Pythagorean triplets - list nr 2

Authors: Jakubik Małgorzata, Szymkowiak Magdalena

In [13]:

```
import math
```

Brute-Force method - trying every possibility

In [ ]:

```
def pytha_triplet(nums_sum:int):  #n^3
    """
    Find the Pythagorean triplet from the given number
     which is a sum of a,b,c.
    Iterate over three nested loops.

    :param num_sum(int): a number which is the sum of a,b,c
    :return: tuple with 'True' (if wanted triplet exists) or 'False' (if does
     not exist), the length of sides of triplet in increasing order and
     the number of operations executed in program
    """
    operation_count=0
    for a in range(1,nums_sum):
        for b in range(1,nums_sum):
            for c in range(1,nums_sum):
                operation_count+=9
                if a**2 + b**2 == c**2 and a+b+c==nums_sum:
                    return(True,a,b,c,operation_count,end-start)
    return (False,-1,-1,-1,operation_count,end-start)
```

In [126…

```
pytha_triplet(1000)
```

Out[126…

```
(True, 200, 375, 425, 1790786250)
```

Nested loops, but range for b is limited. It includes a<b<c.

In [ ]:

```
def pytha_triplet1(nums_sum:int):
    """
    Find the Pythagorean triplet from the given  number which
     is a sum of a,b,c.
    Iterate over two nested loops, where second loop is limited
     according to a<b.

    :param num_sum(int): a number which is the sum of a,b,c
    :return: tuple with 'True' (if wanted triplet exists) or 'False' (if does
     not exist), the length of sides of triplet in increasing order and
     the number of operations executed in program
    """
    operation_count = 0
    for a in range(1, nums_sum):
        for b in range(a+1, nums_sum):
            c = nums_sum - a - b
            operation_count += 8
            if a**2 + b **2 == c**2:
                return (True, a, b, c, operation_count)
    return (False, -1, -1, -1, operation_count)
```

In [127…

```
pytha_triplet1(1000)
```

Out[127…

```
(True, 200, 375, 425, 1432608)
```

Another limits added to range for a and b

In [14]:

```
def pytha_triplet2(nums_sum:int):  #n^2
    """
    Find the Pythagorean triplet from the given number which
     is a sum of a,b,c.
    Iterate over two nested loops, where loops are limited
     according to 'a<b<c'.

    :param num_sum(int): a number which is the sum of a,b,c
    :return: tuple with 'True' (if wanted triplet exists) or 'False' (if does
     not exist), the length of sides of triplet in increasing order and
     the number of operations executed in program
    """
    operation_count=0
    for a in range(1,nums_sum//3):
        for b in range(a+1,nums_sum//2):
            c=nums_sum-a-b
            operation_count+=14
            if a**2 + b**2 == c**2 and a+b+c==nums_sum:
                return(True,a,b,c,operation_count)

    return (False,-1,-1,-1,operation_count)
```

In [128…

```
pytha_triplet2(1000)
```

Out[128…

```
(True, 200, 375, 425, 1114064)
```

Analytical approach

$$a^2 + b^2 = c^2a^2 + 2ab + c^2 = c^2 + 2ab(a + b)^2 = c^2 + 2ab$$

Assuming n is the perimeter of a triangle

$$(n - c)^2 = c^2 + 2ab$$

Having

$$2ab = (n - c)^2 - c^2 \wedge a^2 + b^2 = c^2$$

we can get

$$(a - b)^2 = c^2 - n^2 + 2nc$$

We know that $a - b < 0$ so using $a + b + c = n$ we have

$$b = \frac{n - c - (a - b)}{2} \wedge a = n - b - c$$

In [16]:

```
def pytha_triplet3(nums_sum:int): #n
    """
    Find the Pythagorean triplet from the given number which
    is a sum of a,b,c.
    Apply the analytical method and iterate over range of 'c'.

    :param num_sum(int): a number which is the sum of numbers: a,b,c
    :return: tuple with 'True' (if wanted triplet exists) or 'False' (if does
     not exist), the length of sides of triplet in increasing order and
     the number of operations executed in program
    """
    operation_count=0
    for c in range(1,nums_sum):
        a_b_sqr= c**2 - nums_sum**2 + 2*nums_sum*c
        a_b= math.floor(math.sqrt(abs(a_b_sqr)))
        operation_count+=9

        if a_b**2 == a_b_sqr:
            b=(nums_sum - c + a_b)//2
            a= nums_sum - b - c
            operation_count+=7
            return(True,a,b,c,operation_count)
        else:
            operation_count+=2

    return (False,-1,-1,-1,operation_count)
```

In [8]:

```
pytha_triplet3(1000)
```

Out[8]:

```
(True, 200, 375, 425, 4680)
```

Testing another cases:

In [131…

```
pytha_triplet(56)
```

Out[131…

```
(True, 7, 24, 25, 174960)
```

In [132…

```
pytha_triplet1(56)
```

Out[132…

```
(True, 7, 24, 25, 2608)
```

In [133…

```
pytha_triplet2(56)
```

Out[133…

```
(True, 7, 24, 25, 2212)
```

In [134…

```
pytha_triplet3(56)
```

Out[134…

```
(True, 7, 24, 25, 255)
```

In [138…

```
pytha_triplet(-56)
```

Out[138…

```
(False, -1, -1, -1, 0)
```

In [135…

```
pytha_triplet(20)
```

Out[135…

```
(False, -1, -1, -1, 61731)
```

Comparing the amount of time of finding a triplet:

In [121…

```
%timeit pytha_triplet(1000)
```

```
2min 20s ± 2.4 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [123…

```
%timeit pytha_triplet1(1000)
```

```
139 ms ± 10.3 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [19]:

```
%timeit pytha_triplet2(1000)
```

```
106 ms ± 2.47 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [17]:

```
%timeit pytha_triplet3(1000)
```

```
1.03 ms ± 34 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

To sum up

Taking into considerations, every method is an improvement of the previous one. Reductions of the loops make our function better and analitical approach allow to have only one loop and achive the most effective program with the smallest number of operations.

Link to our code:
https://github.com/maggszy/date_algorithm/blob/main/list2/pitagoras.py