Best-fit-plane through a point cloud

by Vince Cronin, *after* Paláncz and others (2013)

Revised 24 November 2017, based on earlier working drafts.

Input

Start with a clean slate by clearing the input variables

In[1]:=
```
Clear[XYZ, N]
```

The file "**Composite-Loc-Data-V4**" contains all of the aftershock locations published by Greensfelder (1968) and Ryall et al. (1968) with the following exceptions : Greensfelder's events 32, 70, and 91, because their respective epicenters were outside the geographic box lat 39.370 to 39.544, long -120.211 to -120.037. In cases where the same event was reported by both sources, the mean of the locations was used in preference to double-reporting the event or arbitrarily choosing a preferred source.

In[2]:=

	A	B	C	D
1	39.4054	-120.1587	8.5	C01
2	39.4002	-120.1783	6.1	C02
3	39.3846	-120.177	5	C03
4	39.3972	-120.1708	2.2	C04

;

The data is in an Excel spreadsheet in which column A is the latitude and column B is the longitude -- both in decimal degrees where negative latitude is south and negative longitude is west. Column C is the focal depth in kilometers. Column D is an event ID.

Import the Point Cloud Data File

To modify this code so that it can be used for your dataset, do the following.

1. Enter the following as an input line (under the black box, below):

```
pointCloudDataIn = Import[];
```

2. Put your cursor between the square brackets in the **Import[]** statement and click to establish the insertion point we will need in the next step.

3. Go to the **Insert** menu, select **File Path**, navigate to the correct input data file and choose it, and the correct file path will be inserted at the cursor. In this example, the Excel file is located on the desktop of Vince's computer, and so *Mathematica* will insert the path "**/Users/vince/Desktop/Composite-Loc-Data-V4.xls**" between the brackets. The path will be different for every different file, and on every different computer.

```
pointCloudDataIn = Import[“/Users/vince/Desktop/Composite-Loc-Data-V4.xls”];
```

You can make sure it is an input line by clicking on that line's bracket on the far right edge of this window, going to the **Format** menu, selecting the **Style** submenu, and then choosing **Input.**

In[3]:=
```
pointCloudDataIn =
  Import["/Users/vincecronin/Desktop/Composite-Loc-Data-V4.xls"];
```

This won' t work unless you follow directions and add the appropriate input line above this line. Your input line should look something like the example:
```
pointCloudDataIn = Import[“/Users/vincecronin/Desktop/Composite-Loc-Data-V4.xls”];
```

The imported dataset will have to be modified so that it has the correct dimensions for this analysis, which is accomplished with the following code that defines the list **pointCloudData**:

In[4]:=
```
pointCloudData = Flatten[pointCloudDataIn, 1];
```

In[5]:=
```
noRecords = Length[pointCloudData]
```

Out[5]= 187

The origin of the coordinate system used in this analysis is just beyond the south and west corner of the dataset. That is, the coordinate system origin latitude (**csOriginLat**) is smaller than the minimum latitude of any point in the point cloud, and the coordinate system origin longitude (**csOriginLong**) is smaller than the minimum longitude of any point in the point cloud.

Variable **minLat** is the minimum latitude value (associated with the farthest south point) in the point cloud.

In[6]:=
```
minLat = Min[Table[pointCloudData[[i, 1]], {i, 1, noRecords}]]
```

Out[6]= 39.3696

Variable **minLong** is the maximum latitude value (associated with the farthest west point) in the point cloud.

In[7]:=
```
minLong = Min[Table[pointCloudData[[i, 2]], {i, 1, noRecords}]]
```
Out[7]=
```
-120.201
```

In[8]:=
```
csOriginLat = 39.36;
```

In[9]:=
```
csOriginLong = -120.22;
```

User-Defined Functions

In[10]:=
```
unitVect3D[vect_] :=
  {(vect[[1]] / Norm[vect]), (vect[[2]] / Norm[vect]), (vect[[3]] / Norm[vect])};
```

The function **PlanePCA** is adapted from Appendix A - 1 of Paláncz and others (2013)

In[11]:=
```
PlanePCA[XYZ_] := Module[{N, co, Cco, nx, ny, nz, d, sold},
  N = Length[XYZ];
  co = Mean[XYZ];
  Cco = Map[# - co &, XYZ];
  {nx, ny, nz} = Eigenvectors[Covariance[Cco]][[3]];
  sold = Solve [{nx, ny, nz}.Total[XYZ] - d N == 0, d] // Flatten;
  {-nx/nz, -ny/nz, d/nz} /. sold];
```

Computation

Variable **maxLat** is the maximum latitude value (associated with the farthest north point) in the point cloud.

In[12]:=
```
maxLat = Max[Table[pointCloudData[[i, 1]], {i, 1, noRecords}]]
```
Out[12]=
```
39.4894
```

Variable **midLat** is the latitude that is half-way between **maxLat** and **minLat**.

In[13]:=
```
midLat = (maxLat + minLat) / 2
```
Out[13]=
```
39.4295
```

Variable **maxLong** is the maximum latitude value (associated with the farthest east point) in the point cloud.

In[14]:=
```
maxLong = Max[Table[pointCloudData[[i, 2]], {i, 1, noRecords}]]
```
Out[14]=
```
-120.104
```

Variable **midLong** is the latitude that is half-way between **maxLong** and **minLong**.

In[15]:=
```
midLong = (maxLong + minLong) / 2
```
Out[15]= −120.153

Variable **deg2kmLat** is the number of kilometers per degree of latitude, assuming a mean Earth radius of 6371 km.

In[16]:=
```
deg2kmLat = (6371 * 2 * π) / 360;
```

In[17]:=
```
N[%]
```
Out[17]= 111.195

Variable **deg2kmLongMidLat** is the number of kilometers per degree of longitude, computed at the middle latitude of the point cloud.

In[18]:=
```
deg2kmLongMidLat = deg2kmLat * Cos[midLat Degree]
```
Out[18]= 85.8877

The vectors computed in the following expression are in km units. The origin {0,0,0} is at {csOriginLong, csOriginLat, mean sea level}, where the difference between **csOrginLat** and eqData[[i,1]] is multiplied by **deg2kmLat** or around 111.195 km/degree to compute the y coordinate for any given vector, and the difference between **csOriginLong** and eqData[[i,2]] is multiplied by **deg2kmLongMidLat** or around 85.89 km/degree to compute the x coordinate for any given vector. The result is a rectangular horizontal mapping of the epicenters. The sign of the input focal depths is reversed so that negative depths are below sea level.

In[19]:=
```
XYZ = Table[{(((pointCloudData[[i, 2]] - csOriginLong) * deg2kmLongMidLat)),
    ((pointCloudData[[i, 1]] - csOriginLat) * deg2kmLat),
    (-1) * pointCloudData[[i, 3]]}, {i, 1, noRecords}];
```

In[20]:=
```
maxX = Max[Table[XYZ[[i, 1]], {i, 1, noRecords}]]
```
Out[20]= 9.9458

In[21]:=
```
minX = Min[Table[XYZ[[i, 1]], {i, 1, noRecords}]]
```
Out[21]= 1.64904

In[22]:=
```
midX = (maxX + minX) / 2
```
Out[22]= 5.79742

In[23]:=
```
maxY = Max[Table[XYZ[[i, 2]], {i, 1, noRecords}]]
```
Out[23]= 14.3886

In[24]:=
```
minY = Min[Table[XYZ[[i, 2]], {i, 1, noRecords}]]
```
Out[24]= 1.06747

In[25]:=
```
midY = (maxY + minY) / 2
```
Out[25]= 7.72805

In[26]:=
```
maxZ = Max[Table[XYZ[[i, 3]], {i, 1, noRecords}]]
```
Out[26]= 1.7

In[27]:=
```
minZ = Min[Table[XYZ[[i, 3]], {i, 1, noRecords}]]
```
Out[27]= -14.4

In[28]:=
```
ℕ = Length[XYZ]
```
Out[28]= 187

In[29]:=
```
north = {0, 1, 0};
```

Where is the Centroid of the Point Cloud?

In[30]:=
```
meanVector = Mean[XYZ]
```
Out[30]= {4.80438, 5.69318, -4.03877}

In[31]:=
```
latitude = csOriginLat + (meanVector[[2]] / deg2kmLat)
```
Out[31]= 39.4112

In[32]:=
```
longitude = csOriginLong + (meanVector[[1]] / deg2kmLongMidLat)
```
Out[32]= -120.164

In[33]:=
```
depth = meanVector[[3]]
```
Out[33]= -4.03877

Check

In[34]:=
```
meanVector
```
Out[34]= {4.80438, 5.69318, -4.03877}

In[35]:=
```
recomputedMeanVector = {((longitude - csOriginLong) * deg2kmLongMidLat),
   ((latitude - csOriginLat) * deg2kmLat), depth}
```
Out[35]= {4.80438, 5.69318, -4.03877}

End Check

Solution via SVD

after Paláncz and others (2013)

In[36]:=
```
c0 = Mean[XYZ]
```
Out[36]= {4.80438, 5.69318, -4.03877}

In[37]:=
```
Cc0 = Map[# - c0 &, XYZ];
```

In[38]:=
```
AbsoluteTiming[{u, w, v} = SingularValueDecomposition[Cc0];]
```
Out[38]=
```
{0.004301, Null}
```

In[39]:=
```
v = Last[Transpose[v]]
```
Out[39]=
```
{-0.849291, 0.527174, -0.0281583}
```

In[40]:=
```
({x, y, z} - c0).v == 0 // Simplify
```
Out[40]=
```
1. x + 0.0331551 z == 1.13659 + 0.620723 y
```

In[41]:=
```
NSolve[%, z] // Expand
```
Out[41]=
```
{{z → 34.281 - 30.1612 x + 18.7218 y}}
```

In[42]:=
```
p1 = Plot3D[z /. %[[1]], {x, minX, maxX}, {y, minY, maxY}, BoxRatios → {1, 1, 0.5},
    ClippingStyle → None, Mesh → None, PlotRange → {minZ, maxZ}];
```

The equation of a plane can be written as z = γ - αx + βy, so the result above provides the value of α, β, and γ.

Solution via PCA

after Paláncz and others (2013)

In[43]:=
```
solnPCA = PlanePCA[XYZ]
```
Out[43]=
```
{-30.1612, 18.7218, 34.281}
```

The equation of a plane can be written as z = γ - αx + βy, so the result above provides the value of α, β, and γ.

In[44]:=
```
α = solnPCA[[1]]
```
Out[44]=
```
-30.1612
```

In[45]:=
```
β = solnPCA[[2]]
```
Out[45]=
```
18.7218
```

In[46]:=
```
γ = solnPCA[[3]]
```
Out[46]=
```
34.281
```

The value of **meanX** is equal to (γ + (β*meanY) - meanZ)/(-α).

The value of **meanY** is equal to (γ + (α*meanX) - meanZ)/(-β).

The value of **meanZ** is equal to γ + (α*meanX) + (β*meanY).

Find the Vectors Normal to the Plane

We already know the coordinates of one point on the plane -- the mean point in the XYZ location - vector dataset (**meanVector**). We need to find the coordinates of two other points on the plane in

order to find the vectors normal to the plane, which will allow us to find the strike, dip, and dip angle of the plane.

If we know the x and z coordinates of a point on the plane, we can use the equation of the plane to calculate the y coordinate. We use the z coordinate of the mean vector (**meanZ**), select an x coordinate that is between **xMax** and **xMin** (**midX**), and compute the corresponding y value (**trialY**). The result is another point that is known to be on the plane (**point1**) and is different from the **meanVector**.

In[47]:=
```
meanZ = meanVector[[3]];
```

In[48]:=
```
trialY = (γ + (α * midX) - meanZ) / (-β)
```

Out[48]=
```
7.29299
```

In[49]:=
```
point1 = {midX, trialY, meanZ}
```

Out[49]=
```
{5.79742, 7.29299, -4.03877}
```

This time, we use **midX** as defined above, select a z coordinate (**lowZ**) that is below the z coordinate of the mean vector (**meanZ**), and compute the corresponding y value (**lowY**). The result is another point that is known to be on the plane (**point2**) and is different from the **meanVector**.

In[50]:=
```
lowZ = meanZ - (0.5 (meanZ - minZ))
```

Out[50]=
```
-9.21939
```

In[51]:=
```
lowY = (γ + (α * midX) - lowZ) / (-β)
```

Out[51]=
```
7.01627
```

In[52]:=
```
point2 = {midX, lowY, lowZ}
```

Out[52]=
```
{5.79742, 7.01627, -9.21939}
```

We now know the location vectors to three points that are on the plane -- **point1**, **point2**, and **meanVector**. From these location vectors, we can determine two vectors parallel to the plane -- **vectorOnPlane1** and **vectorOnPlane2**.

In[53]:=
```
vectorOnPlane1 = point1 - meanVector
```

Out[53]=
```
{0.993036, 1.59981, 0.}
```

In[54]:=
```
vectorOnPlane2 = point2 - meanVector
```

Out[54]=
```
{0.993036, 1.32309, -5.18061}
```

The vectors **vectorOnPlane1** and **vectorOnPlane2** are parallel to the plane but not to each other, so we can use their cross product and the user-defined function **unitVect3D** to determine unit vectors normal to the plane.

In[55]:=
```
normal1 = unitVect3D[Cross[vectorOnPlane1, vectorOnPlane2]]
```

Out[55]=
```
{-0.849291, 0.527174, -0.0281583}
```

In[56]:=
```
normal2 = -normal1
```
Out[56]=
```
{0.849291, -0.527174, 0.0281583}
```

In[57]:=
```
normal = If[normal1[[3]] ≥ 0, normal1, normal2]
```
Out[57]=
```
{0.849291, -0.527174, 0.0281583}
```

The dip angle of the best-fit plane (**dipAngle**) is measured in degrees and is the same as the angle between a vector pointing up along the vertical (Z) axis (**vertUpVect**) and the vector normal to the plane (**normal**) that has a positive or zero Z coordinate.

In[58]:=
```
vertUpVect = {0, 0, 1};
```

In[59]:=
```
dipAngle = VectorAngle[vertUpVect, normal] * (180 / π)
```
Out[59]=
```
88.3864
```

The trend of the dip vector (**dipDirection**) is also given in degrees of azimuth, measured clockwise relative to north. To find the dip direction, we start by projecting the normal vector onto a horizontal plane (**projectedVector**) and then finding the unit vector of the resulting projection (**unitProjVector**).

In[60]:=
```
projectedVector = {normal[[1]], normal[[2]], 0}
```
Out[60]=
```
{0.849291, -0.527174, 0}
```

In[61]:=
```
unitProjVector = unitVect3D[projectedVector]
```
Out[61]=
```
{0.849628, -0.527383, 0.}
```

In[62]:=
```
dipDirection =
 If[unitProjVector[[1]] ≥ 0, VectorAngle[north, unitProjVector] * (180 / π),
  360 - (VectorAngle[north, unitProjVector] * (180 / π))]
```
Out[62]=
```
121.829
```

In[63]:=
```
direction = If[dipDirection ≤ 15, "north",
  If[dipDirection ≤ 75, "northeast", If[dipDirection ≤ 105, "east",
    If[dipDirection ≤ 165, "southeast", If[dipDirection ≤ 195, "south",
      If[dipDirection ≤ 255, "southwest", If[dipDirection ≤ 285,
        "west", If[dipDirection ≤ 345, "northwest", "north"]]]]]]]]
```
Out[63]=
```
southeast
```

The right-hand-rule strike azimuth (**rhrStrike**) is 90° less than the dip direction.

In[64]:=
```
rhrStrike = If[dipDirection < 90, (270 + dipDirection), (dipDirection - 90)]
```
Out[64]=
```
31.8288
```

Check

In[65]:=
```
strikeVector = {Sin[rhrStrike Degree], Cos[rhrStrike Degree], 0}
```

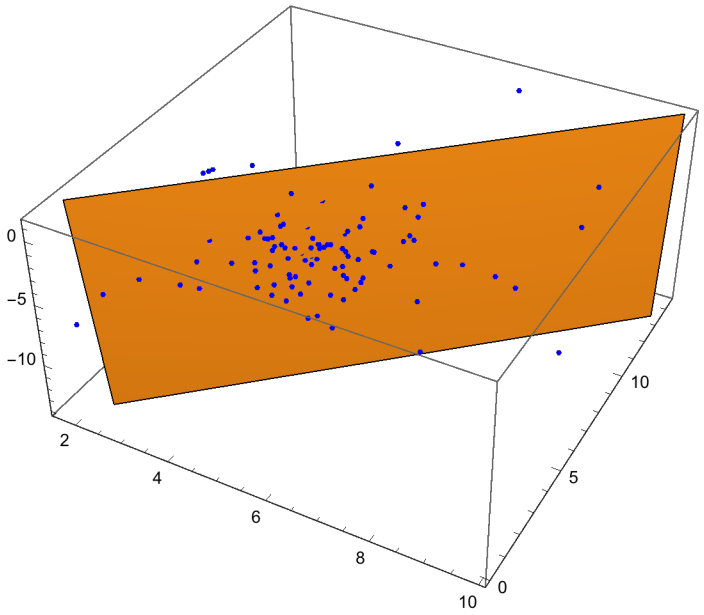
Out[65]=
```
{0.527383, 0.849628, 0}
```

Graphics

In[66]:=
```
p2 = ListPointPlot3D[XYZ, PlotStyle → Blue ];
```

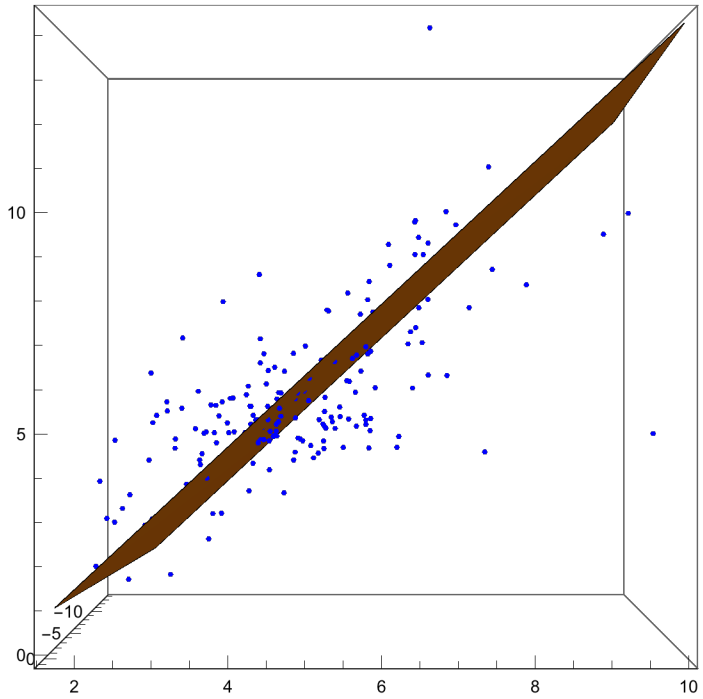
In[67]:=
```
Show[{p1, p2}]
```

Out[67]=



In[68]:=
```
Show[{p1, p2}, ViewPoint → Above]
```

Out[68]=



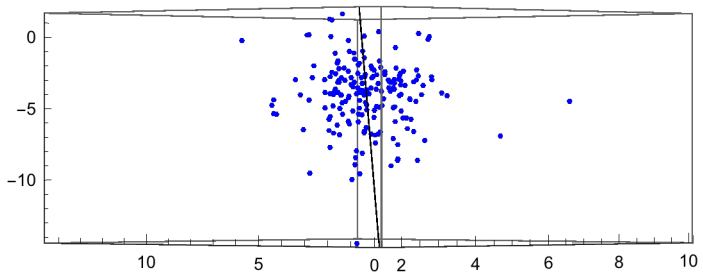
To use the **ViewVector** option, we need to specify the location vector to a point from which we will view the aftershock foci and best-fit plane (**viewOrigin**) and the vector of the point towards which we are looking (**meanVector**). The **viewOrigin** is located a given distance (**centroid2ViewOrigin**) from the centroid of the point cloud, along strike of the best-fit plane in the opposite direction from the **rhrStrike**. So the final view direction is toward the rhrStrike, and the point cloud is projected onto a plane that is perpendicular to the best-fit plane.

In[69]:=
```
centroid2ViewOrigin = 10 * √(maxX^2 + maxY^2) ;
```

In[70]:=
```
viewOrigin = {(-centroid2ViewOrigin * strikeVector[[1]]),
    (-centroid2ViewOrigin * strikeVector[[2]]), meanVector[[3]]} + meanVector;
```

In[71]:=
```
Show[{p1, p2}, ViewVector → {viewOrigin, strikeVector}]
```

Out[71]=



Summary

The best-fit plane through the input list of aftershock focal locations has a right-hand-rule strike

of

In[72]:= **rhrStrike**

Out[72]= 31.8288

degrees and a dip angle of

In[73]:= **dipAngle**

Out[73]= 88.3864

degrees toward

In[74]:= **dipDirection**

Out[74]= 121.829

degrees (dip azimuth), or toward the

In[75]:= **direction**

Out[75]= southeast

That plane passes through a point at latitude

In[76]:= **latitude**

Out[76]= 39.4112

degrees, longitude

In[77]:= **longitude**

Out[77]= −120.164

degrees, at a depth of

In[78]:= **depth**

Out[78]= −4.03877

km (negative depth value indicates an elevation below sea level).

References

Chen C C and Stamos I (2007) Range image segmentation for modeling and object detection in urban scenes, 3 DIM2007 .1

DalleMole V, do Rego R and Araújo A (2010) The Self - Organizing Approach for Surface Reconstruction from Unstructured Point Clouds, in Matsopoulos G.K.(Ed.) Self - Organizing Map, In Tech,

Diebel J R, Thrun S and Brunig M (2006) A Bayesian method for probable surface reconstruction and decimation, ACM Transactions on Graphics (TOG). 1

Fischler M A and Bolles R C (1981) Random sample consensus : A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24, pp

.381 - 395

Huang C - M and Tseng Y - H. Plane fitting methods of Lidar point cloud, Dept. of Geomatics, National Cheng Kung Uni. Taiwan, tseng@mail.ncku.edu.tw

Hubert M , Rousseeuw P J and Van den Branden K (2005) ROBPCA : A new approach to robust principal component analysis, TECHNOMETRICS, Vol. 47. No. 1. pp .64 - 79.

Kohonen T (1998) The self - organizing map, Neurocomputing, Vol. 21, pp .1 - 6.

Lakaemper R and Latecki L J (2006) Extended EM for Planar Approximation of 3 D Data, IEEE Int. Conf. on Robotics and Automation (ICRa), Orlando, Florida, May 2006.

Lukács G, Martin R and Marshall D (1998) Faithful Leat - Squares Fitting of Spheres, cylinders, Cones and Tori for Reliable Segmentation, in Burkhardt H and Neumann B (Eds.) Computer Vision, ECCV ' 98, Vol I, LNCS 1406, pp .671 - 686, Springer - Verlag Berlin Heidelberg.

Mitra N J and Nguyen (2003) SoCG' 03, June 8 - 10, San Diego, California, USA, ACM 1 - 58113 - 663 - 3/03/0006, pp. 322 - 328.

Nievergelt Y (2000) A tutorial history of least square with applications to astronomy and geodesy, J. Comp. and Appl. Mathematics, 121, pp. 37 - 72.

Nurunnabi A, Belton D and West G. (2012) Diagnostic - Robust Statistical Analysis for Local Surface Fitting in 3 D Point Cloud Data, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol.I - 3, 2012 XXII ISPRS Congress, 25 Aug. 2012, Melbourne, Australia, pp .269 - 275

Paláncz B (2014) Fitting Data with Different Error Models, The Mathematica Journal (accepted for publication, to be published in April)

Paláncz, B, Somogyi, A, Lovas, T, and Molnár, B (2013) Plane Fitting to Point Cloud via Gröbner Basis, available via http://library.wolfram.com/infocenter/MathSource/8769/#downloads

Poppinga J, Vaskevicius N, Birk A and Pathak K (2006) Fast Plane Detection and Polygonalization in noisy 3 D Range Images, Int. Conf. on Intelligent Robots and Systems (IROS), Nice, France, IEEE Press 2008

Rose C and Smith D (2000) Symbolic Maximum Likelihood Estimation with Mathematica, The Statistician 49, pp .229 - 240

Russeeuw P J and Van Driessen K (1999) A Fast Algorithm for the Minimum Covariance Determinant Estimator, TECHNOMETRICS, Vol. 41. No. 3. pp. 212 - 223.

Stathas D, Arabatzi O, Dogouris S, Piniotis G, Tsini D and Tsinis D (2003) New Monitoring Techniques on the Determination of Structure Deformation, Proc. 11 th FIG Symp. on Deformation Measurements, Santorini, Greece.

Zuliani M (2012) RANSAC for Dummies, vision.ece.ucsb.edu/~zuliani

Yang M Y and Förtsner W (2010) Plane Detection in Point Cloud Data, TR - IGG - P - 2010 - 01, Techn.Report. Nr .1, Dept. of Photogrammetry Inst. of Geodesy and Geo - information, Uni., Bonn, Germany.

Yaniv Z (2010) Random Sample Consensus (RANSAC) Algorithm, A Generic Implementation, Georgetown University Medical Center, Washington, DC, USA, zivy@isis.georgetown.edu