Geometry Manual

1 Introduction

This document describes the constructors, methods and properties which are included in 'Geometry.dll' and which may be accessed in any Microsoft .NET environment. These perform many geometric calculations, including the fitting of a shape to a cloud of points (in a least squares sense). 'Matrix.dll' must also be added as a reference to your project.

2 Version history

2.1 Version 1.0.0.*

1.0.0.0	Initial Version
1.0.0.1	Addition of Complex, ComplexVector and ComplexMatrix classes. The graph class is not included.
1.0.0.2	Change in structure to make this dll dependant on 'Matrix.dll'. Refactoring and name changing to make all contents compatible with Microsoft's style and design guide. Signing by a strong key so that this dll can be added to the GAC.
1.0.0.2	Changes necessary because of the refactoring of Matrix.dll to make it too conform to the Microsoft style and design guide.
1.0.0.3	Addition of planar quadratic class
1.0.0.5	Addition of further options for cylinder fit.
1.0.0.8	Fix of bug in SyntheticAxisSystem function (bug when all elements of the line's direction are identical). Addition of RMS and MaxError property. Addition of CreatePointArray to Cylinder3D class. Addition of cylinder Height property.
1.0.0.9	Change of prototype of one of the cylinder fit functions.
1.0.0.10	Addition of cone class.
1.0.0.11	Addition of another prototype for circle fit.
1.0.0.12	Fix bug in Circle3D constructor. Addition of IntersectionWith and NearestPointTo functions for Point3D and Line3D classes.

2.2 Manual History

9th June 2013	Initial document.
30th December 2013	Matrix and Vector classes in this Dll are only at Version 1.0.0.* and do not include The Graph class.
17th March 2014	Addition of Complex, ComplexVector and ComplexMatrix classes. The graph class is not included.
4th April 2014	Change in structure to make this dll dependant on 'Matrix.dll'. Refactoring and name changing to make all contents compatible with Microsoft's style and design guide. Signing by a strong key so that this dll can be added to the GAC.
1.0.0.2	Changes necessary because of the refactoring of Matrix.dll to make it too conform to the Microsoft style and design guide.

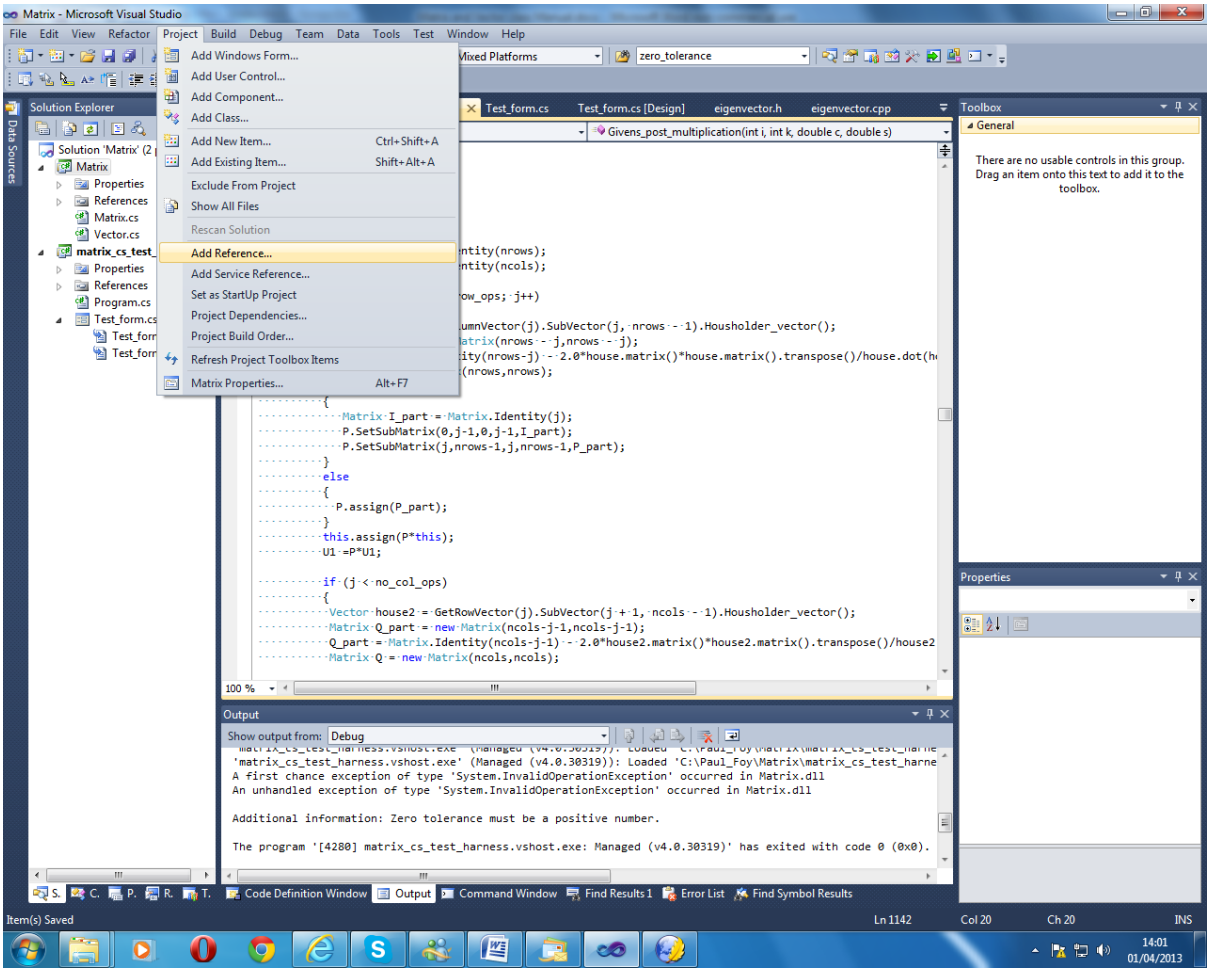
22nd October 2014	Clarification of the relationship between ‘Geometry.dll’ and ‘Matrix.dll’.
20th March 2015	Addition of planar quadratic class.
8th March 2015	Addition of further options for cylinder fit.
11th April 2015	Addition of RMS and MaxError property. Addition of CreatePointArray to Cylinder3D class. Addition of cylinder Height property.
15th April 2015	Addition of cone class.
30th April 2015	Addition of additional prototype for circle fit.
7th May 2015	Addition of IntersectionWith and NearestPointTo functions for Point3D and Line3D classes.

3 Installation Instructions

- The Geometry.dll file
- The license file.

Use of this software implies acceptance of the license condition contained in the license file.

To access the method and properties in the dll add it as a reference to you Microsoft C# .NET, VB.NET or C++/CLI .NET project:



This software depends on the class 'Matrix.dll' available separately. It is necessary also to add a reference to 'Matrix.dll' to the application project.

As the types of 'Geometry.dll' are hidden behind the namespace *MathematicalServices.Geometry'* and the types of 'Matrix.dll' are hidden behind the namespace *MathematicalServices* it is usually necessary to add the statements

```
using MathematicalServices;
```

```
using MathematicalServices.Geometry;
```

to project files.

4 Point3D class

4.1 Constructors

4.1.1 Point3D()

Forms a point each of whose 3 co-ordinate is 0.0.

C# example:

```
Point3D p = new Point3D();
```

4.1.2 Point3D(double x,double y,double z)

Forms a point with the given x, y and z ordinates.

C# example:

```
Point3D p = new Point3D(1.0,2.0,3.0);
```

4.1.3 Point3D(Point3D p)

Form a point whose coordinates are the same as the one provided.

C# example:

```
Point3D p = new Point3D(p1);
```

4.1.4 Point3D(Vector v)

Forms a vector with the first three coordinates of the vector supplied.

C# example:

```
Vector v  = new Vector();
v[0] = 1.0;
v[1] = 2.0;
v[2] = 3.0;
Point3D p = new Point3D(v);
```

4.2 Operators

4.2.1 double [i]

Gets and sets the i th index element. If i is less than 0 or greater than 2 an exception is thrown.

C# example:

```
double element1, element2;
Point3D p = new Point3D();
element1=1.5;
p[0]=element1;
element2=p[1];
```

4.2.2 Vector operator - (Point3D $p1$,Point3D $p2$)

Returns the 3 dimensional vector which is the difference of the two points.

C# example:

```
Point3D p1 = new Point3D(p);
Point3D p2 = new Point3D(p);
Vector   v  = new Vector(p1-p2) //should be the zero vector
```

4.2.3 Vector Subtract(Point3D $p1$,Point3D $p2$)

Performs the subtraction operation as above in languages which can not override the - operator.

4.2.4 Point3D operator + (Point3D p, Vector v)

Adds the first three coordinates of v to point p to return a new point.

C# example:

```
Point3D p1 = new Point3D(p);
Vector v1 = new Vector(v)
Point3D p = new Point3D(p1+v1);
```

4.2.5 Point3D operator +(Point3D p, Vector v)

Performs the addition operation as above in languages which can not override the + operator.

4.2.6 bool operator ==(Point3D $p1$,Point3D $p2$)

Returns true if two points are identical i.e. they refer to the same instance or two instances are element-wise identical. Otherwise false is returned.

C# example:

```
Point3D mypoint3 = new Point3D();
mypoint3.Assign(mypoint2);

if (mypoint2 == mypoint3)
{
          ; // should be identical
}
else if (mypoint2 != mypoint3)
{
   ; // different
}
```

4.2.7 bool operator !=(Point3D $p1$,Point3D $p2$)

Returns false if two points are identical i.e. they refer to the same instance or two instances are element-wise identical. Otherwise true is returned.

C# example:

See 4.2.6.

4.3 Methods

4.3.1 void assign(Point3D p)

Sets the current point to be p.

C# example:

```
Point3D p = new Point3D();
p.Assign(this);
```

4.3.2 Point3D ConvertPointFromLocalSystem(Matrix M ,Point3D O)

Converts the current point from a local coordinate system. M is a 3 by 3 transformation matrix. The columns of M are the coordinates of the axes of the local coordinate system which the current point is expressed in, expressed in terms of the coordinate system we are converting to. O is the origin of the local coordinate system expressed in terms of the coordinate system we are converting to.

4.3.3 Point3D convertPointToLocalSystem(Matrix M ,Point3D O)

Converts the current point to a local coordinate system. M is a 3 by 3 transformation matrix. The columns of M are the coordinates of the axes of the the local coordinate system , expressed in terms of the current coordinate system. O is the origin of the local coordinate system expressed in terms of the current coordinate system.

C# example:

```
Point3D [] syn_points = new Point3D[m_size];
Matrix M = new Matrix(3,3);
M.SetColumnVector(0,x_axis);
M.SetColumnVector(1,y_axis);
M.SetColumnVector(2,z_axis);
for (i = 0; i < m_size; i++)
{
    syn_points[i] = convertPointToLocalSystem(M,plane.point);
}
```

4.3.4 double DistanceTo(Point3D p)

Returns the distance from the current point, P , to p .

4.3.5 double DistanceTo(Plane3D pl)

Returns the shortest distance from the current point P to the plane pl . This is computed by dropping a perpendicular from the current point to the plane (see Figure 1).

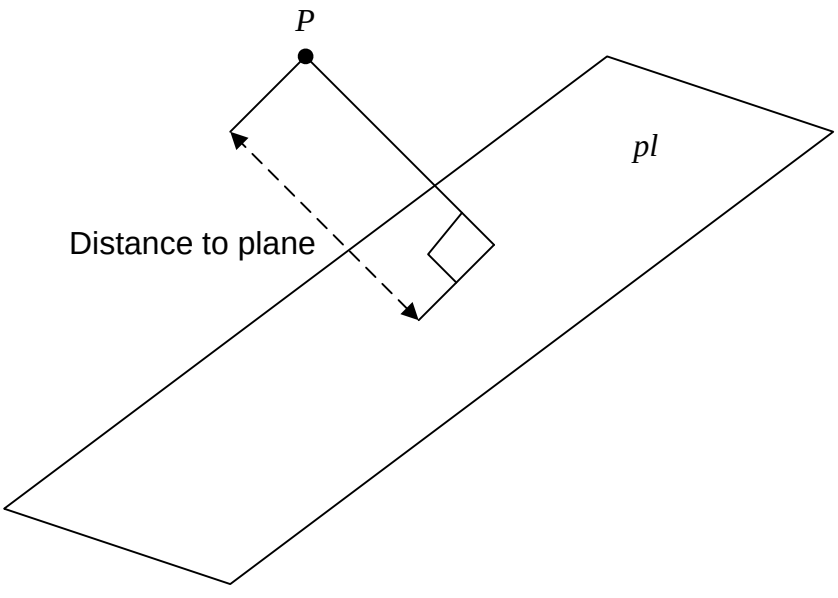


Figure 1 - Distance from a point to a plane.

4.3.6 double DistanceTo(Circle3D *cir*)

Returns the shortest distance from the current point, P, to the 3D circle *cir* (see Figure 2).

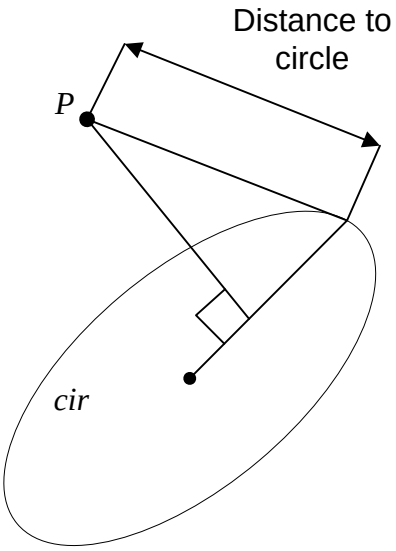


Figure 2 - Distance to circle.

4.3.7 double distance_to(Line3D *l*)

Returns the shortest distance from the current point, P, to the 3D line l. This is computed by dropping a perpendicular from the current point to the line.

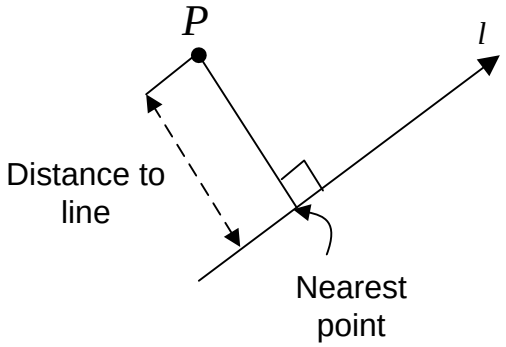


Figure 3 - Distance to line.

4.3.8 double DistanceTo(Sphere3D *s*)

Returns the shortest distance from the current point, P, to the 3D sphere s.

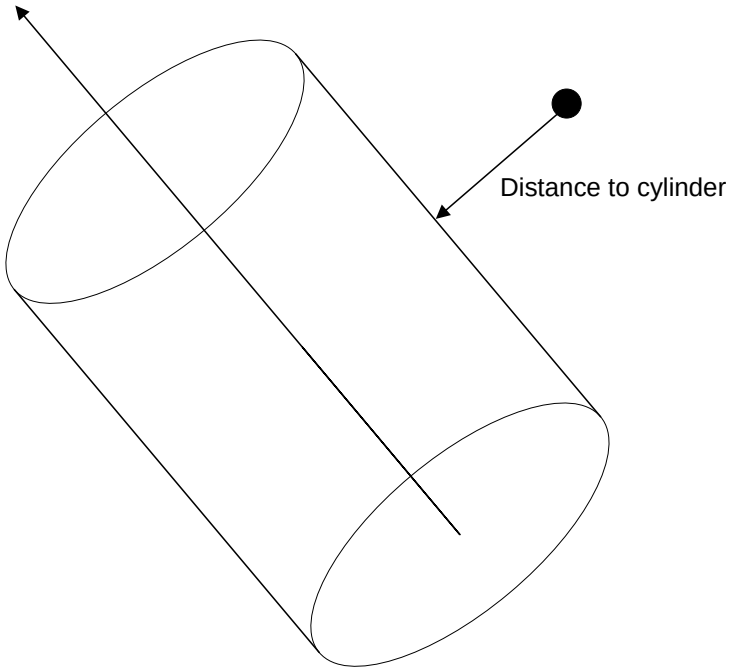


Figure 4 - Distance from a point to a cylinder.

4.3.9 double DistanceTo(Cylinder3D c)

Returns the shortest distance from the current point P, to the 3D cylinder c.

4.3.10 Point3D RotationAbout(Line3D l,angle θ)

Rotates the current point, P, about the line l, by the angle θ. A positive θ representing clockwise rotation, as looking along the direction of the line.

C# example:

```
//Construct a helical spiral of points
   Line3D plane_normal = new Line3D();
   plane_normal.location.Assign(loc);
   plane_normal.direction.Assign(normal);
   double R = 10.0;
   double radius;
   double theta;
   for (i = 0; i < no_points; i++)
   {
      radius = R*(1.0+i/no_points);
      theta  = 1.0*Math.PI*i/no_points;
      Point3D point = new Point3D(loc + radius*p_normal);
      Point3D rotated_point = new Point3D();
      points[i] = point.RotationAbout(plane_normal,theta);
   }
```

5 Point3DArray class

5.1 Constructor

5.1.1 Point3DArray(Point3D[] *pts* ,int *size*)

Initialises an array of points of size *size* .

C# example:

```
Point3D[] points = new Point3D[no_points];

for (i = 0; i < no_points; i++)
{
   points[i] = new Point3D(start + i * direction);
}
Point3DArray array = new Point3DArray(points, no_points);
```

5.2 Properties

5.2.1 int Size

Gets and sets the size of the points array.

5.2.2 Point3D [*i*]

Gets or sets the point at the *i* th index location. If *i* is less than zero or greater than or equal to the size of the array an error is thrown.

5.3 Methods

5.3.1 int FitLine(Line3D *line*)

Fits a line (in a least squares sense) to the array of points (see Figure 5). The location and direction of the line are calculated. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

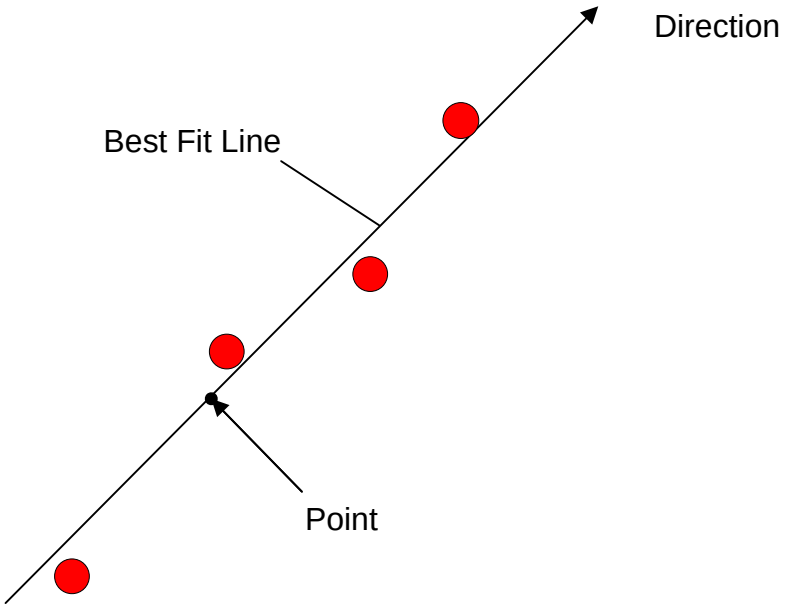


Figure 5 - Best fit line

C# example:

```
Point3DArray array = new Point3DArray(points, no_points);

Line3D line = new Line3D();
array.FitLine(line);
```

5.3.2 int FitQuadratic(Quadratic3D *quad*)

Fits a planar quadratic (in a least squares sense) to the array of points. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

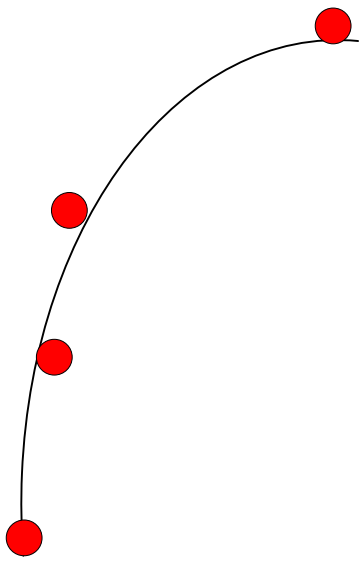


Figure 6 - Best Fit Quadratic

C# example:

```
Point3DArray array = new Point3DArray(points, no_points);

Quadratic3D quad = new Quadratic3D();
array.FitQuadratic(quad);
```

5.3.3 int FitPlane(Plane3D *plane*)

Fits a plane (in a least squares sense) to the array of points (see Figure 7). A point on the plane and the plane's normal are calculated. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

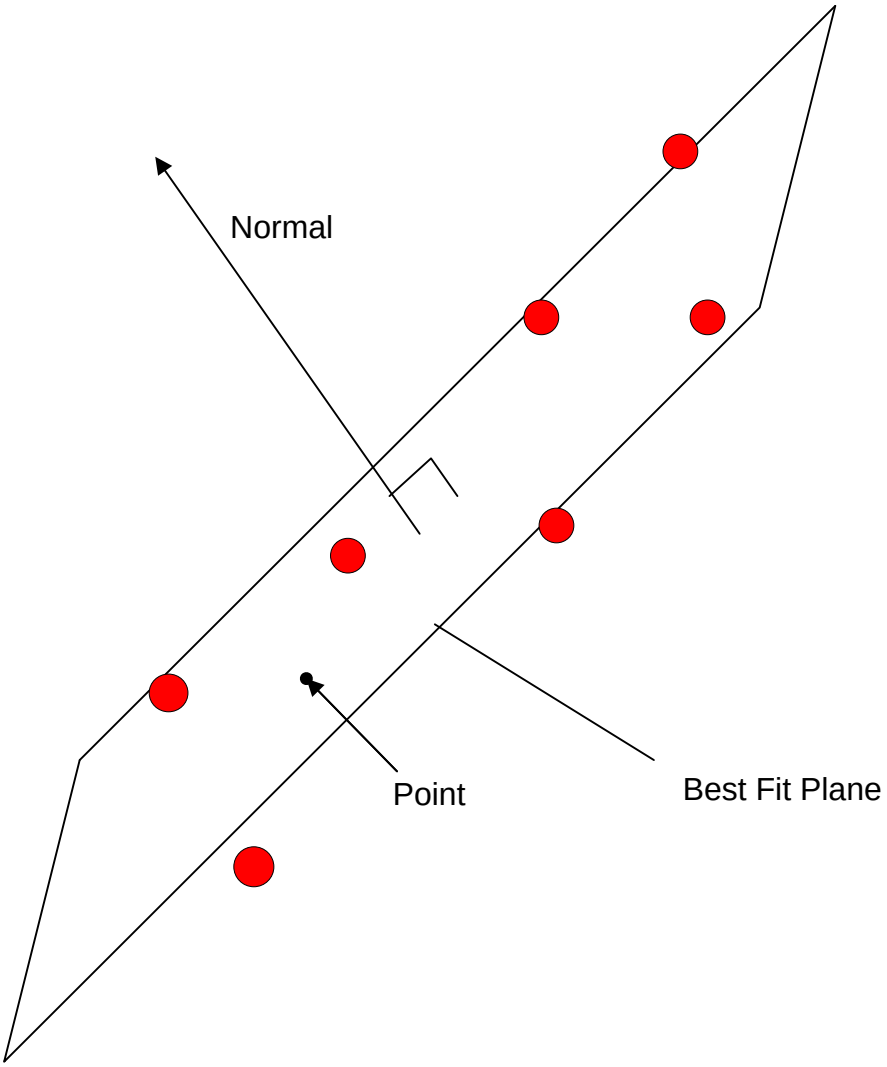


Figure 7 - The best fit plane

C# example:

```
Point3DArray array = new Point3DArray(points, no_points);

Plane3D plane = new Plane3D();
array.FitLine(plane);
```

5.3.4 int FitSphere(Sphere3D *sphere*)

Fits a sphere (in a least squares sense) to the array of points (see Figure 8). The sphere's centre and radius are calculated. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

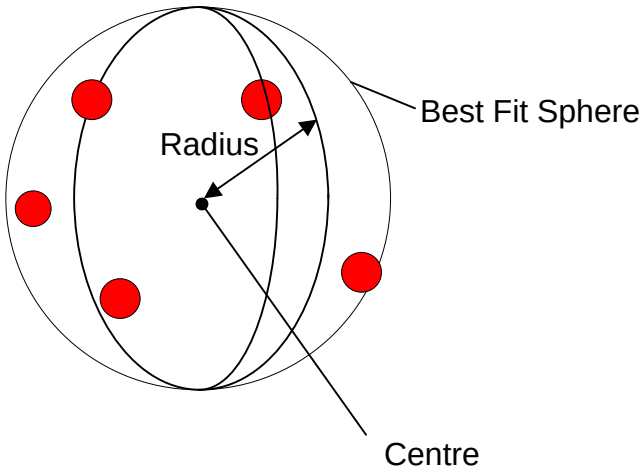


Figure 8 - Best fit sphere

C# example:

```
Point3DArray array = new Point3DArray(points, no_points);

Sphere3D sphere = new Sphere3D();
array.FitSphere(sphere);
```

5.3.5 int FitCircle(Circle3D *circle*)

5.3.6 int FitCircle(Circle3D *circle*, double *radius*)

5.3.7 int FitCircle2(Circle3D *circle*, double *radius*)

Fits a circle (in a least squares sense) to the array of points (see Figure 9). The circles's centre and radius are calculated, together with the normal defining the plane in which the circle lies. The second routine permits an *initial guess* for the radius to be supplied. The third routine does not have radius as a variable i.e. it is fixed. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

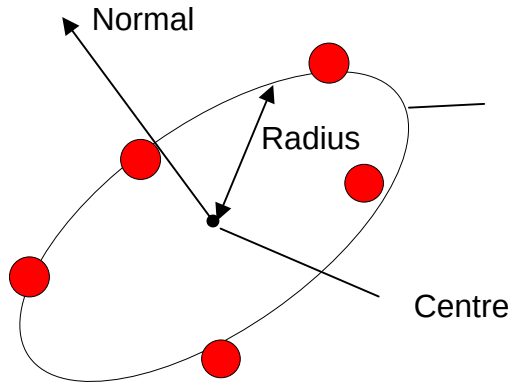


Figure 9 - Best fit circle

C# example:

```
Point3DArray array = new Point3DArray(points, no_points);

Circle3D circle = new Circle3D();
array.FitCircle(circle);
```

5.3.8 int FitCylinder(Cylinder3D *cyl* , int *type*)

Fits a cylinder (in a least squares sense) to the array of points (see Figure 10). The line defining the cylinder together with the radius of the cylinder are returned. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

type determines the procedure to use to find an initial estimation of the cylinder's axis (and hence radius and point on axis). It can take the following values:

- 0 - use a best fit line technique. Good when the data is uniformly spread around the full width and height of the cylinder and the height is much greater than the width.
- 1 - use a best fit plane technique. Good when the data is nearly coplanar.
- 2 - use a technique based upon the normal of the plane defined by the centroid of the data and two data points closest to it. This technique is good when the data is clustered together in one region of the cylinder. The algorithm is according to the following C# code:

```
//First get the centroid of the data.
Vector centroid = new Vector(3);
for (int i = 0; i < m_size; i++)
{
   for (int j = 0; j < 3; j++)
   {
      centroid[j] = centroid[j] + m_array_data[i][j];
   }
}
centroid = centroid / m_size;

Point3D C = new Point3D(centroid);

//Identify and store the point in the data nearest to the centroid.
double max  = 1.0E-12;
int nearest = 0;
for (int i = 0; i < m_size; i++)
{
    double temp = this[i].DistanceTo(C);
    if (temp < max)
    {
        nearest = i;
        max  = temp;
    }
}
Point3D P1 = new Point3D(this[nearest]);
double radius = C.DistanceTo(P1);

//Identify and store the point in the data >= 0.01*radius from P1 and nearest
to C and P1
max = 1.0E12;
```

```
nearest = 0;
for (int i = 0; i < m_size; i++)
{
    double temp1 = this[i].DistanceTo(P1);
    if (temp1 >= (0.01*radius))
    {
        double temp2 = this[i].DistanceTo(C);
      if ( temp2 < max )
      {
          nearest = i;
          max = temp2;
      }
   }
}
Point3D P2 = new Point3D(this[nearest]);
Vector v1 = new Vector(P2-C);
Vector v2 = new Vector(P1-C);
Vector v = new Vector(3);
v[0] = v1[1] * v2[2] - v1[2] * v2[1];
v[1] = v1[2] * v2[0] - v1[0] * v2[2];
v[2] = v1[0] * v2[1] - v1[1] * v2[0];
line.Location = C;

if (v.Modulus > Matrix.ZeroTolerance)
{
    line.Direction = v;
}
else
{
   line.Direction = new Vector(3);
   line.Direction[2] = 1.0;
}
```

5.3.9 int FitCylinder(Cylinder3D *fitted*, Cylinder3D *nom*)

5.3.10 int FitCylinder(Cylinder3D *fitted*, Line3D *nom*)

5.3.11 int FitCylinder(Cylinder3D *fitted*, Vector *nom axis*)

5.3.12 int FitCylinder(Cylinder3D *fitted*, Line3D *nom*, double *nomR*)

5.3.13 int FitCylinder(Cylinder3D *fitted*, Vector *nom axis*, double *nomR*)

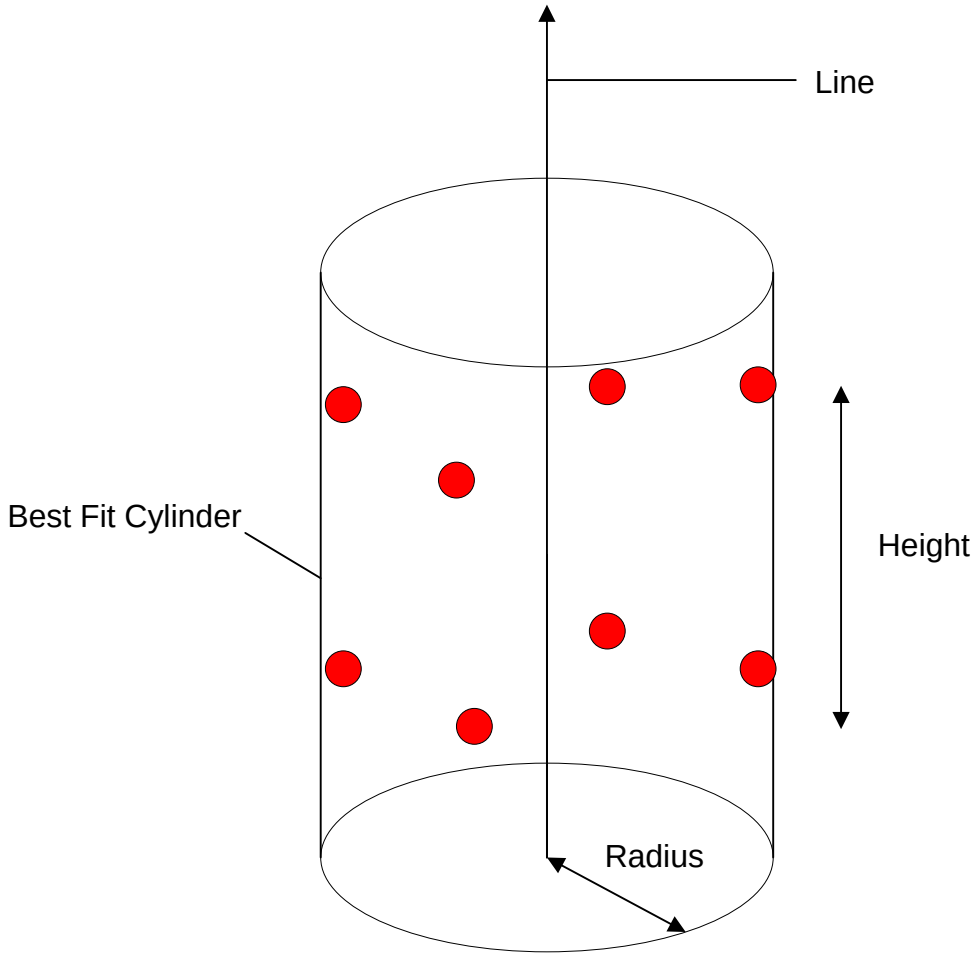


Figure 10 - Best fit cylinder

The fit will either determine its own initial starting conditions (5.3.8) or a variety of initial conditions can be supplied. These range from the full nominal cylinder to just the vector describing the axis of the cylinder or the nominal axis and the nominal radius.

C# example:

```
Point3DArray array = new Point3DArray(points, no_points);
Cylinder3D cylinder = new Cylinder3D();
Line3D line = new Line3D(centre, normal);     //centre is guess of point on axis
                                              //normal is guess of axis
Cylinder3D nominal = new Cylinder3D(line, R); //R is guess of radius

Cylinder3D cylinder = new Cylinder3D ();
array.FitCylinder(cylinder);           //or
```

```
array.FitCylinder(cylinder, nominal);  //or
array.FitCylinder(cylinder, line);     //or
array.FitCylinder(cylinder, normal);   //or
array.FitCylinder(cylinder, line, R);  //or
array.FitCylinder(cylinder, normal, R);
```

5.3.14 int FitCone(Cone3D *cone*, int *type*)

Fits a cone (in a least squares sense) to the array of points. The line defining the cone together with the radius at the point on the line and the semi-angle are returned. If the numerical routine employed fails to converge 0 is returned; otherwise 1 is returned.

type determines the procedure to use to find an initial estimation of the cone's axis (and hence radius and point on axis). It can take the following values:

- 0 - use a best fit line technique. Good when the data is uniformly spread around the full width and height of the cylinder and the height is much greater than the width.
- 1 - use a best fit plane technique. Good when the data is nearly coplanar.
- 2 - use a technique based upon the normal of the plane defined by the centroid of the data and two data points closest to it. This technique is good when the data is clustered together in one region of the cone. See 5.3.8.

5.3.15 int FitCone(Cone3D *fitted*, Cone3D *nom*)

5.3.16 int FitCone(Cone3D *fitted*, Vector *nom axis*)

5.3.17 int FitCone(Cone3D *fitted*, Vector *nom axis*, double *nomAngle*)

5.3.18 int FitCone(Cone3D *fitted*, Vector *nomAxis*, double *nomRadius*, double *nomAngle*)

5.3.19 int FitCone(Cone3D *fitted*, Line3D *nom*, double *nomAngle*)

These routines take nominal quantities with which to base the initial values of the cone's parameters. The semi-angle is in radians. The remaining parameters are deduced from a combination of the nominals and the data.

6 Line3D class

A line is specified by a 3D-point and a direction. This direction need not be a unit vector. However the direction can not be the zero vector.

6.1 Constructors

6.1.1 Line3D()

Initializes the point to be (0,0,0) and the direction to be (0,0,1).

C# example:

```
Line3D line = new Line3D();
```

6.1.2 Line3D(Line3D *line*)

Initializes the line to have the location and direction of that of *line*.

C# example:

```
Line3D line = new Line3D(line2);
```

6.1.3 Line3D(Point3D *loc*,Vector *dir*)

Initialise the line to have location that of *loc*, and direction that of *dir*. If the dimension of *dir* is not 3 or if it is the zero vector an error is thrown.

C# example:

```
Line3D line = new Line3D(loc,dir);
```

6.2 Properties

6.2.1 Vector Direction

Gets and sets the line's direction.

6.2.2 Point3D Location

Gets and sets the line's location.

6.2.3 int RMSError

Gets and sets the root mean square of the distances between the data and the best fit line.

6.2.4 int MaxError

Gets and sets the maximum error of the distances between the data and the best fit line.

6.2.5 double AngleTolerance

Gets and sets the tolerance which is used when comparing if two directions are collinear.

The default value is 1.0E-8 radians.

C# example:

```
Vector n = new Vector(plane.normal.unit_vector);
Vector l = new Vector(direction.unit_vector);
if ( Math.Abs(Math.Acos(n.dot(l))) < Line3D.angle_tolerance )
{
            //The plane and the line are parallel to within our
tolerance
...
```

6.3 Methods

6.3.1 void Assign(Line3D *line*)

Sets the current line to have the same direction and location as *line*.

6.3.2 double DistanceTo(Point3D *pt*)

Return the distance to *pt*. See 4.3.7.

6.3.3 double DistanceTo(Plane3D *plane*)

Returns the distance to *plane*. If the line intersects the plane this distance is zero. Otherwise we drop a perpendicular from the line to the plane to compute the distance.

6.3.4 double DistanceTo(Line3D *line*)

Returns the distance to *line*. This distance is the length of the common perpendicular between the two lines (see Figure 11).

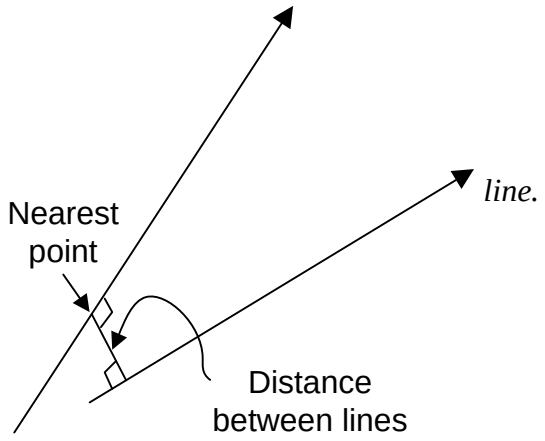


Figure 11 - Distance between two lines.

6.3.5 double DistanceTo(Sphere3D *sphere*)

Returns the distance to *sphere*. If the line intersects the sphere this distance is zero.

C# example:

```
Point3D centre = new Point3D(100.0,0.0,0.0);
Sphere3D sphere = new Sphere3D(10.0,centre);

Point3D loc = new Point3D(0.0,0.0,0.0);
Vector dir  = new Vector(3);
dir[0] = 0;
dir[1] = 0;
```

```
dir[2] = 1;
Line3D line = new Line3D(loc,dir);

Matrix.ZeroTolerance = 1.0E-8;
int conv=0;

double dist = line.DistanceTo(sphere, ref conv); //should be 90
```

6.3.6 double DistanceTo(Circle3D *circle* , ref int *conv*)

Returns the distance to *circle*. Because a numerical method is employed, *conv* is used to determine if the method has converged (1) or not (0). The number of iterations in the numerical routine is governed by 'Matrix.max_no_loops'.

C# example:

```
Vector normal = new Vector(3);
normal[2] = 1.0;
Point3D centre = new Point3D(0.0,0.0,0.0);
Circle3D circle = new Circle3D(10.0,centre,normal);

Point3D loc = new Point3D(0.0,0.0,0.0);
Vector dir  = new Vector(3);
dir[0] = 0;
dir[1] = 0;
dir[2] = 1;
Line3D line = new Line3D(loc,dir);

Matrix.ZeroTolerance = 1.0E-8;
int conv=0;

double dist = line.DistanceTo(circle, ref conv); //should be 10
```

6.3.7 Boolean IntersectsWith(Point3D *pt*)

Returns true if the parent line passes through *pt* . Otherwise false.

C# example:

```
Vector n = new Vector(3);
n[0] = 1.0; n[1] = 1.0; n[2] = 0.0;
Point3D P = new Point3D(0,0,0);
Line3D Line = new Line3D(P,n);
Point3D r = P + 3.0*n;
Boolean rtn = Line.IntersectsWith(r);
```

6.3.8 Boolean IntersectsWith(Line3D *l* ,Point3D *pt*)

Returns true if the parent line intersects with the line *l* . Otherwise false is returned. If it does intersect *pt* is set to the point of intersection.

C# example:

```
Boolean rtn2 = Line.IntersectsWith(Line2,intersection);
```

6.3.9 Point3D NearestPointTo(Point3D *pt*)

Returns the nearest point on the parent line to the point *pt*. See Figure 3.

C# example:

```
Point3D nearest = Line.NearestPointTo(r);
```

6.3.10 Point3D NearestPointTo(Line3D *l*)

Returns the nearest point on the parent line to the line *l*. See Figure 11.

6.3.11 Line3D RotationAbout(Line3D *line*, double *angle*)

Returns the line which is the result of a rotation about *line*. The rotation is in a clockwise sense as looking along *line* (see Figure 12).

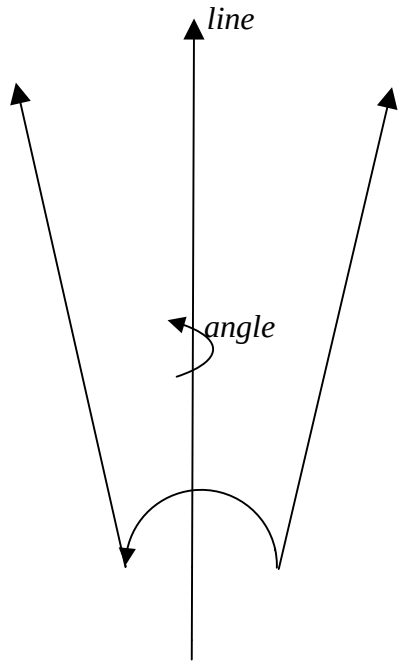


Figure 12 - Rotation of a line about a line.

C# example:

```
Line3D line4 = new
Line3D(line2.RotationAbout(line3,Math.PI/2.0));
```

6.3.12 void SyntheticAxisSystem(Vector *x_axis*,Vector *y_axis*,Vector *z_axis*)

Constructs an orthogonal synthetic axis system whose *z_axis* is parallel to the direction of the line.

C# example:

```
Line3D line = new Line3D(circle.Centre,circle.Normal);
Vector x_axis = new Vector(3);
Vector y_axis = new Vector(3);
Vector z_axis = new Vector(3);
line.SyntheticAxisSystem(x_axis,y_axis,z_axis);
```

6.3.13 int Fit(Point3Darray *pts*)

Calculate a direction and location such that the resultant line is the best fit (in the least squares sense) to the points array of data. If the numerical routine used converges then 1 is returned; otherwise 0 is returned. See 5.3.1.

7 Plane3D class

7.1 Constructors

A plane is defined by a point in the plane and a normal to the plane. We do not require the normal to be a unit vector.

7.1.1 Plane3D()

A point at (0,0,0) and a normal of (0,0,1) is constructed.

7.1.2 Plane3D(Plane3D *plane*)

A plane is constructed having the same point and normal as *plane*.

7.1.3 Plane3D(Point3D *point*,Vector *normal*)

A plane is constructed with point *point* and normal *normal*. If *normal* is not a 3-dimensional vector or is the zero vector an error is thrown.

C# example:

```
Plane3D plane = new Plane3D(pt,dir);
```

7.2 Properties

7.2.1 Vector Normal

This gets or sets the normal of the plane.

7.2.2 Point3D Point

This gets or sets the point in the plane.

7.2.3 int RMSError

Gets and sets the root mean square of the distances between the data and the best fit plane.

7.2.4 int MaxError

Gets and sets the maximum error of the distances between the data and the best fit plane.

7.3 Methods

7.3.1 void Assign(Plane3D *plane*)

Initializes the point and normal to those of *plane*.

7.3.2 double DistanceTo(Point3D *point*)

Returns the distance of the plane from *point*. See 4.3.5.

7.3.3 double DistanceTo(Line3D *line*)

Returns the distance of the plane to *line*. See 6.3.3.

7.3.4 double DistanceTo(Plane3D *plane*)

Returns the distance to the plane *plane*. If the planes intersect the distance between them is zero. Otherwise the distance is calculated by dropping a perpendicular between them.

7.3.5 Int Fit(Point3DArray *pts*)

Computes the best fitting plane (in a least squares sense) to the given points. See 5.3.3.

8 Sphere class

A sphere in 3-dimensional space is specified by its centre and radius.

8.1 Constructors

8.1.1 Sphere3D()

Initializes a sphere with zero radius and centre at (0,0,0).

8.1.2 Sphere3D(Sphere3D *sphere*)

Initializes a sphere with radius and centre that of *sphere*.

8.1.3 Sphere3D(double *R*,Point3D *centre*)

Initializes a sphere with radius *R* and centre *centre*.

C# example:

```
centre = new Point3D(100.0, 0.0, 0.0);
Sphere3D sphere = new Sphere3D(10.0, centre);
```

8.2 Properties

8.2.1 double Radius

Gets and sets the radius of the sphere.

8.2.2 Poin3D Centre

Gets and sets the centre of the sphere.

8.2.3 int RMSError

Gets and sets the root mean square of the distances between the data and the best fit sphere.

8.2.4 int MaxError

Gets and sets the maximum error of the distances between the data and the best fit sphere.

8.3 Methods

8.3.1 void Assign(Sphere3D *sphere*)

Initializes the radius and centre to be those of *sphere*.

8.3.2 Double DistanceTo(Point3D pt)

Returns the distance to pt. See 4.3.6.

8.3.3 int Fit(Point3DArray pts)

Computes the best fitting sphere (in a least squares sense) to the given points. See 5.3.4.

9 Quadratic3D class

This class concerns a curve which is defined by the equation:

$$\vec{r} = Ax^2 + Bx + C$$

Equation 1

where A, B and C are 3-dimensional points and x is a real parameter. It is understood that all points of the quadratic lie in a plane. Alternatively the quadratic may be specified by giving the normal of the plane in which it lies, $\vec{e}_z$, and the other two orthonormal axes, $\vec{e}_x$ and $\vec{e}_y$. The point $\vec{r}$ is then described by specifying a, b and c such that

$$\vec{r} = x\vec{e}_x + (ax^2 + bx + c)\vec{e}_y$$

Equation 2

9.1 Constructors

9.1.1 Quadratic3D()

Initializes the quadratic to be zero (A, B, C, a, b, c identically zero) and $\vec{e}_x, \vec{e}_y, \vec{e}_z$ the standard unit orthormal vectors.

9.1.2 Quadratic3D(Point3D A,Point3D B,Point 3D C)

Initialises the quadratic according to Equation 1 and Equation 2.

9.1.3 Quadratic3D(Vector ex,Vector ey,Vector ez,double a,double b,double c)

Initialises the quadratic according to Equation 1 and Equation 2.

9.2 Properties

9.2.1 Vector XAxis

Gets the x-axis with respect to which the quadratic's parameters are defined.

9.2.2 Vector YAxis

Gets the y-axis with respect to which the quadratic's parameters are defined.

9.2.3 Vector ZAxis

Gets the normal of the plane in which the quadratic is understood to lie.

9.2.4 double a, b, c

Gets the parameters a, b or c

9.2.5 Point3D A, B, C

Gets the parameters A, B, or C.

9.3 Methods

9.3.1 double DistanceTo(Point3D pt)

Returns the shortest distance between the quadratic and the 3D point pt.

9.3.2 Int Fit(Point3DArray pts)

Initialises the member variables of the quadratic with those parameters which give rise to a best fit of the supplied points pts.

If the underlying numerical routine has converged 1 is returned otherwise 0.

10 Circle3D class

A circle in 3D space is specified by its centre, radius and normal to the plane in which it lies. Note that normal need not be a unit vector.

10.1 Constructors

10.1.1 Circle3D()

Initializes the radius to zero the centre to (0,0,0) and the normal to (0,0,1).

10.1.2 Circle3D(Circle3D *circle*)

Initialises the radius, centre and normal to be those of *circle* .

10.1.3 Circle3D(double R ,Point3D *centre* ,Vector n)

Initializes the radius to be R , the centre to be *centre* , and the normal to be n . n does not have to be a unit vector. If the dimension of the vector n is not 3 or if it is the zero vector an error is thrown.

10.2 Properties

10.2.1 double Radius

Gets or sets the circle's radius.

10.2.2 Point3D Centre

Gets or sets the circle's centre.

10.2.3 Vector Normal

Gets or sets the circle's normal.

10.2.4 int RMSError

Gets and sets the root mean square of the distances between the data and the best fit circle.

10.2.5 int MaxError

Gets and sets the maximum error of the distances between the data and the best fit circle.

10.3 Methods

10.3.1 void Assign(Point3D *cir*)

Initializes the circle with the members of *cir* .

10.3.2 Point3DArray CreatePointArray(int N)

Creates a series of N points, centered on the parent circle's centre and in the plane defined by the parent circle's axis. The points trace out a circle.

C# example:

```
Point3DArray points_fitted = cylinder.CreatePointArray(N);
```

10.3.3 double DistanceTo(Point3D *pt*)

Returns the distance to *pt*. See 4.3.6.

10.3.4 int Fit(Point3Darray *pts*)

10.3.5 int Fit(Point3Darray *pts*, double *radius*)

10.3.6 int Fit2(Point3darray *pts*,double *radius*)

Computes the best fitting circle (in a least squares sense) to the given points. See 5.3.5, 5.3.7 and 5.3.6.

11 Cylinder3D class

A cylinder in 3D space is defined by a line describing its axis, together with its radius.

11.1 Constructors

11.1.1 Cylinder3D()

Initializes a cylinder with zero radius and an axis passing through (0,0,0) in the direction of the vector (0,0,1);

11.1.2 Cylinder3D(Cylinder3D cyl)

Initializes a cylinder with radius and axis that of cyl .

11.1.3 Cylinder3D(Line3D $line$,double R)

Initializes a cylinder with radius R and line $line$.

C# example:

```
Cylinder3D cyl = new Cylinder3D(line,10.0);
```

11.2 Properties

11.2.1 double Radius

Gets and sets the radius of the cylinder

11.2.2 Line3D Line

Gets and sets the axis of the cylinder.

11.2.3 double RMSError

Gets the average of the square root of the sum of the errors: $\sqrt{\frac{\sum_{i=1}^{N} e_i^2}{N}}$, where N is the number of points and e_i is the distance from point i to the best fit cylinder.

C# example:

```
double max_error = Matrix.ZeroTolerance;
cylinder.RMSError = 0.0;
for (int i = 0; i < m_size; i++)
{
    double temp = Math.Abs(r_i[i] - cylinder.Radius);
    cylinder.RMSError += (temp*temp);
    if ( temp > max_error )
    {
        max_error = temp;
    }
}
cylinder.MaxError = max_error;
cylinder.RMSError = Math.Sqrt( cylinder.RMSError / m_size );
```

11.2.4 double MaxError

Gets $\max_{i=1}^{N}\{e_i\}$ - the maximum of the distances of point i to the best fit cylinder.

C# example:

See 11.2.3.

11.2.5 double Height

Returns the distance along the cylinder's axis between the point representing one extreme of the data set and the point representing the other extreme.

C# example:

```
//Get the height of the best fit cylinder
double max = -1.0E12, min = 1.0E12;
for (int i = 0 ; i < m_size; i++)
{
    if (syn_points[i][2] > max)
    {
        max = syn_points[i][2];
    }
    if (syn_points[i][2] < min)
    {
        min = syn_points[i][2];
    }
}
cylinder.Height = max - min;
```

11.3 Methods

11.3.1 void Assign(Cylinder3D cyl)

Sets the cylinder to cyl.

11.3.2 DistanceTo(Point3D pt)

Returns the shortest distance from the cylinder to the point.

11.3.3 int Fit(Point3DArray *pts*, int *type*)

11.3.4 int Fit(Point3DArray *pts*, Cylinder3D *nom*)

11.3.5 int Fit(Point3DArray *pts*, Line3D *nom*)

11.3.6 int Fit(Point3DArray *pts*, Line3D *nom*, double *nomR*)

11.3.7 int Fit(Point3DArray *pts*, Vector *nomAxis*)

11.3.8 int Fit(Point3DArray *pts*, Vector *nomAxis*, double *nomR*)

Computes the best fitting cylinder (in a least squares sense) for the set of points. See 5.3.8 etc.

11.3.9 Point3DArray CreatePointArray(int *N*, double *revs*)

Creates a series of N points by tracing a helix of *revs* revolutions in the direction of the cylinder's axis, centere on the point specifying the cylinder's line parameter, and of height the parent cylinder's height parameter..

C# example:

```
Point3DArray points_fitted = cylinder.CreatePointArray(A.Nrows,6.0);
```

12 Cone3D class

A cone in 3D space is defined by a line describing its axis, together with its radius at a prescribed point on the axis, together with the semi-angle. See Figure 13.

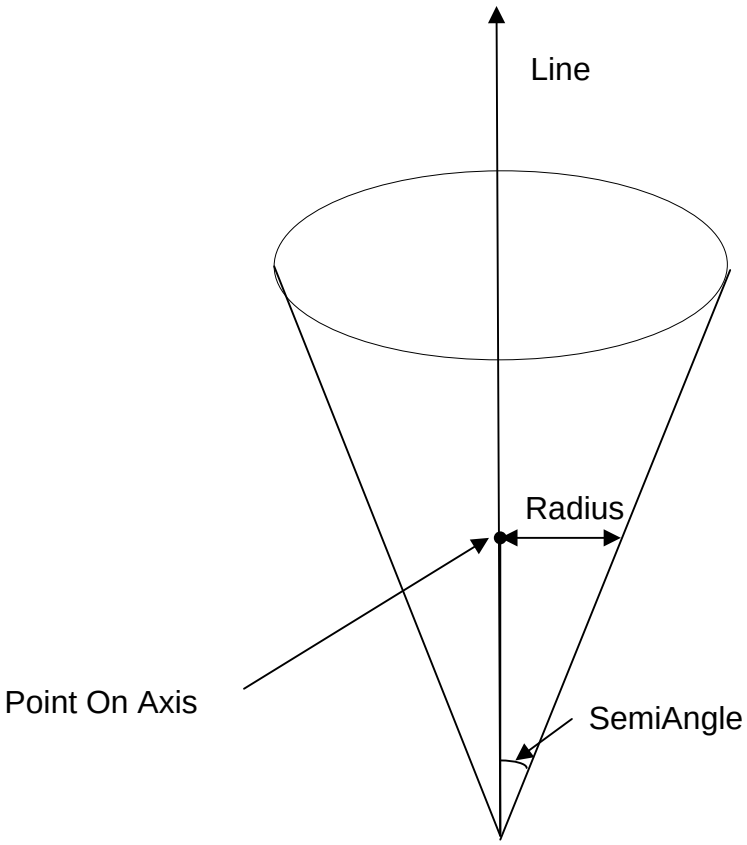


Figure 13 - Parameters of a cone.

12.1 Constructors

12.1.1 Cone3D()

Initializes a cone with zero radius and semi-angle and an axis passing through (0,0,0) in the direction of the vector (0,0,1);

12.1.2 Cone3D(Cone3D *cone*)

Initializes a cylinder with radius, semi-angle and axis that of *cone*.

12.1.3 Cone3D(Line3D *line*,double R, double α)

Initializes a cone with radius R, line *line* and semi-angle α. α is in radians.

C# example:

```
Cone3D cone = new Cone3D(line,10.0,5.0);
```

12.2 Properties

12.2.1 double Radius

Gets and sets the radius of the cone.

12.2.2 Line3D Line

Gets and sets the axis of the cone. The axis is pointing away from the apex of the cone.

12.2.3 double RMSError

Gets the average of the square root of the sum of the errors: $\sqrt{\frac{\sum_{i=1}^{N} e_i^2}{N}}$, where N is the number of points and e_i is the distance from point i to the best fit cone.

12.2.4 double MaxError

Gets $\max_{i=1}^{N}\{e_i\}$ - the maximum of the distances of point i to the best fit cone.

12.2.5 double Height

Returns the distance along the cone's axis between the point representing one extreme of the data set and the point representing the other extreme.

12.2.6 Double SemiAngle

Gets and sets the semi-angle of the cone in radians.

12.3 Methods

12.3.1 void Assign(Cone3D *cone*)

Sets the cone to *cone*.

12.3.2 DistanceTo(Point3D *pt*)

Returns the shortest distance from the cone to the point.

12.3.3 int Fit(Point3DArray *pts*, int *type*)

12.3.4 int Fit(Point3DArray *pts*, Cone3D *nom*)

12.3.5 int Fit(Point3DArray *pts*, Vector *nomAxis*)

12.3.6 int Fit(Point3DArray *pts*, Vector *nomAxis*, double *nomAngle*)

12.3.7 int Fit(PointArray3D *pts*, Vector *nomAxis*, double *nomRadius*, double *nomAngle*)

12.3.8 int Fit(PointArray3D *pts*, Line3D *nom*, double *nomAngle*)

Computes the best fitting cone (in a least squares sense) for the set of points.

12.3.9 Point3DArray CreatePointArray(int *N*, double *revs*)

Creates a series of N points by tracing a helix of *revs* revolutions in the direction of the cone's axis, centered on the point specifying the parent cone's line parameter. The axial height traversed will be the height of the parent cone.

C# example:

```
Point3DArray points_fitted = cone.CreatePointArray(A.Nrows,6.0);
```

13 References

[1] Matrix Computations (2nd Edition), Gene H Golub, Charles F. Van Loan, The John Hopkins University Press, 1989.

14 Contact

Paul D. Foy – paulfoy@mathematicalservices.co.uk