//

Trilateration Equation

____

maXbox Starter 115 – Get Target-point from 4 Sensors.

"Time behaves like space – timeless.

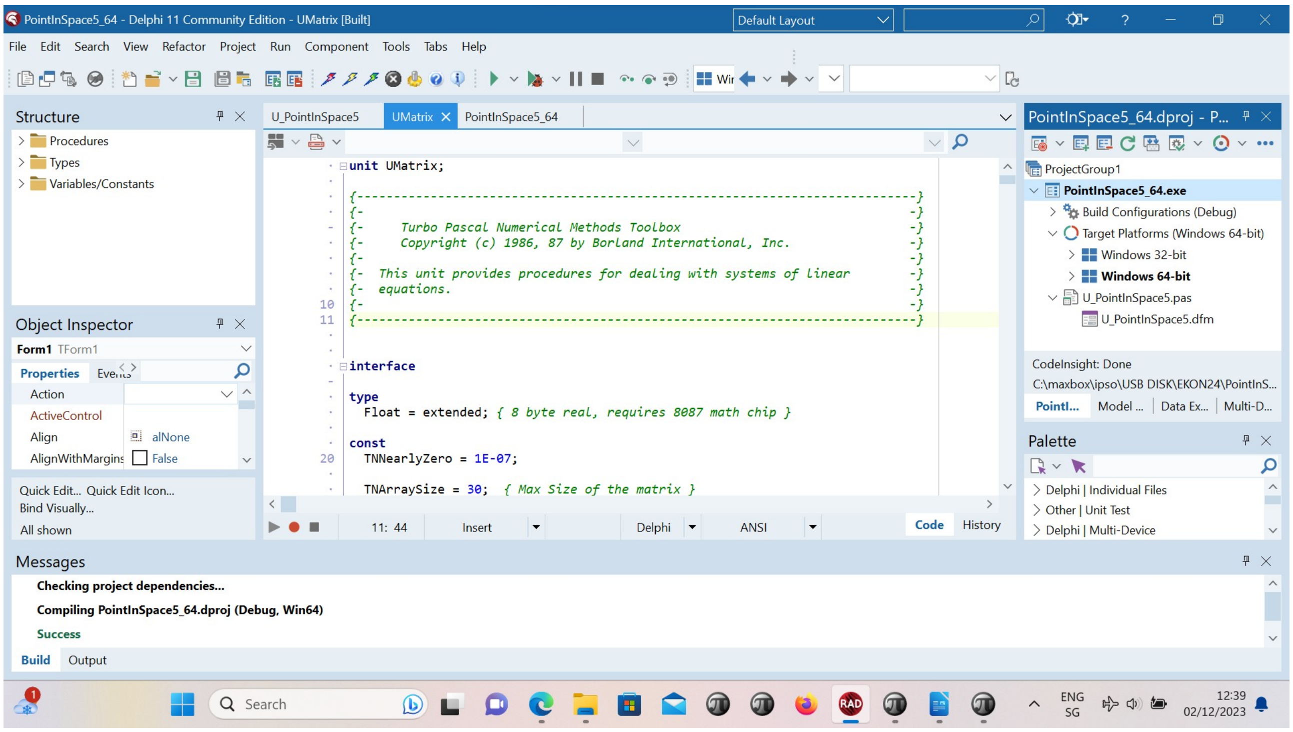
Source: 966_U_PointInSpace52_mX4Form2_64.pas & PointInSpace5_64.exe

If you have a code base of 32-bit Windows Delphi applications that you want to convert to 64-bit Windows, you should first do a reorganisation of the sources for an overview. Today we deal with an old Borland library *UMatrix.pas* and a mathematical problem with of course a solution too.

```
{-----------------------------------------------------------------------}
{-                                                                     -}
{-      Turbo Pascal Numerical Methods Toolbox                         -}
{-      Copyright (c) 1986, 87 by Borland International, Inc.          -}
{-                                                                     -}
{-   This unit provides procedures for dealing with systems of linear  -}
{-   equations.                                                        -}
{-                                                                     -}
{-----------------------------------------------------------------------}
```

Given the 3D coordinate locations of four sensors and their known distance from a target, calculate the location of the target.
Point From 4 Sensors (delphiforfun.org)

The source is organised in a project-, a form- and a calculation unit:



```
unit UMatrix;

{-----------------------------------------------------------------------}
{-                                                                     -}
{-      Turbo Pascal Numerical Methods Toolbox                         -}
{-      Copyright (c) 1986, 87 by Borland International, Inc.          -}
{-                                                                     -}
{-   This unit provides procedures for dealing with systems of linear  -}
{-   equations.                                                        -}
{-                                                                     -}
{-----------------------------------------------------------------------}


interface

type
  Float = extended; { 8 byte real, requires 8087 math chip }

const
  TNNearlyZero = 1E-07;

  TNArraySize = 30;  { Max Size of the matrix }
```

```
Checking project dependencies...
Compiling PointInSpace5_64.dproj (Debug, Win64)
Success
```

pic1: 115_dradstudio_code.png

So its not always that easy to open your old 32-bit application in the IDE, add and activate the 64-bit Windows target platform, and compile your application as a 64-bit target Windows application. We must search also our old dependencies and check the compatibility with components.

Source Organisation

Unit *UMatrix* is the unit from the old Borland Turbo Pascal Numeric Toolbox which contains the Gaussian Elimination procedure used here among other matrix operations.
So we have sensors at known locations in 3D space. Each sensor can supply distance information to a target but knows nothing about the target's direction. Alternatively, the sensors are Satellites and the target is a GPS receiver which reads very accurate time stamps transmitted by the satellites and calculates distances based on time offsets between its clock when a clock time message is received and satellites' clock time when the message was sent (as contained in the message).
Given the sensor (X, Y, Z) coordinates and their distance-to-target values, we'll use Gaussian Elimination to solve a set of linear equations describing the problem and derived as follows:

The distance Di to target at (Xt, Yt, Zt) for sensor "i" located at (Ai, Bi, Ci) is given by

$\mathrm{Sqr}(D_i)=\mathrm{Sqr}(A_i - X_t) + \mathrm{Sqr}(B_i - Y_t) + \mathrm{Sqr}(C_i - -Z_t)$

after expanding:

$D_i^2 = A_i^2 - 2A_iX_t + X_t^2 +B_i^2 - 2B_iY_t +Y_t^2 +C_i^2 - 2C_iZ_t + Z_t^2$

So for the reorganisation of the sources I have the latest revision with patches as from issues and it goes like this:

```
begin { procedure Gaussian_Elimination }
  Initial(Dimen, Coefficients, Constants, Solution, Error);
  if Dimen > 1 then begin
    UpperTriangular(Dimen, Coefficients, Constants, Error);
    if Error = 0 then
      BackwardsSub(Dimen, Coefficients, Constants, Solution);
  end;
end; { procedure Gaussian_Elimination }
```

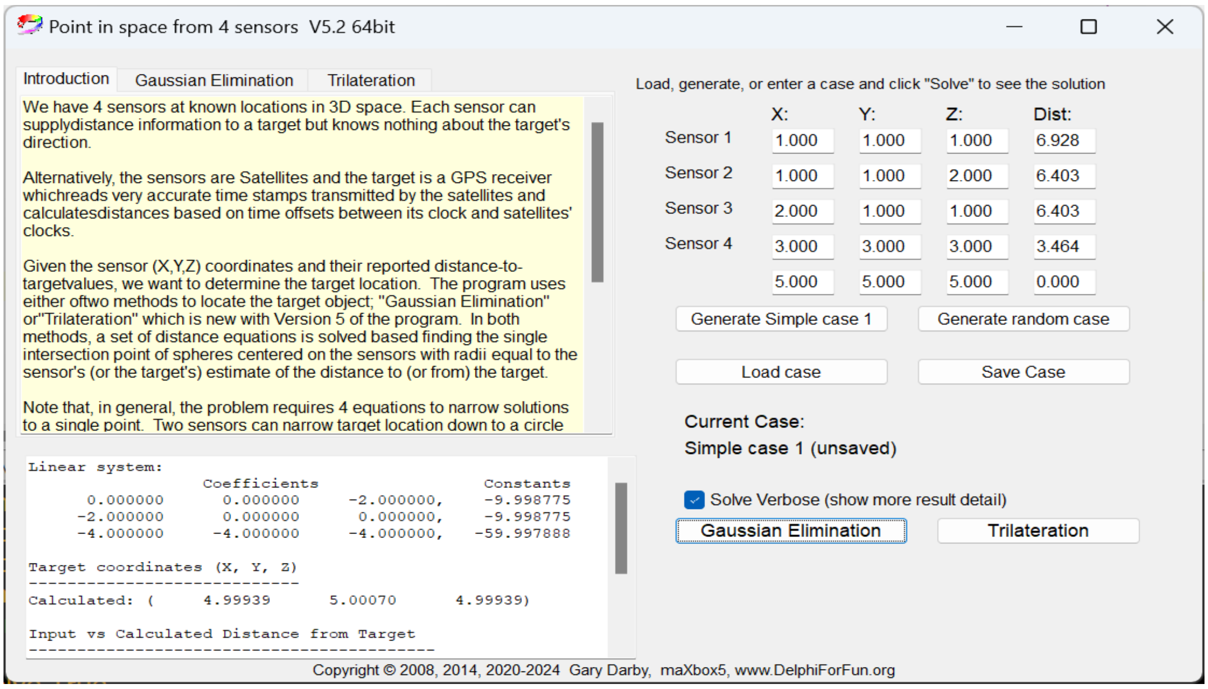
Solving this system of quadratic equations can be tricky, but we can subtract the Sensor1 equation from each of the other 3 to obtain linear equations like the following example for Sensor2::

$2(A_1-A_2)X_t + 2(B_1-B_2)Y_t + 2(C_1-C_2)Z_t = D_2^2-A_2^2-B_2^2-C_2^2-D_1^2$

The resulting 3 equations in 3 unknowns form a system of linear equations which are solved here using Gaussian Elimination to find the (Xt, Yt, Zt) target coordinates.
Note that the original problem requires 4 equations to resolve the 3 unknowns (the x, y, and z coordinates of the target). Two sensors can narrow target location down to a circle (the intersection of 2 spheres with target distances as radii), the 3rd sensor narrows the location down to two possible points, (the intersection of the circle with the 3rd

sensor's sphere circle).
The 4th sensor should resolve which of those two points is the target. Any error in specified locations or distances would either have no solution or require that some input values be adjusted. The techniques applied here result in reported distances being adjusted to produce a solution. Differences between input and calculated distances are listed as part of the solution. Lets have a look at the init routine *Initial* which is called in the main by Gaussian Elimination:

```
procedure Initial(Dimen           : integer;
              var Coefficients : TNmatrix;
              var Constants    : TNvector;
              var Solution     : TNvector;
              var Error        : byte);

{-----------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                   -}
{- Output: Solution, Error                                 -}
{-                                                         -}
{- This procedure test for errors in the value of Dimen.   -}
{- This procedure also finds the solution for the          -}
{- trivial case Dimen = 1.                                 -}
{-----------------------------------------------------------}

begin
  Error:= 0;
  if Dimen < 1 ten
    Error:= 1
  else
    if Dimen = 1 then
      if ABS(Coefficients[1, 1]) < TNNearlyZero then
        Error:= 2
      else
        Solution[1]:= Constants[1] / Coefficients[1, 1];
end; { procedure Initial }
```



We have 4 sensors at known locations in 3D space. Each sensor can supplydistance information to a target but knows nothing about the target's direction.

Alternatively, the sensors are Satellites and the target is a GPS receiver whichreads very accurate time stamps transmitted by the satellites and calculatesdistances based on time offsets between its clock and satellites' clocks.

Given the sensor (X,Y,Z) coordinates and their reported distance-to-targetvalues, we want to determine the target location. The program uses either oftwo methods to locate the target object; "Gaussian Elimination" or"Trilateration" which is new with Version 5 of the program. In both methods, a set of distance equations is solved based finding the single intersection point of spheres centered on the sensors with radii equal to the sensor's (or the target's) estimate of the distance to (or from) the target.

Note that, in general, the problem requires 4 equations to narrow solutions to a single point. Two sensors can narrow target location down to a circle

Load, generate, or enter a case and click "Solve" to see the solution

	X:	Y:	Z:	Dist:
Sensor 1	1.000	1.000	1.000	6.928
Sensor 2	1.000	1.000	2.000	6.403
Sensor 3	2.000	1.000	1.000	6.403
Sensor 4	3.000	3.000	3.000	3.464
	5.000	5.000	5.000	0.000

Current Case:
Simple case 1 (unsaved)

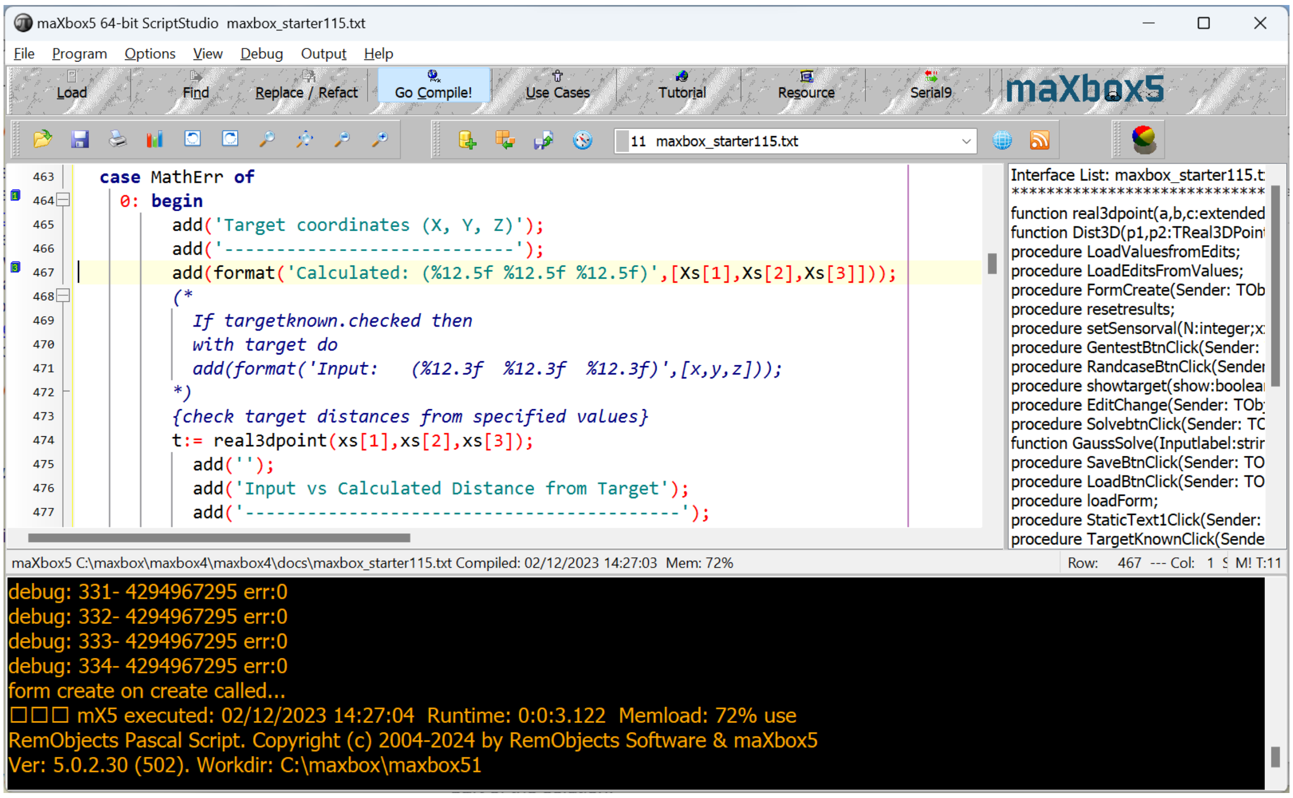
[✓] Solve Verbose (show more result detail)

```
Linear system:
                  Coefficients                    Constants
      0.000000      0.000000     -2.000000,     -9.998775
     -2.000000      0.000000      0.000000,     -9.998775
     -4.000000     -4.000000     -4.000000,    -59.997888

Target coordinates (X, Y, Z)
---------------------------
Calculated: (     4.99939      5.00070      4.99939)

Input vs Calculated Distance from Target
------------------------------------------
```

Copyright © 2008, 2014, 2020-2024 Gary Darby, maXbox5, www.DelphiForFun.org

Pic2: 115_GUIDesignandRuntime2023-12-02143419.png

The Solve button returns a position which may be at distances different from those in the original distance equations but do satisfy the distances relative to Sensor 1. That is part of what we sacrifice by eliminating one of the equations and converting quadratics to linear. A better solution in the ”GPS case might be to incorporate dummy 4th variable representing the distance error due to clock synchronization errors between the satellites and the GPS receiver. The multivariate Newton-Raphson algorithm is a possible way to solve by using 4 quadratic equations in 4 unknowns.


maXbox5 64-bit ScriptStudio maxbox_starter115.txt
File Program Options View Debug Output Help
Load Find Replace / Refact Go Compile! Use Cases Tutorial Resource Serial9
maXbox5
11 maxbox_starter115.txt

```
case MathErr of
  0: begin
     add('Target coordinates (X, Y, Z)');
     add('----------------------------');
     add(format('Calculated: (%12.5f %12.5f %12.5f)',[Xs[1],Xs[2],Xs[3]]));
     (*
       If targetknown.checked then
       with target do
       add(format('Input:   (%12.3f  %12.3f  %12.3f)',[x,y,z]));
     *)
     {check target distances from specified values}
     t:= real3dpoint(xs[1],xs[2],xs[3]);
       add('');
       add('Input vs Calculated Distance from Target');
       add('------------------------------------------');
```

```
Interface List: maxbox_starter115.t
****************************
function real3dpoint(a,b,c:extended
function Dist3D(p1,p2:TReal3DPoint
procedure LoadValuesfromEdits;
procedure LoadEditsFromValues;
procedure FormCreate(Sender: TOb
procedure resetresults;
procedure setSensorval(N:integer;x:
procedure GentestBtnClick(Sender:
procedure RandcaseBtnClick(Sender
procedure showtarget(show:boolea
procedure EditChange(Sender: TOb
procedure SolvebtnClick(Sender: TC
function GaussSolve(Inputlabel:strir
procedure SaveBtnClick(Sender: TO
procedure LoadBtnClick(Sender: TO
procedure loadForm;
procedure StaticText1Click(Sender:
procedure TargetKnownClick(Sende
```

```
maXbox5 C:\maxbox\maxbox4\maxbox4\docs\maxbox_starter115.txt Compiled: 02/12/2023 14:27:03  Mem: 72%     Row: 467 --- Col: 1 $ M! T:11
```

```
debug: 331- 4294967295 err:0
debug: 332- 4294967295 err:0
debug: 333- 4294967295 err:0
debug: 334- 4294967295 err:0
form create on create called...
☐☐☐ mX5 executed: 02/12/2023 14:27:04  Runtime: 0:0:3.122  Memload: 72% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
Ver: 5.0.2.30 (502). Workdir: C:\maxbox\maxbox51
```

Pic3: 115_Designtime2023-12-02143302.png

In Delphi or maXbox, I can include a folder's source code by adding it to the project Search Path or define as an include file, or adding it to the Library Path. The Search Path applies for the *UMatrix.pas* only to the current project, while the Library Path applies to any project opened with the IDE.

Version 5 Test Case

Version 5 posted today adds a second method for locating the target. Trilateration uses the locations of three sensors to exactly narrow the target location down to at most two possibilities. See this Wikipedia article for an excellent discussion and derivation of the Math involved. I translated a C code implementation to Delphi for testing.
Trilateration - Wikipedia

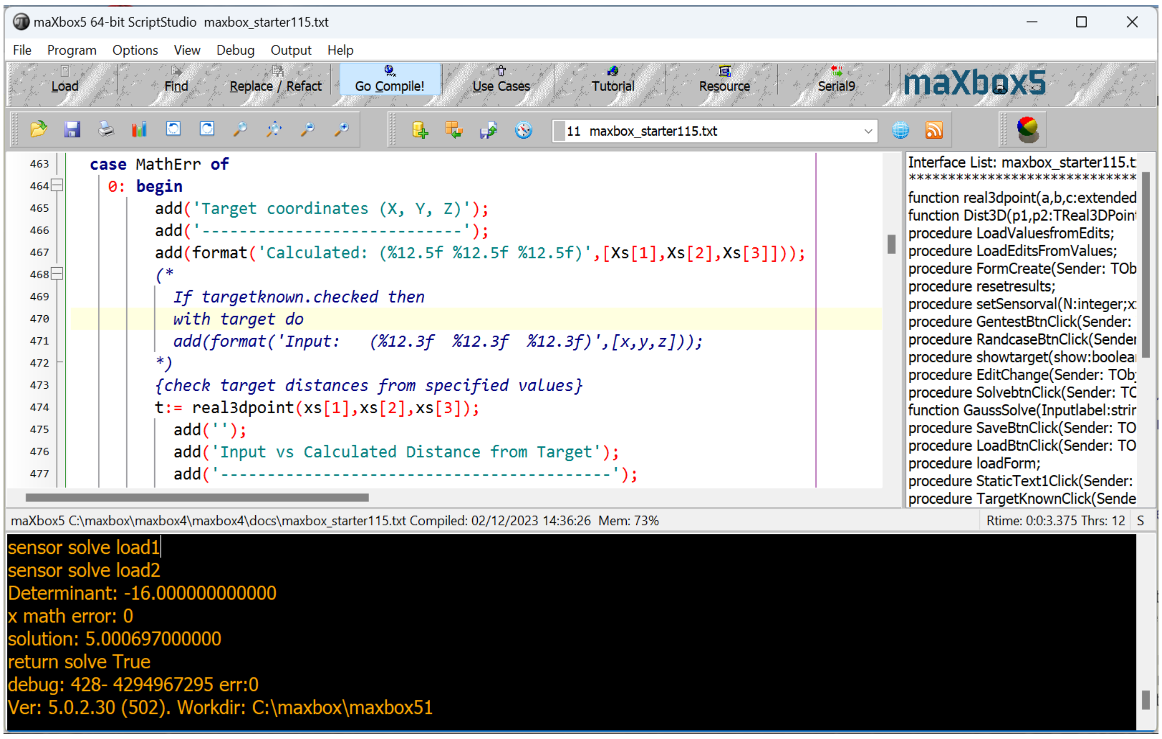
By solving the 4 combination using 3 of the 4 sensors and saving the two solutions from each case, it seems that we have good luck in identifying the single solution. It is much more robust to the Gaussian Elimination version, easily solving the case submitted by a user which led to the current investigation: Using notation (x, y, z, r) to represent the x, y, z, coordinates and r, the measured distance to the target.

A test case is defined by (0,0,0,10), (10,0,0,10), (0,10,0,10), and (10,10,0,10). Gaussian Elimination finds no solution or an incorrect solution if a small increment is added to the z coordinate of one of the sensors to remove the singularity. Trilateration correctly identifies the solution as (5, 5, 7.07) or (5, 5, -7.07) which is also valid:

```
Solution 1: 5.00 5.00 7.07
  Distance to sphere 1 is 10.000 (radius 10.000)
  Distance to sphere 2 is 10.000 (radius 10.000)
  Distance to sphere 3 is 10.000 (radius 10.000)
Solution 2: 5.00 5.00 -7.07
  Distance to sphere 1 is 10.000 (radius 10.000)
  Distance to sphere 2 is 10.000 (radius 10.000)
  Distance to sphere 3 is 10.000 (radius 10.000)

Solution 1: 5.00 5.00 7.07
  Distance to sphere 1 is 10.000 (radius 10.000)
  Distance to sphere 2 is 10.000 (radius 10.000)
  Distance to sphere 4 is 10.000 (radius 10.000)
Solution 2: 5.00 5.00 -7.07
  Distance to sphere 1 is 10.000 (radius 10.000)
  Distance to sphere 2 is 10.000 (radius 10.000)
  Distance to sphere 4 is 10.000 (radius 10.000)

Solution 1: 5.00 5.00 -7.07
  Distance to sphere 1 is 10.000 (radius 10.000)
  Distance to sphere 3 is 10.000 (radius 10.000)
  Distance to sphere 4 is 10.000 (radius 10.000)
Solution 2: 5.00 5.00 7.07
  Distance to sphere 1 is 10.000 (radius 10.000)
  Distance to sphere 3 is 10.000 (radius 10.000)
  Distance to sphere 4 is 10.000 (radius 10.000)

Solution 1: 5.00 5.00 -7.07
  Distance to sphere 2 is 10.000 (radius 10.000)
  Distance to sphere 3 is 10.000 (radius 10.000)
  Distance to sphere 4 is 10.000 (radius 10.000)
Solution 2: 5.00 5.00 7.07
  Distance to sphere 2 is 10.000 (radius 10.000)
  Distance to sphere 3 is 10.000 (radius 10.000)
  Distance to sphere 4 is 10.000 (radius 10.000)

Sum of coordinate differences:
  Solution 1: 28.28427, Solution #2: 28.28427
Use Solution 2 set
Solution 2: 5.00 5.00 -7.07
  Distance to sphere 1 is 10.00000 (vs. measured 10.00000)
  Distance to sphere 2 is 10.00000 (vs. measured 10.00000)
  Distance to sphere 3 is 10.00000 (vs. measured 10.00000)
  Distance to sphere 4 is 10.00000 (vs. measured 10.00000)
```

As you can see in Pic2 there are 19 *TEdit* controls for user input; 4 for each of the 4 sensors plus 3 if the user wants to enter target values. The target values were convenient when debugging the code with sets of points with known solutions. In order to simplify the code, I defined an array, Sensors, of *TSensorEdits* records, each containing 4 *TEdit* references (object references are always pointers), plus the numeric version of the X, Y, Z, and R (distance) values represent by the edits

for that specific sensor.
Perhaps as suggested we can just define references to avoid an assembler.
Geometrically, each of the 4 sensors and its target distance define a sphere upon which the target must lie, and that should be the common point of intersection of all 4 spheres.


maXbox5 64-bit ScriptStudio maxbox_starter115.txt
File Program Options View Debug Output Help
Load Find Replace / Refact Go Compile! Use Cases Tutorial Resource Serial9
11 maxbox_starter115.txt

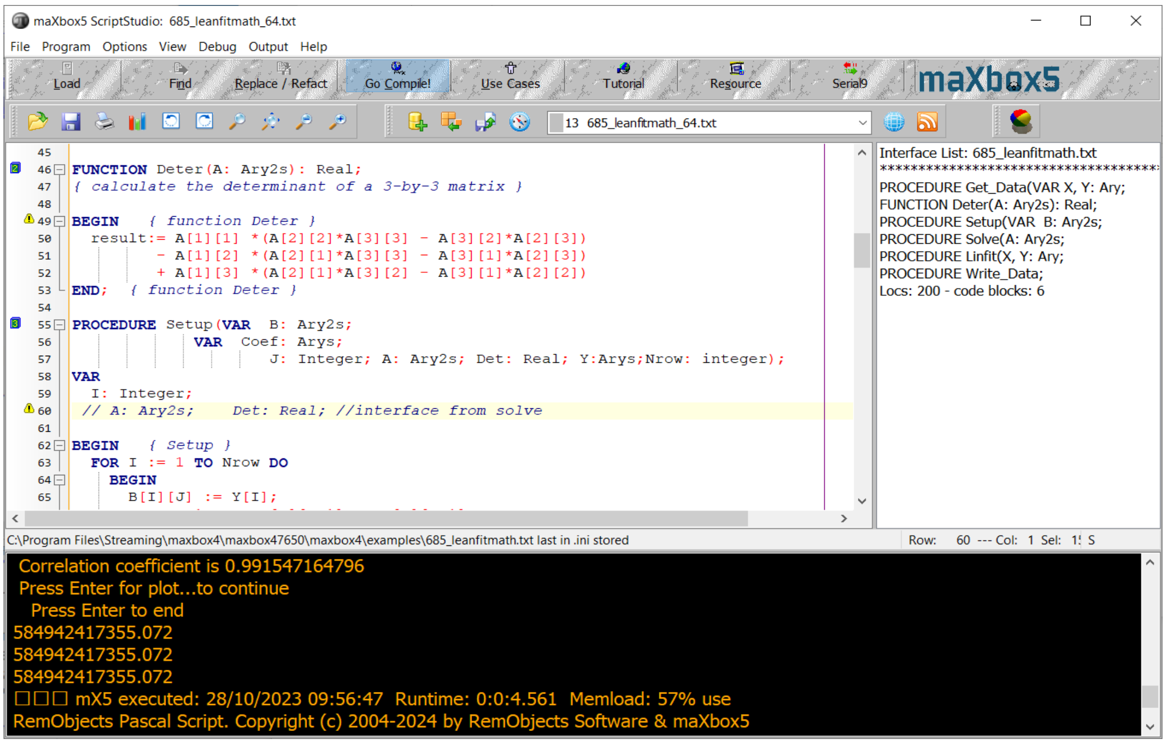
```
case MathErr of
  0: begin
       add('Target coordinates (X, Y, Z)');
       add('--------------------------');
       add(format('Calculated: (%12.5f %12.5f %12.5f)',[Xs[1],Xs[2],Xs[3]]));
       (*
         If targetknown.checked then
         with target do
         add(format('Input:    (%12.3f  %12.3f  %12.3f)',[x,y,z]));
       *)
       {check target distances from specified values}
       t:= real3dpoint(xs[1],xs[2],xs[3]);
         add('');
         add('Input vs Calculated Distance from Target');
         add('-----------------------------------------');
```

```
Interface List: maxbox_starter115.t
****************************************
function real3dpoint(a,b,c:extended
function Dist3D(p1,p2:TReal3DPoin
procedure LoadValuesfromEdits;
procedure LoadEditsFromValues;
procedure FormCreate(Sender: TOb
procedure resetresults;
procedure setSensorval(N:integer;x
procedure GentestBtnClick(Sender:
procedure RandcaseBtnClick(Sende
procedure showtarget(show:boolea
procedure EditChange(Sender: TOb
procedure SolvebtnClick(Sender: TC
function GaussSolve(Inputlabel:strir
procedure SaveBtnClick(Sender: TO
procedure LoadBtnClick(Sender: TO
procedure loadForm;
procedure StaticText1Click(Sender:
procedure TargetKnownClick(Sende
```

```
maXbox5 C:\maxbox\maxbox4\maxbox4\docs\maxbox_starter115.txt Compiled: 02/12/2023 14:36:26  Mem: 73%    Rtime: 0:0:3.375 Thrs: 12  S
```

```
sensor solve load1
sensor solve load2
Determinant: -16.000000000000
x math error: 0
solution: 5.000697000000
return solve True
debug: 428- 4294967295 err:0
Ver: 5.0.2.30 (502). Workdir: C:\maxbox\maxbox51
```

Pic4: 115_GUIDesignandRuntime2023-12-02143419.png


maXbox5 ScriptStudio: 685_leanfitmath_64.txt
File Program Options View Debug Output Help
Load Find Replace / Refact Go Compile! Use Cases Tutorial Resource Serial9
13 685_leanfitmath_64.txt

```

FUNCTION Deter(A: Ary2s): Real;
{ calculate the determinant of a 3-by-3 matrix }

BEGIN   { function Deter }
  result:= A[1][1] *(A[2][2]*A[3][3] - A[3][2]*A[2][3])
         - A[1][2] *(A[2][1]*A[3][3] - A[3][1]*A[2][3])
         + A[1][3] *(A[2][1]*A[3][2] - A[3][1]*A[2][2])
END;  { function Deter }

PROCEDURE Setup(VAR  B: Ary2s;
                VAR  Coef: Arys;
                     J: Integer; A: Ary2s; Det: Real; Y:Arys;Nrow: integer);
VAR
  I: Integer;
 // A: Ary2s;    Det: Real; //interface from solve

BEGIN   { Setup }
  FOR I := 1 TO Nrow DO
    BEGIN
      B[I][J]  := Y[I];
```

```
Interface List: 685_leanfitmath.txt
************************************
PROCEDURE Get_Data(VAR X, Y: Ary;
FUNCTION Deter(A: Ary2s): Real;
PROCEDURE Setup(VAR  B: Ary2s;
PROCEDURE Solve(A: Ary2s;
PROCEDURE Linfit(X, Y: Ary;
PROCEDURE Write_Data;
Locs: 200 - code blocks: 6
```

```
C:\Program Files\Streaming\maxbox4\maxbox47650\maxbox4\examples\685_leanfitmath.txt last in .ini stored    Row:  60 --- Col:  1 Sel:  1: S
```

```
 Correlation coefficient is 0.991547164796
 Press Enter for plot...to continue
  Press Enter to end
584942417355.072
584942417355.072
584942417355.072
□□□ mX5 executed: 28/10/2023 09:56:47  Runtime: 0:0:4.561  Memload: 57% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
```

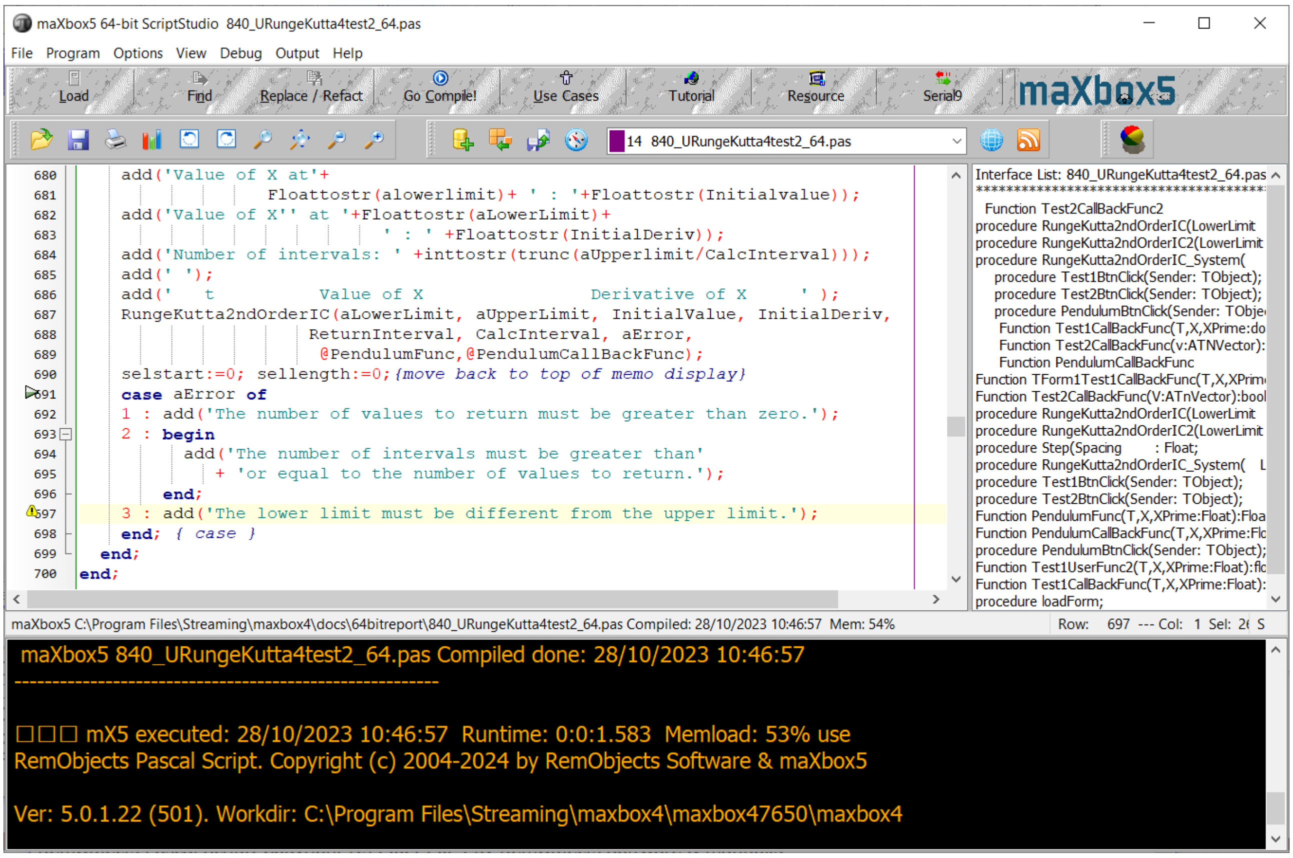
Pic5: 7_mX5_64bitGUI.png

This target coordinate technique above used to help with the understanding about what each distance represents and what the difference results of. Alternatively, the sensors are Satellites and the target is a GPS receiver which reads very accurate time stamps transmitted by the satellites and calculates distances based on time offsets between its clock and satellites' clocks. About the topic:

How GPS Receivers Work - Trilateration vs Triangulation - GIS Geography

Another interesting topic is the Runge-Kutta technique for solving second order ordinary differential equations, like in a pendulum motion.

Pendulums, Simple and otherwsie (delphiforfun.org)


maXbox5 64-bit ScriptStudio 840_URungeKutta4test2_64.pas
File Program Options View Debug Output Help
Load
Find
Replace / Refact
Go Compile!
Use Cases
Tutorial
Resource
Serial9
maXbox5
14 840_URungeKutta4test2_64.pas

```
    add('Value of X at'+
              Floattostr(alowerlimit)+ ' : '+Floattostr(Initialvalue));
    add('Value of X'' at '+Floattostr(aLowerLimit)+
                        ' : ' +Floattostr(InitialDeriv));
    add('Number of intervals: ' +inttostr(trunc(aUpperlimit/CalcInterval)));
    add(' ');
    add('   t          Value of X                Derivative of X     ' );
    RungeKutta2ndOrderIC(aLowerLimit, aUpperLimit, InitialValue, InitialDeriv,
                  ReturnInterval, CalcInterval, aError,
                   @PendulumFunc,@PendulumCallBackFunc);
    selstart:=0; sellength:=0;{move back to top of memo display}
    case aError of
    1 : add('The number of values to return must be greater than zero.');
    2 : begin
          add('The number of intervals must be greater than'
            + 'or equal to the number of values to return.');
        end;
    3 : add('The lower limit must be different from the upper limit.');
    end; { case }
  end;
end;
```

```
Interface List: 840_URungeKutta4test2_64.pas
****************************************
  Function Test2CallBackFunc2
procedure RungeKutta2ndOrderIC(LowerLimit
procedure RungeKutta2ndOrderIC2(LowerLimit
procedure RungeKutta2ndOrderIC_System(
    procedure Test1BtnClick(Sender: TObject);
    procedure Test2BtnClick(Sender: TObject);
    procedure PendulumBtnClick(Sender: TObje
     Function Test1CallBackFunc(T,X,XPrime:do
     Function Test2CallBackFunc(v:ATNVector):
     Function PendulumCallBackFunc
Function TForm1Test1CallBackFunc(T,X,XPrim
Function Test2CallBackFunc(V:ATnVector):bool
procedure RungeKutta2ndOrderIC(LowerLimit
procedure RungeKutta2ndOrderIC2(LowerLimit
procedure Step(Spacing       : Float;
procedure RungeKutta2ndOrderIC_System(   L
procedure Test1BtnClick(Sender: TObject);
procedure Test2BtnClick(Sender: TObject);
Function PendulumFunc(T,X,XPrime:Float):Floa
Function PendulumCallBackFunc(T,X,XPrime:Flo
procedure PendulumBtnClick(Sender: TObject);
Function Test1UserFunc2(T,X,XPrime:Float):flo
Function Test1CallBackFunc(T,X,XPrime:Float):
procedure loadForm;
```

```
maXbox5 C:\Program Files\Streaming\maxbox4\docs\64bitreport\840_URungeKutta4test2_64.pas Compiled: 28/10/2023 10:46:57  Mem: 54%     Row:  697 --- Col:  1  Sel: 2( S
```

```
maXbox5 840_URungeKutta4test2_64.pas Compiled done: 28/10/2023 10:46:57
------------------------------------------------------

□□□ mX5 executed: 28/10/2023 10:46:57  Runtime: 0:0:1.583  Memload: 53% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5

Ver: 5.0.1.22 (501). Workdir: C:\Program Files\Streaming\maxbox4\maxbox47650\maxbox4
```

Pic6: 8_mX5_64bitGUI2.png

Conclusion:

Triangulation is process of measuring bearings and calculating distances (using the Sine Rule).
Trilateration is the process of measuring distances and calculating bearings (using Cosine Rule).
Geometrically, each of the 4 sensors and its target distance define a sphere upon which the target must lie, and that should be the common point of intersection of all 4 sphere.
64-bit isn’t a magic bullet. Other than that, you only need to change if you’ve already encountered one of the limits imposed by 32-bit or you want to develop plug-ins for 64-bit app or just be compatible with a 64-bit operation system.
Script: softwareschule.ch/examples/maxbox_starter115.txt

References:

Compiled Project:
https://github.com/maxkleiner/maXbox4/releases/download/V4.2.4.80/maxbox5.zip

Topic:
http://delphiforfun.org/Programs/Math_Topics/PointFrom4Sensors.htm
https://gis.stackexchange.com/questions/17344/differences-between-triangulation-and-trilateration

Preparation:
https://stackoverflow.com/questions/4051603/how-should-i-prepare-my-32-bit-delphi-programs-for-an-eventual-64-bit-compiler

Doc and Tool: https://maxbox4.wordpress.com

Appendix

```
Unit UMatrix
Procedure Determinant( Dimen : integer; Data : TNmatrix; var Det : Float; var Error : byte)');
Procedure Inverse2( Dimen : integer; Data : TNmatrix; var Inv : TNmatrix; var Error : byte)');
 Procedure Gaussian_Elimination(Dimen:integer;Coefficients:TNmatrix;Constants:TNvector;var Solution:TNvector;var Error:byte);
 Procedure Partial_Pivoting(Dimen:integer; Coefficients:TNmatrix;Constants:TNvector; var Solution:TNvector;var Error:byte);
 Procedure LU_Decompose(Dimen:integer;Coefficients:TNmatrix;var Decomp:TNmatrix;var Permute:TNmatrix;var Error:byte);
 Procedure LU_Solve(Dimen: integer;var Decomp TNmatrix;Constants:TNvector;var Permute:TNmatrix;var Solution:TNvector;var Error:byte);
 Procedure Gauss_Seidel( Dimen : integer; Coefficients : TNmatrix; Constants : TNvector; Tol : Float; MaxIter : integer; var Solution : TNvector; var Iter : integer; var Error : byte)');
end;

 CL.AddTypeS('TNvector', 'array[1..30] of Extended');
 //TNvector = array[1..TNArraySize] of Float;
 CL.AddTypeS('TNmatrix', 'array[1..30] of TNvector');
 //TNmatrix = array[1..TNArraySize] of TNvector;

 CL.AddConstantN('TNNearlyZero','Extended').setExtended( 1E-07);
 CL.AddConstantN('TNArraySize','LongInt').SetInt( 30);
 CL.AddDelphiFunction('Procedure Gaussian_Elimination( Dimen : integer; Coefficients : TNmatrix; Constants : TNvector; var Solution : TNvector; var Error : byte)');
 CL.AddDelphiFunction('Procedure Partial_Pivoting( Dimen : integer; Coefficients : TNmatrix; Constants : TNvector; var Solution : TNvector; var Error : byte)');
 CL.AddDelphiFunction('Procedure LU_Decompose( Dimen : integer; Coefficients : TNmatrix; var Decomp : TNmatrix; var Permute : TNmatrix; var Error : byte)');
 CL.AddDelphiFunction('Procedure LU_Solve2( Dimen : integer; var Decomp : TNmatrix; Constants : TNvector; var Permute : TNmatrix; var Solution : TNvector; var Error : byte)');
 CL.AddDelphiFunction('Procedure Gauss_Seidel( Dimen : integer; Coefficients : TNmatrix; Constants : TNvector; Tol : Float; MaxIter : integer; var Solution : TNvector; var Iter : integer; var Error : byte)');
end;

Solution 1: 5.00 5.00 5.00
  Distance to sphere 1 is 6.928 (radius 6.928)
  Distance to sphere 2 is 6.403 (radius 6.403)
  Distance to sphere 3 is 6.403 (radius 6.403)
Solution 2: 5.00 -3.00 5.00
  Distance to sphere 1 is 6.928 (radius 6.928)
  Distance to sphere 2 is 6.403 (radius 6.403)
  Distance to sphere 3 is 6.403 (radius 6.403)

Sum of coordinate differences:
  Solution 1: 1.75172, Solution #2: 24.05681
Using Solution 1 set
Solution 1: 5.00 5.00 5.00
  Distance to sphere 1 is 6.92800 (vs. measured 6.92800)
  Distance to sphere 2 is 6.40300 (vs. measured 6.40300)
  Distance to sphere 3 is 6.40300 (vs. measured 6.40300)
  Distance to sphere 4 is 3.46390 (vs. measured 3.46400)
//}
```

Max Kleiner 02/12/2023

maXbox4exe Digitally signed by maXbox4exe
Date: 2023.12.02 17:06:00
+01'00'