that they are relatively prime is $27/(4\pi^2) \approx 0.684$. This agrees better with the fraction of 0.703 of the total number of pairs (w, n) we found to be relatively prime. The difference may be explained by the fact that of course the numbers w and n are not random. Moreover, we have observed that certain primes like 7, 13, 19, 31 turn out *not* to occur as gcd-values in our experiments. This is easily proved for a given prime p by checking all the possible residue classes mod p for w and n.

Now we turn to the number of steps in the Euclidean algorithm. The values which can be assumed by $w = b^2 - ac$ and $n = a^3 + 3b^3 + 9c^3 - 9abc$ for $|a| \le K$, $|b| \le K/\theta$, and $|c| \le K/\theta^2$ ($K = 255$, $\theta = \sqrt[3]{3}$), are approximately $-(K/\theta)^2 < w < 2(K/\theta)^2$ and $-4K^3 < n < 4K^3$. Hence, n is generally much larger than w, so we start by reducing n mod w in order to get numbers which are generally of the same size (and we add 1 to the number of steps in our analysis). For $w \approx 2(K/\theta)^2$ (≈ 62521) the maximum number of steps is bounded by [4, Summary of §4.5.3] $1 + \lceil 4.8 \log_{10} w - 0.32 \rceil \leq 24$, which agrees with the number 20 observed in our computations.

An upper bound for the *average* number of steps is [loc. cit.] $1 + 1.9405 \log_{10} w$. For $w = 2(K/\theta)^2$ this yields 10.3. We can expect that $|w|$ lies in the interval $[1, (K/\theta)^2)$ with a probability which is larger (at least by a factor of 2) than the probability that w lies in the interval $((K/\theta)^2, 2(K/\theta)^2)$. In fact, we observed that about 92.5% of the values of w are $< (K/\theta)^2$.

The actual average number of steps 8.6 we found corresponds to $w \approx \frac{1}{4}(K/\theta)^2$, which is heuristically consistent with the quadratic form of the formula for w.

ACKNOWLEDGMENTS

We are grateful to the referee for his criticism, which helped to improve this paper. We acknowledge the provision of computer time on the Cyber 205 by the former Dutch Working Group on the Use of Supercomputers (the present Dutch National Computing Facilities Foundation). The second author wishes to thank Dik Winter in particular for his very informative discussions on double-precision arithmetic.

BIBLIOGRAPHY

1. J. W. S. Cassels, *The rational solutions of the Diophantine equation* $Y^2 = X^3 - D$, Acta Math. **82** (1950), 243–273.
2. V. L. Gardiner, R. B. Lazarus, and P. R. Stein, *Solutions of the Diophantine equation* $x^3 + y^3 = z^3 - d$, Math. Comp. **18** (1964), 408–413.
3. D. R. Heath-Brown, *Searching for solutions of* $x^3 + y^3 + z^3 = k$, Sém. Théorie des Nombres, Paris 1989–1990 (D. Sinnou, ed.), Birkhäuser, Boston, 1992, pp. 71–76.
4. Donald E. Knuth, *The art of computer programming,* Vol. 2, *Seminumerical algorithms*, Addison-Wesley, Reading, MA, 1981.
5. D. H. Lehmer, *On the Diophantine equation* $x^3 + y^3 + z^3 = 1$, J. London Math. Soc. **31** (1956), 275–280.
6. J. C. P. Miller and M. F. C. Woollett, *Solutions of the Diophantine equation* $x^3 + y^3 + z^3 = k$, J. London Math. Soc. **30** (1955), 101–110.
7. L. J. Mordell, *On an infinity of integer solutions of* $ax^3 + ay^3 + bz^3 = bc^3$, J. London Math. Soc. **30** (1955), 111-113.
8. M. Scarowsky and A. Boyarsky, *A note on the Diophantine equation* $x^n + y^n + z^n = 3$, Math. Comp. **42** (1984), 235–237.
9. J. J. F. M. Schlichting, *Double precision BLAS*, Algorithms and Applications on Vector and Parallel Computers (H. J. J. te Riele, Th. J. Dekker, and H. A. van der Vorst, eds.), North-Holland, Amsterdam, 1987, pp. 229–249.

(Heath-Brown) MAGDALEN COLLEGE, OXFORD, OX1 4AU, ENGLAND

(Lioen and te Riele) CENTRE FOR MATHEMATICS AND COMPUTER SCIENCE (CWI), KRUISLAAN 413, 1098 SJ AMSTERDAM, THE NETHERLANDS
E-mail address, W. M. Lioen: walter@cwi.nl
E-mail address, H. J. J. te Riele: herman@cwi.nl