Sums of squares – how many possibilities?

Arjen Markus

August 1, 2024

Introduction

It is well-known that any positive integer can be written as the sum of at most four squares, a fact that is known as *Lagrange's four square theorem.*[1] For instance:

$$\begin{aligned} 7 &= 2^2 + 1^2 + 1^2 + 1^2 \\ 10 &= 3^2 + 1^2 \\ 15 &= 3^2 + 2^2 + 1^2 + 1^2 \end{aligned}$$

Many integers can be written as such a sum in different ways:

$$\begin{aligned} 10 &= 3^2 + 1^2 \\ &= 2^2 + 2^2 + 1^2 + 1^2 \\ 50 &= 7^2 + 1^2 \\ &= 5^2 + 5^2 \\ &= 6^2 + 3^2 + 2^2 + 1^2 \end{aligned}$$

Let us be precise: if we ignore the order of the terms and allow only positive integers, then the following, somewhat lengthy program will print the sequences of up to four squares that add up to a given number:

```
program explicit_loops
    implicit none

    character(len=20) :: string
    integer           :: number, maxi, maxj, maxk, maxl, count
    integer           :: i, j, k, l

    if ( command_argument_count() < 1 ) then
        write( *, * ) 'What is the number?'
        read( *, * )  number
    else
```

[1] https://mathworld.wolfram.com/LagrangesFour-SquareTheorem.html

```
    call get_command_argument( 1, string )
    read( string, * ) number
endif

write( *, * ) 'Number to be written as a sum of squares:', number

maxl = isqrt(number)  ! Make sure we have the largest possible integer

count = 0
do l = maxl,1,-1

    if ( l**2 == number ) then
        count = count + 1
        write(*,*) count, ':', l
        cycle
    endif

    maxk = isqrt(l**2)
    do k = maxk,1,-1

        if ( l**2 + k**2 == number ) then
            count = count + 1
            write(*,*) count, ':', l, k
            cycle
        endif

        maxj = isqrt(k**2)
        do j = maxj,1,-1

            if ( l**2 + k**2 + j**2 == number ) then
                count = count + 1
                write(*,*) count, ':', l, k, j
                cycle
            endif

            maxi = isqrt(j**2)
            do i = maxi,1,-1

                if ( l**2 + k**2 + j**2 + i**2 == number ) then
                    count = count + 1
                    write(*,*) count, ':', l, k, j, i
                endif
            enddo
        enddo
    enddo
enddo
```

```
    write(*,*) 'Total number of solutions: ', count
contains

integer function isqrt( n )
    integer, intent(in) :: n

    isqrt = int( sqrt(real(n)) )

    !
    ! Take care of possible rounding errors (if n is very large)
    !
    if ( (isqrt+1)**2 == n ) then
        isqrt = isqrt + 1
    endif
end function isqrt

end program explicit_loops
```

(Note that for large integers, determining the largest square that is equal or smaller than the given integer via `int( sqrt( real(n) ) )` may produce too small a value, due to truncation. The function `isqrt` is meant to prevent that from happening.)

It works as follows:

- Given a number n, determine the largest integer l such that $l^2 \leq n$. This is the start of the first loop. All other integers to be squared should be equal or lower than the index of the first loop.
- The difference $n - l^2$ determines the start of the second loop: $k^2 \leq (n - l^2)$ and so on.
- Print the result if the sum is equal to the given number n.

It will produce a list of all the combinations, as required (see Table 1 for several values of n).

1 Recursive solution

The code of the program in the previous section is repetitive as the four integers are each determined in their own loop and therefore it would be tedious to extend it to, say, the sum of cubes – every integer can be written as the sum of at most nine cubes or 19 fourth-powers.[2]. So, let's look at an alternative, using recursion:

[2] `https://en.wikipedia.org/wiki/Waring%27s_problem`

Table 1: Number of solutions (N) for a few integers (n).

n	N	n	N	n	N
1	1	11	1	30	2
2	1	12	2	40	2
3	1	13	2	50	5
4	2	14	1	60	3
5	1	15	1	70	5
6	1	16	2	80	2
7	1	17	2	90	9
8	1	18	3	100	7
9	2	19	2	200	5
10	2	20	2	300	13

```
program recursive_solution
    implicit none

    character(len=20) :: string
    integer           :: number, maxvalue, count
    integer           :: values(0)

    ... as in the first program

    write( *, * ) 'Number to be written as a sum of squares:', number

    maxvalue = isqrt(number)  ! Make sure we have the largest possible integer

    count = 0
    call determine_solutions( number, maxvalue, count, values )

    write(*,*) 'Total number of solutions: ', count
contains

recursive subroutine determine_solutions( number, maxvalue, count, values )
    integer, intent(in)    :: number
    integer, intent(in)    :: maxvalue
    integer, intent(inout) :: count
    integer, intent(in)    :: values(:)

    integer                :: i
    integer                :: maxnext

    do i = maxvalue,1,-1
```

```
        if ( size(values) < 4 ) then
            if ( number == i**2 ) then
                count = count + 1
                write(*,*) count, ':', [values, i]

            elseif ( number > i**2 ) then
                maxnext = isqrt(i**2)
                call determine_solutions( number-i**2, maxnext, count, [values, i] )
            endif
        endif
    enddo
end subroutine determine_solutions

... function isqrt is the same

end program recursive_solution
```

This program works in much the same way but instead of a nested do-loop, it calls itself with adjusted arguments. One of these arguments is the list of integers that should add up to the given number n. But it only does so if the list is short enough: we want four integers, no more.

2 Sums of more than four squares

The recursive construction allows us to easily vary the program so that other mathematical problems can be solved:

- How can a number be written as at most nine cubes?
- What if you allow any number of squares?

To illustrate the latter mathematical problem, the number 10 can be written as:

$$\begin{aligned} 10 &= 3^2 + 1^2 \\ &= 2^2 + 2^2 + 1^2 + 1^2 \\ &= 2^2 + 1^2 + ... + 1^2 \\ &= 1^2 + ... + 1^2 \end{aligned}$$

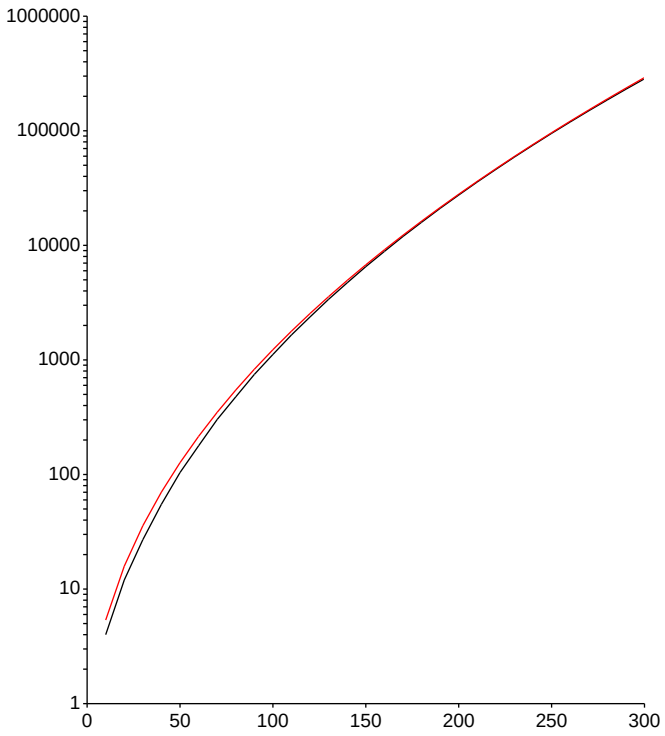

Figure 1: Number of solutions for the sum of any number of squares equal to the given value. See text for further information.

Table 2 shows the number of solutions for a few values of n. As can be seen, the number grows very fast (see also Figure 1). The red line in the figure was estimated visually, the number of solutions is approximately, at least up to $n = 300$:

$$N(n) \quad \approx \quad 0.5 \; e^{n^{0.46}}$$

This mathematical problem also leads to an interesting programming problem: with the recursive solution we consume more and more memory and at some point (for n large enough), the stack memory is exhausted. Experimentation shows that this happens around $n = 3000$ (simply set the variable `maxvalue` to 1 or 2 instead of `isqrt(n)` and enter a largish number) – the program dies without printing the number of solutions.

The amount of memory taken from the stack for each recursion can be limited either by compile options (to take the required memory for the constructed array

Table 2: Number of solutions (N) for the sum of any number of squares for a few integers (n).

n	N	n	N	n	N
10	4	110	1653	210	35750
20	12	120	2374	220	46030
30	27	130	3385	230	59027
40	55	140	4714	240	75005
50	104	150	6521	250	94987
60	178	160	8849	260	119360
70	302	170	11953	270	149482
80	476	180	15895	280	185960
90	747	190	21031	290	230580
100	1116	200	27482	300	284316

from the heap) or by using a fixed array and counting the number of integers separately. A more interesting solution is to combine iteration and recursion.

```
program iterative_unlimited
    implicit none

    character(len=20)    :: string
    integer              :: idx, number, maxvalue, count, residue
    integer, allocatable :: values(:)

    ... same as the other programs

    write( *, * ) 'Number to be written as a sum of squares:', number

    allocate( values(number) ) ! Provide the storage

    count = 0
    idx   = 1

    values    = 0
    values(1) = maxvalue

    do
        residue  = number - sum(values(1:idx)**2)

        if  ( residue > 0 ) then
            maxvalue = min( isqrt(residue), values(idx) )

            if ( maxvalue > 0 ) then
                idx = idx + 1
```

```
            values(idx) = maxvalue
        endif

    elseif ( residue == 0 ) then
        count = count + 1
        write(*,*) count, ':', values(1:idx)

        do while ( values(idx) > 0 )
            values(idx) = values(idx) - 1

            if ( values(idx) <= 0 ) then
                idx = idx - 1
            else
                exit
            endif
        enddo

    else
        if ( values(idx) > 1 ) then
            values(idx) = values(idx) - 1
        else
            idx = idx - 1
        endif
    endif

    if ( idx <= 0 ) then
        exit
    endif
enddo

write(*,*) 'Total number of solutions: ', count
contains
...
end program iterative_unlimited
```

The program uses a fixed amount of memory and uses a work array `values` to keep track of the solution and the candidates. Via the indeterminate loop together with the index `idx` by which the work array is adjusted, the program iterates over all sets of integers v_i such that the following conditions hold:

$$\begin{array}{rcll} v_{i+1} & \leq & v_i & i < n \\ \Sigma v_i^2 & \leq & n & \end{array}$$

Note that this means that if v_i becomes zero, the index i has to be decreased.

The logic is more complicated, but it is now (in principle) possible to determine the number of solutions for very large values of n.