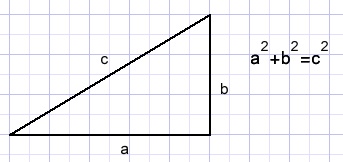
Pythagorean triples

In secondary school students are confronted with the theorem of Pythagoras:



In a right angled triangle, the square of the hypothenuse c equals the sum of squares of the right angled sides a and b.

Calculations with above formula usually result in roots, numbers that only may be approximated.

Some values of a and b however result in an integer value of c, such as

* 3,4 🡪 c=5
* 5,12 🡪 c=13

(3,4,5) and (5,12,13) are called Pythagorean triples.

The question arises: are there more triples?

This Delphi project was written to find all triples below 1000.

**Analogous triples**

(3,4,5) being a triple involves that also (6,8,10), (9,12,15) .. are triples of similar triangles.

A simple criterion GCD(a,b) = 1 eliminates these analogous triples.

function GCD(a,b : word) : word; //greatest common devisor

var h : word;

begin

repeat

if a < b then

begin

h := a;

a := b;

b := h;

end;

a := a mod b;

until a = 0;

result := b;

end;

This function uses the Euclidean lemma: GCD(a,b) = GCD(a mod b,b).

Example: GCD(77,21) = GCD(14,21)=GCD(21,14)=GCD(7,14)=GCD(14,7)=GCD(0,7)=7

**Time measurement.**

A microseconds timer component is added to the project.

To find the real processing time without the burden of reporting (memo1.lines.add( *string*))

detected triples are first stored in array **ptriples**[ ]

type TP3 = record

a,b,c : word;

end;

…

var p3nr : byte; //sequence number of triple

ptriples : array[1..200] of TP3;

The project presented here has three selectable procedures to find Pythagorean triples.

**Method nr 1.**

Uses no floating point operations and a preset table of squares.

Var squares : array[1..1500] of dword;

…

procedure presetsquares;

var i : word;

begin

for i := 1 to 1500 do squares[i] := i\*i;

end;

**Summary:**

Var c2 : dword;

…

{a and b are incrementing variables in nested repeat..until loops}

C2 := squares[a] + squares[b];

C := b;

{a third nested repeat..until loop increments c}

If c2 = squares[c] then …. //report a,b,c as new triple

This is the slowest method. Processing time is over 100 milliseconds.

**Method2**

procedure p3method2;

var a,b : word;

a2,b2 : dword;

c : single;

begin

form1.proctimer.start;

for a := 1 to 998 do

begin

a2 := a\*a;

for b := a+1 to 999 do

begin

b2 := b\*b;

c := sqrt(a2 + b2);

if frac(c) = 0 then

if GCD(a,b) = 1 then addtriple(a,b,round(c));

end;

end;

form1.proctimer.stop;

end;

This method has two nested for loops to increment a and b.

It uses floating point operations sqrt( ) and frac( ) to calculate the square root and extract the fraction.

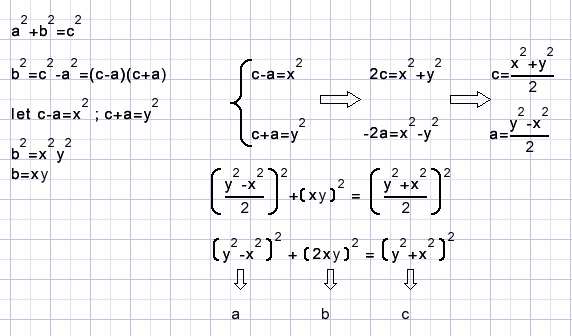
If frac(V) = 0, the value v is integer.

Processing time is around 40 milliseconds.

**Method3**

This method does not search for triples but uses formulas that yield triples.

Below is the theory:



a,b,c are replaced by x,y.

Each combination where x<> y generates a triple.

Processing time is 85 microseconds.

Unlike methods 1. and 2 were a and b were systematically incremented, the triples have to be sorted to obtain the same result sequence. A simple exchange sort procedure is used. Sorting time is not measured.

**Method selection**

A simple label is used as a button.

On the canvas of it’s parent (Form1) edges are painted which change color on an enter/leave event.

A left click on the label selects the next method, a right click selects the lower method.

Below a picture is shown of the project at work.

Please refer to the source code for details.

