1991

D.R. Heath-Brown, W.M. Lioen, H.J.J. te Riele

On solving the Diophantine equation $x^3+y^3+z^3=k$ on a vector computer

Department of Numerical Mathematics Report NM-R9121 December

On Solving the Diophantine Equation $x^3 + y^3 + z^3 = k$ on a Vector Computer

D.R. Heath-Brown
Magdalen College, Oxford, OX1 4AU, England

W.M. Lioen, H.J.J. te Riele
CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

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 non-zero 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 $w_i x_i \equiv 1 \bmod n_i$, $i = 1, 2, \cdots$ at vector speed. Moreover, the basic double precision arithmetic operations $(+,-,\times,/)$ 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 $|x|$, $|y|$, $|z| \leq 10^8$; this considerably extends an earlier search in the cube $|x|$, $|y|$, $|z| \leq 2^{16}$. 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 exist. 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 $(k = 30, 33, 39, 42, \cdots)$ for which no solution is known (yet).

1991 Mathematics Subject Classification: 11D25, 11Y50.
Keywords & Phrases: Cubic Diophantine equation, vector computer, Euclidean algorithm.

Report NM-R9121
ISSN 0169-0388
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

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 $|x|, |y|, |z| \leq N$, 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 $k = 3$, and significant changes have to be made for other values of k, depending mainly on the class number of $\mathbb{Q}(\sqrt[3]{k})$.

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

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

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

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 [5] and [2] solutions of (1) were computed by means of a straightforward algorithm which for given z and k checks whether all the possible combinations of values of x and y in a chosen range satisfy (1). The range chosen in [2] (which includes the one chosen in [5]) was:

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

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 a lot of values of k for which no solution at all is known. Those below 100 (and $\not\equiv \pm 4 \bmod 9$) are:

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

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 [5], [6] and [4].

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], [5] and [7]). 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 Section 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 $(b^2 - ac)^{-1} \bmod n$ (Section 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 [8]. The results of our computations are listed in Section 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

$$1 \leq |z| \leq |x+y| \leq |\epsilon| K^3, \qquad (3)$$

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

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:

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

$$|a|,\ \theta|b|,\ \theta^2|c| \leq K r^{1/3},$$

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).

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 $r|b$; 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 $1 \leq |z| \leq |n|$ 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

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

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

k	r	*restrictions on a, b, c*
2	1	$a+2b+4c \equiv 1$ or $2 \bmod 6$
		and either $2 \not| a$
		or $2\|a$, $4 \not| a$, $b \equiv 1 \bmod 4$
		or $4\|a$, $b+2c \equiv 1 \bmod 4$
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	$a \equiv 0$, $b \equiv 0$, $c \equiv 1 \bmod 3$

Table 1: Values of k, r, a, b, and c

2a. $2 \not| a,\ 3 \not| a-(b+c)$:

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

2b. $2 \not| b,\ 3 \not| c-(a+b)$:

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

2c. $2 \not| c,\ 3 \not| a+b+c$:

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

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 Section 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 $\vec{n}$ and $\vec{w}$ with components n_i and w_i respectively, and which computes a vector $\vec{u}$ with components u_i such that $w_i u_i \equiv 1 \bmod n_i$. This is described in Section 3.1.

In Step 4 the product $v \cdot \overline{w}$ becomes too large for the normal integer capacity of the Cyber 205. Therefore, we have written a special vectorized version of the modular multiplication, which returns an integer result vector.

The integers involved in Steps 5 and 6 may become so large that in general they do not fit in the normal integer capacity of the Cyber 205 (which is 48 bits). Therefore, we have written special routines for double precision arithmetic operations on the Cyber 205 (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 [8], 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 strides two in vector operations on the double precision numbers. In order to avoid this in our implementation, we have stored the upper and lower parts in two *separate* arrays.

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

3.1 Solving the equation $wx \equiv 1 \bmod n$ in step 3

For given w and n, the scalar equation $wx \equiv 1 \bmod n$, where $\gcd(w, n) = 1$ and $1 \leq w < n$, can be solved as follows. We 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. In order to compute the c.f. of w/n 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 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 = [b/a]
     r = b − q×a
     if r = 0 then
c now a contains gcd(w,n)
c if a = 1 return sign×d1 else 0 into u
        if a = 1 then
           u = sign×d1
        else
           u = 0
        endif
        return
     endif
     h = q×d1 + d0
     d0 = d1
     d1 = h
     b = a
     a = r
     sign = −sign
     goto 10
```

For the *vectorization* of this algorithm, we have to keep in mind the following basic principle. We want to do the same operations on all vector elements. To that end we construct vectors for **n** and **w** (in the remainder of this section all vectors are printed in boldface) and some temporaries, and keep an index vector **ind** up to date to administrate for which components a solution has been found. This index-vector is initialized (before the label 10 – line) as follows:

```
     do 5 i=1, len
  5     ind(i) = i
```

All the scalar variables of the algorithm, except 'sign', are turned now into vectors of length 'len', and each scalar operation in this algorithm becomes a vector operation. Of course, the number of steps to be taken before the algorithm terminates is generally not the same for the different components (this number of steps roughly varied between 10 and 30 for the numbers we treated in our jobs).

After the execution of the line: '**r**=**b**−**q**×**a**', a *bit* vector called **mask** of length len is filled, each component of which becomes '0' if the corresponding component of **r** is zero, and '1' otherwise. With the aid of **mask**, the computed inverses are transported from **d1** into **u** in two steps, viz., a *compress* step and a *scatter* step. These replace the 'if r = 0 then' – part of the above scalar algorithm. (For simplicity, we assume here that all the gcd's of the components of the input vectors **n** and **w** are 1, i.e., those components of **a** are 1, for which the corresponding components of **mask** are zero. In our program this complication is handled by the use of a second bit vector.)

- First (*compress* step), those components of **ind** are stored into an auxiliary vector called **order** for which the corresponding elements of **mask** are '0'. This operation can be done very efficiently on the Cyber 205 with a so-called compress instruction. A similar compress operation (also governed by **mask**) is carried out on **d1**, where the output vector is an auxiliary vector called **t**. If sign $= -1$ then **t** is replaced by $-$**t**.

- Next (*scatter* step), the elements of **t** are scattered into **u** where the location in **u** of each component of **t** is determined by the corresponding component of **order**. Like the compress instruction, a scatter instruction is available on the Cyber 205 to do this very efficiently.

The Cyber 205 has a clock cycle period of 20 nanoseconds ($= 20 \times 10^{-9}$ sec.). Adding two long vectors takes 1/p clock cycles per element on a p-pipe Cyber 205. Compressing a vector is done at the same speed, so compressing a vector of length len takes approximately len$\times 10^{-8}$ seconds on the 2-pipe Cyber 205 we used (we worked with len = 10,000). Scattering a vector of length len' into some target vector takes approximately len'$\times 3 \times 10^{-8}$ seconds, and this time is independent of the length of the target vector.

In order now to let the algorithm continue only with those components for which the inverses have not yet been found, a compress operation is carried out on the vectors **ind**, **q**, **r**, **a**, **d0**, and **d1**, where all those components are removed for which the corresponding components of **mask** are '0'. The length len is adapted accordingly. If after this compress operation len = 0, we are finished. Otherwise, the six update lines '**h=q×d1+d0**' ... 'sign = −sign' are executed and the program jumps back to label 10. It should be noted that most of the vector movement in these six lines can be done efficiently by operating on *pointers to the vectors* (on the Cyber 205, these are called vector descriptors) rather than on the vectors themselves.

3.2 Results

We have run our program for solving (1) for the values of k mentioned in Section 2, for various values of K. The results are listed in Table 2. Table 2 also gives, for $k = 2, 3, 30, 39, 42$, the size of the largest (x, y, x)-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 Section 2. The fundamental units ϵ were taken from [1, Table 2 on page 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 cases $k = 2$ and $k = 3$. We only present the *new* solutions found and not those which were given already in [5] 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 spent on the Cyber 205 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 Section 2). The total number of triples (a,b,c) 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 triples (a,b,c).

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	

Table 2: New solutions of (1)

Acknowledgement We acknowledge the provision of computer time on the Cyber 205 by the former Dutch Working Group on the Use of Supercomputers. The second author wishes to thank Dik Winter in particular for his very informative discussions on double precision arithmetic.

References

[1] J.W.S. Cassels. The rational solutions of the Diophantine equation $Y^2 = X^3 - D$. *Acta Math.*, 82:243–273, 1950.

[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.*, 18:408–413, 1964.

[3] D.R. Heath-Brown. Searching for solutions of $x^3 + y^3 + z^3 = k$. In *Sém. Théorie des Nombres, Paris 1989-1990.* Birkhäuser. to appear.

[4] D.H. Lehmer. On the Diophantine equation $x^3 + y^3 + z^3 = 1$. *J. London Math. Soc.*, 31:275–280, 1956.

[5] 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.*, 30:101–110, 1955.

[6] L.J. Mordell. On an infinity of integer solutions of $ax^3 + ay^3 + bz^3 = bc^3$. *J. London Math. Soc.*, 30:111–113, 1955.

[7] M. Scarowsky and A. Boyarsky. A note on the Diophantine equation $x^n + y^n + z^n = 3$. *Math. Comp.*, 42:235–237, 1984.

[8] J.J.F.M. Schlichting. Double precision BLAS. In H.J.J. te Riele, Th.J. Dekker, and H.A. van der Vorst, editors, *Algorithms and Applications on Vector and Parallel Computers*, pages 229–249. North-Holland, 1987.