The modelling of helices

PUBLISHER

Loughborough University of Technology

REPOSITORY RECORD

East, Martin. 2021. “The Modelling of Helices”. Loughborough University. https://doi.org/10.26174/thesis.lboro.14884011.v1.

The Modelling of Helices

by

Martin East

A Master's project report submitted in partial fulfilment of the requirements for the award of MSc. in Computer Integrated Engineering of the Loughborough University of Technology, September 1988.

Academic Supervisor :	H. W. F. English, BSc, PhD, Loughborough University of Technology.
Industrial Supervisor :	F. N. Rigby, BSc, PhD, Pafec Ltd.

Acknowledgements

I am very grateful for the help and support I have had from many people throughout the last year. Without their assistance I would probably not have been able to complete the course. I would particularly like to thank the following (in no particular order) :-

> Dr. Neil Rigby and Dr. Y. T. Lee of Pafec Ltd., Dr. Alan Ball, Dr. Howard English, my friends and fellow students, Hercules, and my family, without whom none of this would have been possible.

Contents :-

Chapter 1 : Introduction

1.1 A Brief Introduction to Solid Modelling 1
1.2 Aims of Project 2

Chapter 2 : Defining a Helix

2.1 Parametric Equation of Surface 3
2.2 Cartesian Equation of Surface 4

Chapter 3 : Interrogating the Helix

3.1 Calculation of the Normal to a Helix 7
    3.1.1 Parametric Representation
    3.1.2 Using the Cartesian Equation of the Surface
3.2 Closest Approach of a Plane to a Helix 8
3.3 Bounding Box of a Helix 9
3.4 Finding the Parameters of a Point on a Helix 10
    3.4.1 Finding Θ
    3.4.2 Finding ø
3.5 Intersection Between a Line and a Helix 12
    3.5.1 Using Cartesian Equation of the Surface
    3.5.2 Using the Parametric Equation of the Surface
3.6 Intersection Between a Plane and a Helix 13
    3.6.1 Special Cases
    3.6.2 General Case
3.7 Intersection of a Helix with a General Quadric Surface 15

Cpapter 4 : Two Computer Programs to Calculate Intersections

4.1 Program to Calculate the Intersection Between a Line and a Helix 17
4.2 Program to Calculate the Intersection Between a Plane and a Helix 18

Chapter 5 : Future Work and Conclusion
5.1 Future Work 20
5.2 Conclusion 20

Appendix 1 : Program to Calculate the Intersection Between a Line and a Helix
Appendix 2 : Program to Calculate the Intersection Between a Plane and a Helix
Appendix 3 : Suggested Reading

Chapter 1

Introduction

1.1 A Brief Introduction to Solid Modelling

The basic idea of solid modelling is to represent solids using a computer in such a way that the solid can be

(i) visualised - usually on a screen but possibly plotted on paper

(ii) interrogated - intersects with other solids and surfaces need to be found, normals calculated, etc.

(iii) modified

(iv) manufactured - tool paths of CNC machines or robotics control need to be calculated.

It is common to build up a solid by the use of Boolean operations (union, intersection or difference) on primitive solids. Such a solid modeller will have several different primitives (for example sphere, cuboid, cone, torus, etc.) and each time one of these primitives is specified it will be given certain parameters to uniquely define it, for example a sphere is uniquely defined by its position in space and its radius.

Any model must be amenable to interrogation. Many calculations need to be made for the model to be displayed, manufactured by NC machines or combined with other solids to form more complex bodies. These include :-

(i) The normal to the surface, which is needed for numerous operations including ray tracing (for displaying the model), calculating NC machine offsets, determining silhouette curves, shading and hidden surface removal.

(ii) The intersection of a line with the model is similarly important in ray tracing and display calculations and is also needed in Boolean operations with other solids since a vertex of the new solid will be formed where the edge of a solid meets the surface of the model.

(iii) The curves of intersection of the model with a plane (or more complex surface) need to be calculated to work out the edges of the solid formed by a Boolean operation on the model with some other solid bounded by the surface.

1.2 Aims of the Project

The basic aim of this project is to investigate the mathematics of helical surfaces to determine if it is possible to incorporate helices in a solid modeller. It is thought that one area where helices may be useful is in the modelling of coil springs.

Chapter 2

Defining a Helix

Unless otherwise stated all helices referred to are assumed to have a circular cross-section and be right handed. The extension to left handed helices is trivial.

For reference the parametric equation of a right handed helical line is

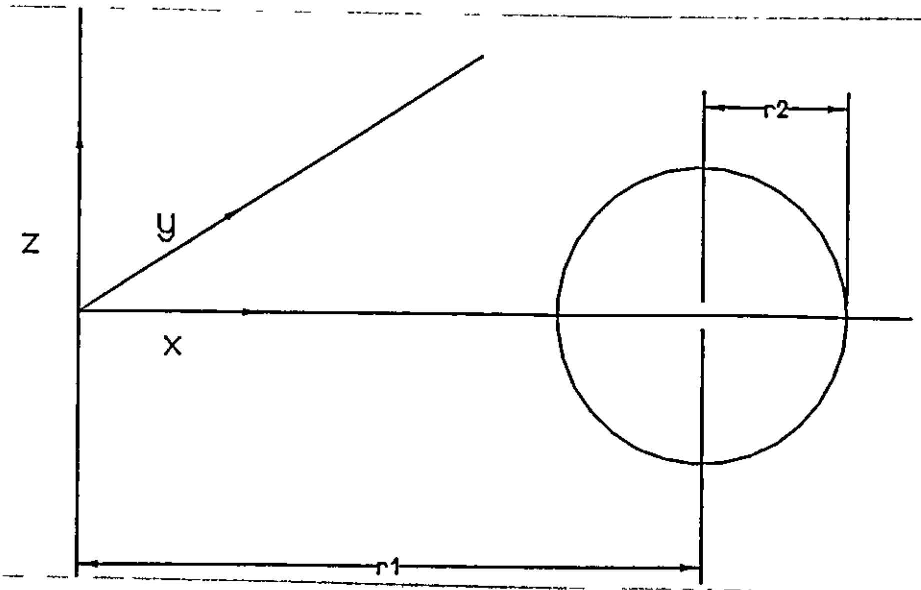
$$\underline{r}(t) = \begin{bmatrix} r\cos t \\ r\sin t \\ kt \end{bmatrix}$$

and that of a left handed helical line is

$$\underline{r}(t) = \begin{bmatrix} r\cos t \\ -r\sin t \\ kt \end{bmatrix}$$

2.1 Parametric Equation of Surface

Consider a circle of radius r_2 in the xz plane whose centre is a distance r_1 (where $r_1 > r_2$) from the z axis. Rotating this circle about the z axis and simultaneously applying a translation in the z direction generates the helical surface.



The circle is defined parametrically by

$$\underline{R} = \begin{bmatrix} r_1 + r_2 \cos \phi \\ 0 \\ r_2 \sin \phi \end{bmatrix} \qquad 0 \le \phi \le 2\pi$$

Applying the matrix transformations for the sweep along the path of a right handed helical line gives the surface of the helix as

$$\underline{R} = \begin{bmatrix} (r_1 + r_2 \cos \phi) \cos \theta \\ (r_1 + r_2 \cos \phi) \sin \theta \\ r_2 \sin \phi + k\theta \end{bmatrix} \qquad (1)$$

where $0 \le \phi \le 2\pi$ and, for a helix given by n rotations of the circle about the z axis, $0 \le \theta \le 2n\pi$.

2.2 Cartesian Equation of the Surface

From (1) above we have :-

$$x = (r_1 + r_2 \cos \phi) \cos \theta \qquad (2)$$

$$y = (r_1 + r_2 \cos \phi) \sin \theta \qquad (3)$$

$$z = r_2 \sin \phi + k\theta \qquad (4)$$

The cartesian equation of the surface is found by eliminating the parameters θ and ϕ from these three equations.

Squaring equations (2) and (3) and adding them gives

$$x^2 + y^2 = (r_1 + r_2 \cos \phi)^2$$

$$\Rightarrow \quad \cos \phi = \frac{\pm (x^2 + y^2)^{1/2} - r_1}{r_2} \qquad (5)$$

If we take the negative root then, since $r_1 > r_2$, we would have $\cos \phi < -1$. Thus we only take the positive root in (5).

$$\text{Hence} \quad \cos \phi = \frac{(x^2 + y^2)^{1/2} - r_1}{r_2} \qquad (6)$$

Chapter 2

Defining a Helix

Unless otherwise stated all helices referred to are assumed to have a circular cross-section and be right handed. The extension to left handed helices is trivial.

For reference the parametric equation of a right handed helical line is

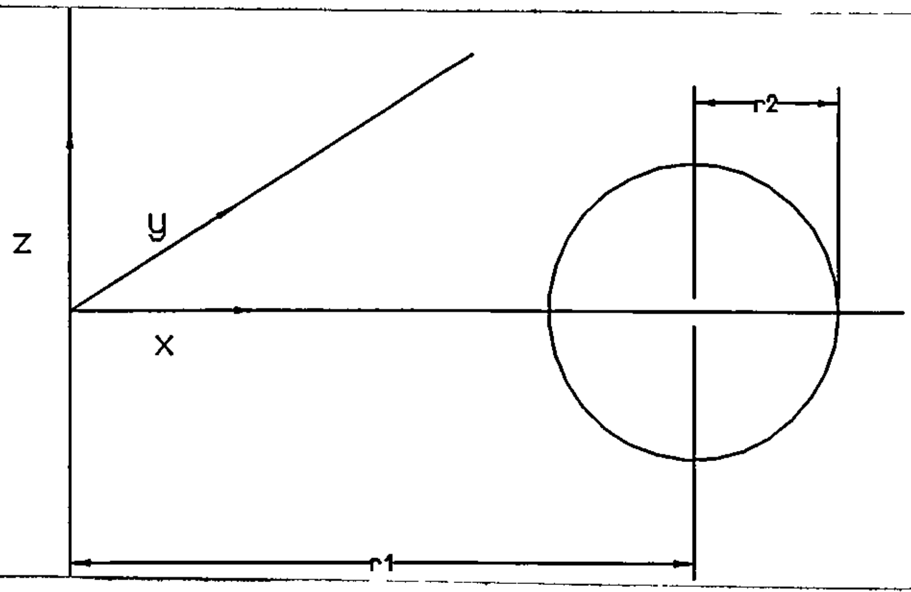
$$\underline{r}(t) = \begin{bmatrix} r\cos t \\ r\sin t \\ kt \end{bmatrix}.$$

and that of a left handed helical line is

$$\underline{r}(t) = \begin{bmatrix} r\cos t \\ -r\sin t \\ kt \end{bmatrix}$$

2.1 Parametric Equation of Surface

Consider a circle of radius r_2 in the xz plane whose centre is a distance r_1 (where $r_1 > r_2$) from the z axis. Rotating this circle about the z axis and simultaneously applying a translation in the z direction generates the helical surface.



and so

$$\sin \phi = \pm \sqrt{1 - \left[\frac{(x^2 + y^2)^{1/2} - r_1}{r_2} \right]^2}$$

taking the positive root for $0 \leq \phi \leq \pi$ and the negative root for $\pi \leq \phi \leq 2\pi$

$$\Rightarrow \quad \sin \phi = \pm \frac{1}{r_2} \left[2r_1 (x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2 \right]^{1/2} \text{———} (7)$$

Dividing (3) by (2) gives

$$\frac{y}{x} = \tan \Theta$$

and from (4)

$$\Theta = \frac{z - r_2 \sin \phi}{k}$$

thus

$$\frac{y}{x} = \tan \frac{z - r_2 \sin \phi}{k}$$

Now, substituting for sin ø from (7) gives

$$\frac{y}{x} = \tan \frac{z \pm \left[2r_1 (x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2 \right]^{1/2}}{k}$$

$$\Rightarrow \quad y - x \tan \frac{z \pm \left[2r_1 (x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2 \right]^{1/2}}{k} = 0 \text{———} (8)$$

which appears to be the equation of the surface of a helix. Unfortunately this is not the case.

It can be seen that if a point (x_1, y_1, z_1) satisfies equation (8) then the point $(-x_1, -y_1, z_1)$ will also satisfy it. This shows that (8) does not give the equation for the surface of a helix but the equation for the surface of a double helix. This, of course, means that when dealing with a helix it is not possible to deal solely with the cartesian equation of the surface.

Additional complications arise because of the ± in the definition. The helix is, in effect, made up of two surfaces, namely

$$y - x \tan \frac{z + \left[2r_1 (x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{1/2}}{k} = 0 \text{ ——— } (9)$$

corresponding to $\pi \leq \text{ø} \leq 2\pi$, and

$$y - x \tan \frac{z - \left[2r_1 (x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{1/2}}{k} = 0 \text{ ——— } (10)$$

corresponding to $0 \leq \text{ø} \leq \pi$.

Chapter 3

Interrogating the Helix

3.1 Calculation of the Normal to a Helix

3.1.1 Parametric Representation

The standard method of finding the normal to a parametric surface is to partially differentiate the parametric equation of the surface with respect to each of the parameters and then work out the cross product of the partial derivatives. These partial derivatives are easily calculated for the helix.

The surface of the helix is given by

$$\underline{s} = \begin{bmatrix} (r_1 + r_2 \cos \phi) \cos \theta \\ (r_1 + r_2 \cos \phi) \sin \theta \\ r_2 \sin \phi + k\theta \end{bmatrix}$$

so

$$\frac{\partial \underline{s}}{\partial \theta} = \begin{bmatrix} -(r_1 + r_2 \cos \phi) \sin \theta \\ (r_1 + r_2 \cos \phi) \cos \theta \\ k \end{bmatrix}$$

and

$$\frac{\partial \underline{s}}{\partial \phi} = \begin{bmatrix} -r_2 \sin \phi \cos \theta \\ -r_2 \sin \phi \sin \theta \\ r_2 \cos \phi \end{bmatrix}$$

and so $\underline{n} = \frac{\partial \underline{s}}{\partial \theta} \times \frac{\partial \underline{s}}{\partial \phi}$ is given by

$$\underline{n} = \begin{bmatrix} r_2 \cos \phi \cos \theta \, (r_1 + r_2 \cos \phi) + k r_2 \sin \phi \sin \theta \\ r_2 \cos \phi \sin \theta \, (r_1 + r_2 \cos \phi) - k r_2 \sin \phi \cos \theta \\ r_2 \sin \phi \, (r_1 + r_2 \cos \phi) \end{bmatrix} \quad \text{———} (11)$$

3.1.2 Using the Cartesian Equation of the Surface

For a surface with cartesian equation f (x, y, z) = 0 the surface normal direction is given by the vector $(\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)$.

Differentiation of the equation of the surface yields :-

$$\frac{\partial f}{\partial x} = -\tan \frac{z \pm \left[2r_1(x^2+y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{1/2}}{k}$$

$$-\frac{x}{k}\sec^2\left(\left\{z \pm \left[2r_1(x^2+y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{1/2}\right\}/k\right)$$

$$.\left[2r_1(x^2+y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{-1/2}\left[xr_1(x^2+y^2)^{-1/2} - 2x\right]$$

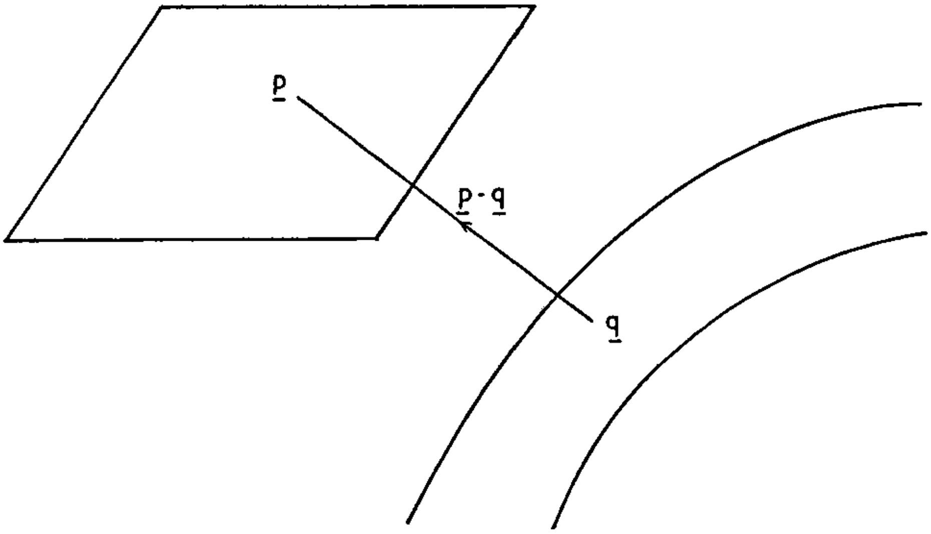
$$\frac{\partial f}{\partial y} = 1 - \frac{x}{k}\sec^2\left(\left\{z \pm \left[2r_1(x^2+y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{1/2}\right\}/k\right)$$

$$.\left[2r_1(x^2+y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{-1/2}\left[yr_1(x^2+y^2)^{-1/2} - 2y\right]$$

$$\frac{\partial f}{\partial z} = -x\sec^2\left(\left\{z \pm \left[2r_1(x^2+y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2\right]^{1/2}\right\}/k\right)$$

Obviously it is much more difficult to calculate the normal using the cartesian equation than using the parametric form of the surface.

3.2 Closest Approach of a Plane to a Helix



The closest approach of a plane to a helix occurs when the vector $\underline{p} - \underline{q}$ has a minimal value. When this happens $(\underline{p} - \underline{q})$ is perpendicular to both the plane and the surface of the helix. If $\underline{n}$ and $\underline{m}$ are the normal vectors to the plane and helix respectively then we get the following equations

$$(\underline{p}-\underline{q}) \times \underline{n} = 0$$

and $$(\underline{p}-\underline{q}) \times \underline{m} = 0$$

We also have the equation of the plane to be satisfied by $\underline{p}$

$$\underline{p} \cdot \underline{n} + d = 0$$

This gives us seven simultaneous equations for the five unknowns p_1, p_2, p_3, θ and $\varnothing$. Although this is theoretically soluble it turns out that the equations are excessively complicated and it would be far better to avoid having to solve such equations altogether.

3.3 Bounding Box of a Helix

Bounding boxes are boxes, usually cuboid, that enclose solids and are used to give a rough indication of the boundaries of the solid to assist in the detection of intersections and interference.

In order to find the bounding box of a helix it is necessaryto calculate the closest approach of several planes to the helix. As we have seen this is very difficult to do and best avoided. However, a bounding box can be constructed around the helix by using the fact that a helix is bounded by a cylinder.

For a helix given parametrically by

$$\underline{s} = \begin{bmatrix} (r_1 + r_2 \cos \varnothing) \cos \theta \\ (r_1 + r_2 \cos \varnothing) \sin \theta \\ r_2 \sin \varnothing + k\theta \end{bmatrix} \qquad 0 \le \varnothing \le 2\pi,\ 0 \le \Theta \le 2n\pi$$

the helix is contained inside the cylinder

$$\{ (x, y, z) : x^2 + y^2 \le (r_1 + r_2)^2,\ -r_2 \le z \le 2kn\pi + r_2 \} \text{ ———— } (12)$$

and lies outside the cylinder

$$\{ (x, y, z) : x^2 + y^2 < (r_1 - r_2)^2 \} \text{ ———————— } (13)$$

Where the bounding box of a cylinder is required it makes more sense to deduce the bounding cylinder from (12) and then work out the bounding box of this bounding cylinder. This is much simpler and in most cases would give either an identical answer or a very similar answer.

3.4 Finding the Parameters of a Point on a Helix

Given a point on a helix it is important to be able to find the parameters corresponding to that point. The following technique can be used to

(i) find the parameters of a point on a helix, or

(ii) given a point in space, determine whether or not the point is on the helix, finding the parameters if and only if the point is on the helix.

The location of a pont on the surface of a helix is determined by two parameters, Θ and $\varnothing$. $\varnothing$ is used to determine a point on the (circular) cross-section of the helix and Θ is used to give the angle of rotation of that section around the axis of the helix. The determination of parameters is made slightly more complicated by the fact that Θ may vary over a range of more than 2π, but in practice this does not present any insurmountable problems.

The problem is broken down into the following steps :-

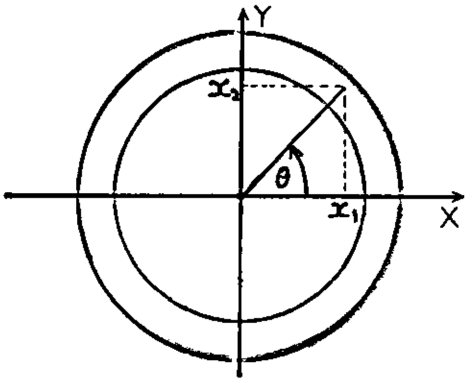
1. Find Θ . If Θ cannot be found then the point is not on the helix.
2. If Θ is found then try to find $\varnothing$. If $\varnothing$ cannot be found then the point is not on the helix.

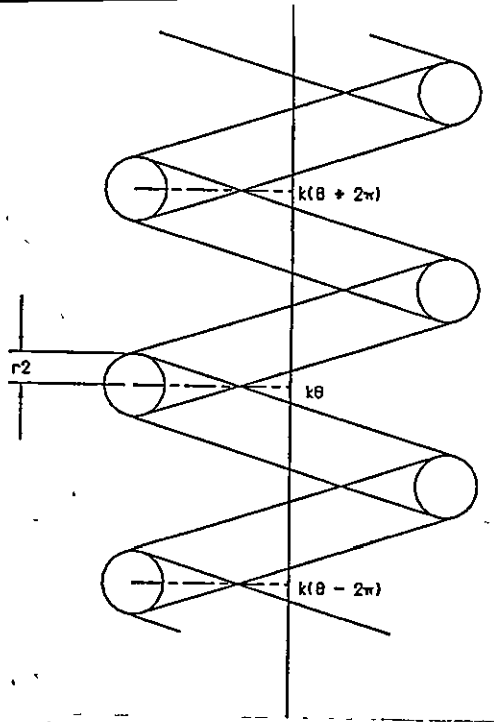
3.4.1 Finding Θ

Let $\underline{x} = (x_1, x_2, x_3)$ be the co-ordinates of a point (not necessarily on the helix). As an initial guess for Θ work out Θ_1 such that $0 \leq \Theta_1 \leq 2\pi$, and

$$\cos \Theta_1 = x_1 / (x_1^2 + x_2^2)^{1/2} \ , \ \sin \Theta_1 = x_2 / (x_1^2 + x_2^2)^{1/2}$$

(see diagram)





There are now three cases to consider :-

1. If $k\theta_1 - r_2 \le x_3 \le k\theta_1 + r_2$ then this is the correct value of θ.

2. If $x_3 > k\theta_1 + r_2$ then as a next guess try $\theta_2 = \theta_1 + 2\pi$ and continue increasing the guesses at θ by 2π until either

(a) $k\theta_i - r_2 \le x_3 \le k\theta_i + r_2$, in which case θ_i is the correct value of θ

or (b) $k\theta_{i-1} + r_2 < x_3 < k\theta_i - r_2$, in which case no correct value of can be found and the point is not on the helix.

3. If $x_3 < k\theta_1 - r_2$ then as a next guess try $\theta_2 = \theta_1 - 2\pi$ and continue decreasing the guesses at θ by 2π until either

(a) $k\theta_i - r_2 \le x_3 \le k\theta_i + r_2$, in which case θ_i is the correct value of θ

or (b) $k\theta_i + r_2 < x_3 < k\theta_{i-1} - r_2$, in which case no correct value of can be found and the point is not on the helix.

3.4.2 Finding ø

Given that θ has now been found it still remains to find ø. We know from equation (6) earlier that

$$\cos \phi = \frac{(x_1^2 + x_2^2)^{1/2} - r_1}{r_2} \qquad (14)$$

and also $x_3 = r_2 \sin \phi + k\theta$

Hence

$$\sin ø = \frac{x_3 - k\theta}{r_2} \qquad \text{———————— (15)}$$

If (14) and (15) are consistent then they give us ø. To be consistent they must satisfy

$$\cos^2 ø + \sin^2 ø = 1$$

i.e. $$\left[\frac{(x_1^2 + x_2^2)^{1/2} - r_1}{r_2}\right]^2 + \left[\frac{x_3 - k\theta}{r_2}\right]^2 = 1$$

=> $$\left[(x_1^2 + x_2^2)^{1/2} - r_1\right]^2 + \left[x_3 - k\theta\right]^2 = r_2^2$$

If the equations are consistent then the point is on the helix and ø is found from equations (14) and (15). If the equations are not consistent then the point is not on the helix.

3.5 Intersection between a Line and a Helix

3.5.1 Using Cartesian Equation of the Surface

Consider the parametric equation of a line in vector form

$$\underline{r} = \underline{s} + \lambda \underline{t} = (s_1 + \lambda t_1,\ s_2 + \lambda t_2,\ s_3 + \lambda t_3) \qquad \text{———— (16)}$$

and the cartesian equations of the surface of a double helix

$$y - x\tan\frac{z + [2r_1(x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2]^{1/2}}{k} = 0 \qquad \text{———— (17a)}$$

and

$$y - x\tan\frac{z - [2r_1(x^2 + y^2)^{1/2} + r_2^2 - r_1^2 - x^2 - y^2]^{1/2}}{k} = 0 \qquad \text{———— (17b)}$$

To find the points of intersection of a line and a helix substitute (16) into (17a) and (17b). This gives the two equations

$$(s_2 + \lambda t_2) - (s_1 + \lambda t_1)\tan(\{s_3 + \lambda t_3 + [2r_1(A + \lambda B + \lambda^2 C)^{1/2} + r_2^2 - r_1^2$$
$$- A - \lambda B - \lambda^2 C]^{1/2}\} / k) = 0 \qquad \text{———————— (18a)}$$

and

$$(s_2 + \lambda t_2) - (s_1 + \lambda t_1)\tan(\{s_3 + \lambda t_3 - [2r_1(A + \lambda B + \lambda^2 C)^{1/2} + r_2^2 - r_1^2 - A - \lambda B - \lambda^2 C]^{1/2}\}/k) = 0 \quad \text{———————} \quad (18b)$$

where $A = s_1^2 + s_2^2$, $B = 2(s_1 t_1 + s_2 t_2)$ and $C = t_1^2 + t_2^2$.

These two equations must both be solved to give all the points of intersection between a line and a double helix. When all the roots of (18a) and (18b) have been found then these roots must be checked to see which are intersections of the line with the "phantom" helix. This can be done by trying to find the parameters, ø and Θ , of the point on the helical surface. If these parameters can be found then the point is on the "real" helix, if not then the point is on the "phantom" helix.

A program was written to calculate the intersection between a line and a helix and is discussed in the next chapter.

3.5.2 Using the Parametric Equation of the Surface

The parametric form of a line is given in (16) and that of the helix in (1). Where the line intersects the helix the equations (1) and (16) are equal. Thus we get :-

$$s_1 + \lambda t_1 = (r_1 + r_2\cos\phi)\cos\theta \quad \text{———————} \quad (19)$$

$$s_2 + \lambda t_2 = (r_1 + r_2\cos\phi)\sin\theta \quad \text{———————} \quad (20)$$

$$s_3 + \lambda t_3 = r_2\sin\phi + k\theta \quad \text{———————} \quad (21)$$

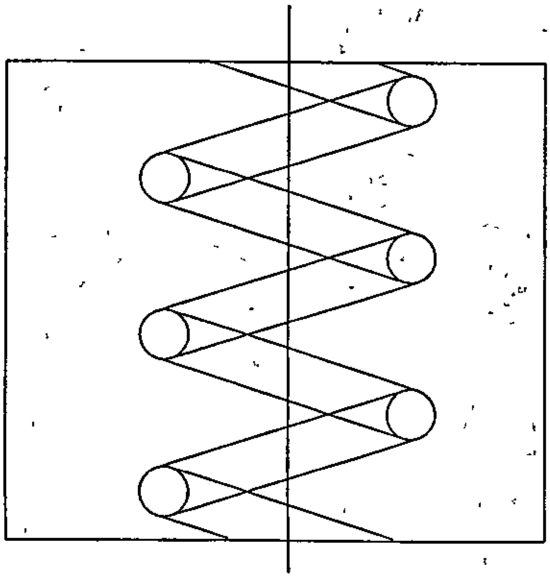
Hence we need to solve this set of three simultaneous equations in three unknowns. Given k, r_1 , r_2 , s_1 , s_2 , s_3 , t_1 , t_2 , t_3 we have to find λ , Θ and ø.

3.6 Intersection between a Plane and a Helix

3.6.1 Special Cases

There are two special cases to consider :-

(a) When the plane contains the axis of the helix



In this case the curves of intersection are circles.

(b) z = constant plane.

This plane intersection may be needed to work out the end faces of a coil spring.

From the parametric equation of the helix

$$\underline{s} = \begin{bmatrix} (r_1 + r_2 \cos \phi) \cos \theta \\ (r_1 + r_2 \cos \phi) \sin \theta \\ r_2 \sin \phi + k\theta \end{bmatrix} \qquad 0 \leq \phi \leq 2\pi,\ 0 \leq \theta \leq 2n\pi$$

Putting z = constant (c, say) , we get

$$r_2 \sin \phi + k\theta = c$$

$$\Rightarrow \qquad \theta = \frac{c - r_2 \sin \phi}{k}$$

and , substituting this expression for θ in the equation of the surface we find that the curve of intersection is given parametrically by

$$x(\phi) = (r_1 + r_2 \cos \phi) \cos \frac{c - r_2 \sin \phi}{k} \qquad 0 \leq \phi \leq 2\pi$$

$$y(\phi) = (r_1 + r_2 \cos \phi) \sin \frac{c - r_2 \sin \phi}{k}$$

3.6.2 General Case

The implicit form of the equation of a plane (in vector notation) is

$$\underline{a} \cdot \underline{n} + d = 0 \quad \text{———————} \quad (22)$$

Where $\underline{a}$ is the position vector of a point on the plane and $\underline{n}$ is the normal to the plane.

The intersection of this plane with the helix

$$\underline{s} = \begin{bmatrix} (r_1 + r_2 \cos \phi) \cos \theta \\ (r_1 + r_2 \cos \phi) \sin \theta \\ r_2 \sin \phi + k\theta \end{bmatrix}$$

is obtained by substituting $\underline{s}$ for $\underline{a}$ in (22) and solving the resulting equation. So, substituting $\underline{s}$ into (22) we get

$$n_1 \ (r_1 + r_2 \cos \phi) \cos\theta + n_2 \ (r_1 + r_2 \cos \phi) \sin\theta + n_3 \ (r_2 \sin \phi + k\theta) + d = 0 \qquad (23)$$

For any given value of θ , equation (23) is of the form

$$K_1 (r_1 + r_2 \cos \phi) + K_2 \sin \phi + K_3 = 0$$

or

$$K_4 + K_5 \cos \phi + K_2 \sin \phi = 0 \quad \text{———————} \quad (24)$$

This equation in ϕ can be solved by using the substitution

$$t = \tan \phi /2$$

Then by using a range of values of θ in (23) , equation (24) will give a series of points which can be interpolated to give the curve of intersection.

A program was written to calculate the intersection of a plane and a helix using the above method and is discussed in the next chapter.

3.7 Intersection of a Helix with a General Quadric Surface

The implicit equation of a general quadric surface is

$$A x^2 + B y^2 + C z^2 + 2D xy + 2E yz + 2F xz + 2G x + 2H y + 2J z + K = 0 \quad \text{——} (25)$$

It is possible to find the curve of intersection of a helix with a general quadric surface by using a technique similar to that for finding the curve of intersection of a helix with a general plane. By substituting

$$x = (r_1 + r_2 \cos \phi) \cos \theta$$

$$y = (r_1 + r_2 \cos \phi) \sin \theta$$

$$z = r_2 \sin \phi + k\theta$$

into (25) , the following equation is obtained :-

$$A\,(r_1^2 + 2\,r_1\,r_2 \cos\phi + r_2^{\,2} \cos^2\phi)\cos^2\theta$$

$$+\,B\,(r_1^2 + 2\,r_1\,r_2 \cos\phi + r_2^{\,2} \cos^2\phi)\sin^2\theta$$

$$+\,C\,(k^2\,\theta^2 + 2k\,r_2\,\theta \sin\phi + r_2^{\,2} \sin^2\phi)$$

$$+\,2D\,(r_1^2 + 2\,r_1\,r_2 \cos\phi + r_2^{\,2} \cos^2\phi)\sin\theta\,\cos\theta$$

$$+\,2E\,(r_1\,r_2 \sin\phi + r_2^{\,2} \cos\phi \sin\phi + kr_1\theta + k\theta\,r_2 \cos\phi)\sin\theta$$

$$+\,2F\,(r_1\,r_2 \sin\phi + r_2^{\,2} \cos\phi \sin\phi + kr_1\theta + k\theta\,r_2 \cos\phi)\cos\theta$$

$$+\,2G\,(r_1 + r_2 \cos\phi)\cos\theta$$

$$+\,2H\,(r_1 + r_2 \cos\phi)\sin\theta$$

$$+\,2J\,(r_2 \sin\phi + k\theta)$$

$$+\,K = 0$$

or , on re-arranging

$$\cos^2\phi\,(A\,r_2^{\,2} \cos^2\theta + B\,r_2^{\,2} \sin^2\theta + 2D\,r_2^{\,2} \sin\theta \cos\theta)$$

$$+\sin^2\phi\,(C\,r_2^{\,2})$$

$$+\sin\phi\cos\phi\,(2E\,r_2^{\,2} \sin\theta + 2F\,r_2^{\,2} \cos\theta)$$

$$+\cos\phi\,(2A\,r_1\,r_2 \cos^2\theta + 2B\,r_1\,r_2 \sin^2\theta + 4D\,r_1\,r_2 \sin\theta\,\cos\theta$$

$$+\,2E\,r_2\,k\theta \sin\theta + 2F\,r_2\,k\theta \cos\theta + 2G\,r_2 \cos\theta + 2H\,r_2 \sin\theta)$$

$$+\sin\phi\,(C\,r_2\,k\theta + 2E\,r_1\,r_2 \sin\theta + 2F\,r_1\,r_2 \cos\theta + 2J\,r_2)$$

$$+\,(A\,r_1^{\,2} \cos^2\theta + B\,r_1^{\,2} \sin^2\theta + C\,k^2\,\theta^2 + 2D\,r_1^{\,2} \sin\theta \cos\theta$$

$$+\,2E\,r_1\,k\theta \sin\theta + 2F\,r_1\,k\theta \cos\theta + 2G\,r_1 \cos\theta + 2H\,r_1 \sin\theta$$

$$+\,2J\,k\theta + K) = 0$$

which , for any given θ is of the form

$$L\cos^2\phi + M\sin^2\phi + N\sin\phi\cos\phi + P\cos\phi + Q\sin\phi + R = 0$$

As before, this can be solved by a substitution of $t = \tan \phi/2$. In this case a quartic equation results, so for each θ there are four possible points of intersection. Again, the problem can be solved for a range of values of θ and the resulting points interpolated to give the curve of intersection.

Chapter 4

Two Computer Programs to Calculate Intersections

Two computer programs were written to see if in practice the intersections between a helix and a plane or a line could be worked out.

4.1 Program to Calculate the Intersection Between a Line and a Helix

The program uses the cartesian equations of the surface of a double helix and the parametric equation of a line to find the points of intersection. This approach was selected to avoid having to solve simultaneous equations. The main advantage appeared to be that by solving equations in just one variable, lambda, it would be much easier to ensure that all intersections between the line and the helix were found. This is very important. However this approach does have its drawbacks, chiefly :-

(i) In order to check whether a point is on the surface, two complicated functions must be evaluated.

(ii) After finding the points on the line that satisfy the cartesian equations of the helix, they must all be rechecked to determine which are on the "phantom" helix. It would be much better to have an approach that did not generate these "phantom" intersections at all.

(iii) This very roundabout way of calculating points of intersection leads to an enormous and very cumbersome program containing many procedures and functions. This greater complexity increases the chances of "bugs" creeping in, and indeed, the program as presented here does not work as there was not sufficient time to finish debugging it.

Two possible alternative approaches are :-

(i) As it is relatively easy to find the intersect of a helix with a plane, first find the curve of intersection of the helix with a plane containing the line. Then intersect the line with the resulting curve of intersection.

(ii) The procedure get_parameters (see Appendix) may be slightly modified so that if it tries to find the parameters of a point not on the helix it will determine whether the point is inside or outside the helix. This can be done by calculating the values of $\cos^2 ø$ and $\sin^2 ø$ from equations (14) and (15) in section 3.4.2 . If these values sum to a value greater than 1 then the point is outside the helix. Successive points on the line can be checked to see where the line crosses from outside the helix to inside. Once the position of the root is approximately known it should be found quite quickly. A listing of the program and documentation for the program are given in Appendix 1.

4.2 Program to Calculate the Intersection Between a Plane and a Helix

This program finds the intersect by solving equation (23) from section 3.6.2. This equation is :-

$$n_1 (r_1 + r_2 \cos ø) \cos\theta + n_2 (r_1 + r_2 \cos ø) \sin\theta + n_3 (r_2 \sin ø + k\theta) + d = 0$$

and so, re-arranging :-

$$r_2 (n_1 \cos\theta + n_2 \sin\theta) \cos ø + n_3 r_2 \sin ø + n_1 r_1 \cos\theta + n_2 r_1 \sin\theta + kn_3\theta + d = 0 \qquad (26)$$

Let $t = \tan ø/2$, then $\cos ø = \dfrac{(1 - t^2)}{(1 + t^2)}$, $\sin ø = \dfrac{2t}{(1 + t^2)}$ and substitute into (26).

Hence

$$r_2 (n_1 \cos\theta + n_2 \sin\theta) \frac{(1 - t^2)}{(1 + t^2)} + \frac{n_3 r_2 2t}{(1 + t^2)} + n_1 r_1 \cos\theta + n_2 r_1 \sin\theta + kn_3\theta + d = 0$$

$$\Rightarrow (1 - t^2) r_2 (n_1 \cos\theta + n_2 \sin\theta) + 2n_3 r_2 t$$
$$+ (1 + t^2)(n_1 r_1 \cos\theta + n_2 r_1 \sin\theta + kn_3\theta + d) = 0$$

$$\Rightarrow t^2 \{ (r_1 - r_2)(n_1 \cos\theta + n_2 \sin\theta) + kn_3\theta + d \} + t \{ 2n_3 r_2 \}$$
$$+ \{ (r_1 + r_2)(n_1 \cos\theta + n_2 \sin\theta) + kn_3\theta + d \} = 0$$

This is simply a quadratic in t. For a given value of θ it can be solved. Hence t, and so ø can be found. Putting $k_1 = n_1 \cos\theta + n_2 \sin\theta$, $k_2 = kn_3\theta + d$, we get

$$t^2 \{ (r_1 - r_2) k_1 + k_2 \} + t \{ 2n_3 r_2 \} + \{ (r_1 + r_2) k_1 + k_2 \} = 0 \qquad (27)$$

which has roots

$$t = \frac{-n_3 r_2 \pm (n_3^2 r_2^2 - \{ (r_1 - r_2) k_1 + k_2 \} \{ (r_1 + r_2) k_1 + k_2 \})}{(r_1 - r_2) k_1 + k_2} \qquad (28)$$

Let the roots to (27) be t_1 and t_2. Cos ø and sin ø can be worked out for each of t_1 and t_2 and these values substituted into the parametric equation of the helix to give the points of intersection.

The following cases should be noted :-

(i) t_1 and t_2 complex for a given θ

This indicates that theplane does not intersect the helix for that value of θ.

(ii) $t_1 = t_2$

This occurs when

$n_3^2 r_2^2 - \{ (r_1 - r_2) k_1 + k_2 \} \{ (r_1 + r_2) k_1 + k_2 \} = 0$

and indicates that for that value of θ the plane just touches the helix.

(iii) t_1 and t_2 real and distinct

In this case the plane cuts the helix at two distinct points for that value of θ.

(iv) $\{ (r_1 - r_2) k_1 + k_2 \} = 0$ for some θ

When this occurs (27) reduces to

$t \{ 2n_3 r_2 \} + \{ (r_1 + r_2) k_1 + k_2 \} = 0$

$$\Rightarrow t = - \frac{\{ (r_1 + r_2) k_1 + k_2 \}}{\{ 2n_3 r_2 \}} \qquad (29)$$

This gives one root. The other root is, roughly speaking, at $t = \infty$. Since $t = \tan ø/2$ this means that $ø / 2 = \pi / 2, 3\pi / 2, 5\pi / 2, \ldots$

$\Rightarrow ø = \pi, 3\pi, 5\pi, \ldots$

But in this case $0 \leq ø \leq 2\pi$, and so we must have $ø = \pi$ as the missing root. So one value of ø is worked out from equation (29) and the other value is $ø = \pi$.

A listing of the program and documentation for the program appear in Appendix 2.

Chapter 5

Future Work and Conclusions

5.1 Future Work

It is obvious that there remains a lot of work to be done on this subject, and many areas must be the subject of future research before a modeller can include a helix amongst its primitives. These areas must include :-

1. The intersection of a helix with more general curves. Provided that the curves are described parametrically the approach used for intersecting a helix with a straight line could be generalised to cover other curves.
2. The intersection of a helix with a general plane or quadric surface can only be solved in terms of points rather than curves. If some method could be found to determine the actual curves of intersection (as opposed to calculating points on the curves and then interpolating the points) this would be a big step forward.
3. The methods so far used to solve line/helix intersections are very much "brute force" approaches. These are inefficient and some alternative method of finding the solutions to the equations should be sought.

5.2 Conclusions

It is safe to say that where helices are concerned nothing is simple. Indeed compared with the other primitives commonly used in solid modellers they are very complicated. While nothing was found to suggest that it would be impossible to incorporate the modelling of helices in a CAD system it was readily seen that the greater complexity of the equations describing helices means that more computing time would be needed to solve them and thus lead to a slower modeller.

Much more work remains to be done on the mathematics of helical surfaces and any system modelling them would, initially at least, have to be implemented in a limited form.

Appendix 1

Program to Calculate the Intersection Between a Line and a Helix

All the procedures in this program use the following Ada packages :-

pafec_base_types
pafec_standard_types
pafec_int_io
pafec_double_io
text_io
pafec_maths

The programs were written in Ada (ANSI / MIL-STD 1815A) and run on Apollo DN3000 and DN4000 workstations.

The following procedures were used in the program :-

A1.1 Procedure line_intersect_helix

1. Purpose

To calculate points of intersection between a straight line and a helix.

2. Keywords

Line, helix, intersection, bounding cylinder.

3. Language

Ada. ANSI / MIL - STD 1815A

The following data types are declared :-

three_vec is array(1..3) of double.
four_vec is array(1..4) of double.

4. Description

Line_intersect_helix calls the procedures get_line and get_helix_implicit to read in data specifying a line and a helix. The line is checked to see if it is vertical, and if it is vertical the procedure check_for _intersection_vertical is called to see if an intersection is possible. If an intersection is possible then procedure calculate_intersection_vertical works out the points of intersection.

If the line is not vertical then procedure intersect_bounding_cylinder is called. This works out if the line cuts the bounding cylinders, and if it does intersect the bounding cylinders it calculates the points of intersection, whether it cuts both inner and outer cylinders, whether it cuts just the outer cylinder or whether it just touches the outer cylinder. The sections of the line in between the outer and inner bounding cylinders of the helix are then checked to see if and where they intersect the helix itself.

5. Parameters

None.

6. Error Indicators

None.

7. Auxiliary Routines

Line_intersect_helix calls the following routines :-

- get_line
- get_helix_implicit
- check_for_intersection_vertical
- calculate_intersection_vertical
- intersect_bounding_cylinder
- check_point
- check_lambda_in _range

8. Accuracy

All arithmetic is carried out in double precision.

A1.2 Procedure get_line

1. Purpose

Accepts user input uniquely specifying a line.

2. Keywords

Point, direction

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The line is specified by two vectors. One giving a point on the line and the other giving the direction of the line.

5. Parameters

parameter	mode	type	description
s	out	three_vec	position vector of a point on the line.
t	out	three_vec	direction vector of line.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All variables are in double precision.

A1.3 Procedure get_helix_implicit

1. Purpose

Accepts user input uniquely specifying an infinitely long helix.

2. Keywords

Major radius, minor radius, pitch.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The helix is specified by its major and minor radii and its pitch.

5. Parameters

parameter	mode	type	description
major_radius	out	double	major radius of helix.
minor_radius	out	double	minor radius of helix.
pitch	out	double	pitch of helix divided by 2π.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All variables in double precision.

A1.4 Procedure_check_for_intersection_vertical

1. Purpose

Checks to see if a given vertical line intersects with a helix.

2. Keywords

Major radius, minor radius, intersection.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The position vector of one point on the line, s, is passed to the routine along with the major and minor radii of the helix. If the line lies between the bounding cylinders of the helix the logical flag intersection_possible returns the value true.

5. Parameters

parameter	mode	type	description
s	in	three_vec	position vector of a point on the vertical line.

major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
intersection_possible	out	boolean	returns true if the line lies between the bounding cylinders of the helix.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All variables in double precision.

A1.5 Procedure calculate_intersection_vertical

1. Purpose

Calculates points of intersection between a vertical line and a helix.

2. Keywords

Major radius, minor radius, theta, phi, pitch.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The procedure first calculates one value of the helical parameter theta where the line intersects the helix. The position vector of a point on the line, $\underline{s}$, is passed over by the calling routine. The x and y co-ordinates of the points of intersection are known, in this case they are the x and y components of the vector $\underline{s}$. So we can deduce a value for theta where an intersection occurs, say $_1$, with 0 ≤ $_1$ ≤ 2π. Furthermore we can deduce cos ø from equation (6). This gives two possible values for ø in the interval [0, 2π] and these two values of ø give the two points of intersection. All the other points of intersection occur at the same ø values and at theta values of $_1 \pm 2\pi$, $_1 \pm 4\pi$, etc.

5. Parameters

parameter	mode	type	description
s	in	three_vec	position vector of a point on the vertical line.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All arithmetic is carried out in double precision.

A1.6 Procedure Intersect_bounding_cylinder

1. Purpose

Calculates points of intersection (if any) between a line and the bounding cylinders of a helix.

2. Keywords

Inner cylinder, outer cylinder.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The procedure works by substituting the parametric equation of a line into the implicit equation of the cylinders and solving the resulting equations in lambda (the parameter of the line). The procedure first checks whether or not the line intersects the outer cylinder. There are three possible results :-

1. The line does not intersect the cylinder. In this case the logical flag intersection_possible is set to false.

2. The line is tangential to the cylinder. In this case the logical flag point_intersection is set to true and the root is calculated and stored as lambda(1).

3. The line intersects the cylinder at two distinct points. In this case the larger of the two roots is stored as lambda(1) and the smaller as lambda(2). The procedure then checks to see if the line intersects the inner cylinder as well. If the line intersects the inner cylinder then the logical flag intersect_inner_cyl is set to true and the roots are calculated and stored as lambda(3) and lambda(4).

5. Parameters

parameter	mode	type	description
s	in	three_vec	position vector of a point on the vertical line.
t	in	three_vec	direction vector of line
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
point_intersection	out	boolean	returns true if line is tangential to outer cylinder.
intersection_possible	out	boolean	returns true if line intersects outer cylinder.
intersect_inner_cyl	out	boolean	returns true if the line intersects the inner cylinder.
lambda	out	four_vec	returns the values of lambda where the line intersects the cylinders.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All arithmetic is carried out in double precision.

A1.7 Procedure check_point

1. Purpose

Checks whether a given point is on the surface of the helix.

2. Keywords

line, helix.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

Check_point uses the procedure get_parameters to determine if the point is on the helix. If the point is on the helix, the co-ordinates of the point are printed out.

5. Parameters

parameter	mode	type	description
s	in	three_vec	position vector of a point on the line.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.
lambda	in	double	value of the parameter of the line corresponding to the point to be checked.

6. Error Indicators

None.

7. Auxiliary Routines

Check_point calls the procedure get_parameters.

8. Accuracy

All arithmetic is carried out in double precision.

A1.8 Procedure_get_parameters

1. Purpose

To determine if a given point is on the helix and, if it is, determine the values of the helical parameters, theta and phi, corresponding to the point.

2. Keywords

theta, phi, point, helix.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The procedure first calls procedure try_and _find_theta. If theta cannot be found then the point is not on the helix. If theta is found then procedure try_and_find_phi is called. If phi cannot be found then the point is not on the helix. If the point is not on the helix then this is signalled to the calling routine by a logical flag, otherwise the values of theta and phi are passed back to the calling routine. For more information see section 3.4 of this report.

5. Parameters

parameter	mode	type	description
x	in	three_vec	position vector of point for which parameters are to be determined.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.
point_not_on_helix	out	boolean	returns true if the point is not on the helix.
theta	in out	double	parameter of helical surface.
phi	in out	double	parameter of helical surface.

6. Error Indicators

None.

7. Auxiliary Routines

Get_parameters calls the procedures try_and_find_theta and try_and_find_phi.

8. Accuracy

All variables are in double precision.

A1.9 Procedure try_and_find_theta

1. Purpose

To calculate the value of the helical parameter theta for a given point.

2. Keywords

theta, point, helix, pitch, major radius, minor radius.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

For a detailed description of the method of operation of this procedure see section 3.4.1 of this report.

5. Parameters

parameter	mode	type	description
x	in	three_vec	position vector of point for which the parameter theta is to be determined.
pitch	in	double	pitch of helix divided by 2π.
minor_radius	in	double	minor radius of helix.
theta	in out	double	parameter of helical surface.
theta_found	in out	boolean	returns true if theta is found.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All arithmetic is in double precision.

A1.10 Procedure try_and_find_phi

1. Purpose

To calculate the value of the helical parameter phi for a given point.

2. Keywords

theta, phi, point, helix, major radius, minor radius.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

For a detailed description of the method of operation of this procedure see section 3.4.2 of this report.

5. Parameters

parameter	mode	type	description
x	in	three_vec	position vector of point for which the parameter phi is to be determined.
theta	in	double	parameter of helical surface.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.
phi	out	double	parameter of helical surface.

phi_found	out	boolean	returns true if phi is found.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All arithmetic is in double precision.

A1.11 Procedure check_lambda_in_range

1. Purpose

To determine the points of intersection of a line with a helix.

2. Keywords

lambda, root, solution, intersection, discontinuity.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

For a section of line between two given points, a series of points on the line are substituted into the cartesian equations of the surface of the helix. The values given by this are stored in the arrays f1 and f2. These values are then checked to see where sign changes occur. A sign change means one of two things, either

1. There is an intersection between the two points
or 2. The equation has a discontinuity between the two points.

The possibility of a discontinuity arises because of the tan term in the equation.

5. Parameters

parameter	mode	type	description
lower_limit	in	double	lower value of parameter of line, lambda, corresponding to one

			end point of the line segment being checked for intersections.
upper_limit	in	double	upper value of parameter of line, corresponding to the other end of the line segment.
s	in	three_vec	position vector of a point on the line.
t	in	three_vec	direction vector of the line.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.

6. Error Indicators

None.

7. Auxiliary Routines

Check_lambda_in_range calls the procedure get_parameters and the functions line_helix1 and line_helix2.

8. Accuracy

All arithmetic is in double precision.

A1.12 Function line_helix1

1. Purpose

To help determine if a given point lies on the surface of a helix by checking whether it satisfies one of the cartesian equations of the surface.

2. Keywords

None.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The co-ordinates of the point to be checked are substituted into one of the cartesian equations of the surface. If the value given by the function is zero then the point is on the surface of the double helix. For more details see section 3.5.1 of this report.

5. Parameters

parameter	mode	type	description
lambda	in	double	value of the parameter of the line corresponding to the point to be checked.
s	in	three_vec	position vector of a point on the line.
t	in	three_vec	direction vector of the line.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All arithmetic is in double precision.

A1.12 Function line_helix2

1. Purpose

To help determine if a given point lies on the surface of a helix by checking whether it satisfies one of the cartesian equations of the surface.

2. Keywords

None.

3. Language

Ada. ANSI / MIL - STD 1815A

4. Description

The co-ordinates of the point to be checked are substituted into one of the cartesian equations of the surface. If the value given by the function is zero then the point is on the surface of the double helix. For more details see section 3.5.1 of this report.

5. Parameters

parameter	mode	type	description
lambda	in	double	value of the parameter of the line corresponding to the point to be checked.
s	in	three_vec	position vector of a point on the line.
t	in	three_vec	direction vector of the line.
major_radius	in	double	major radius of helix.
minor_radius	in	double	minor radius of helix.
pitch	in	double	pitch of helix divided by 2π.

6. Error Indicators

None.

7. Auxiliary Routines

None.

8. Accuracy

All arithmetic is in double precision.

STOR-A-FILE IMAGING LTD

DOCUMENTS OF POOR ORIGINAL HARD COPY

Date 12/02/08 Authorised by: Simon Cockbill Issue 2 Page 1 of 1

If in hard copy, this page is UNCONTROLLED and only valid on date of issue 16-May-08

```
with pafec_base_types               ; use pafec_base_types         ;
with pafec_standard_types           , use pafec_standard_types     ;
with pafec_int_io                   ; use pafec_int_io             ;
with pafec_double_io                , use pafec_double_io          ;
with text_io                        , use text_io                  ;
with pafec_maths                    , use pafec_maths              ;

procedure line_intersect_helix is

type   three_vec    is array(1  3) of double    ;
type   four_vec     is array(1  4) of double    ,

s                         .  three_vec          ;
t                         :  three_vec          ,
lambda                    :  four_vec           ;
major_radius              .  double             ,
minor_radius              .  double             ;
pitch                        double             ,
intersection_possible        boolean            ,
point_intersection        :  boolean            ,
intersect_inner_cyl       •  boolean            ;

  procedure get_line ( s          :    out   three_vec    ,
                       t          .    out,  three_vec     ) is

  begin

    put("type in co-ordinates of first point on line =>"),
    get(s(1)),   get(s(2)),  get(s(3)),
    new_line,
    put("type in direction vector of line =>");
    get(t(1)),   get(t(2)),  get(t(3));
    new_line,

  end get_line ;

  procedure get_helix_implicit (major_radius  .   out  double ;
                                minor_radius  .   out  double ,
                                pitch             out  double  ) is

  begin

    put("major radius of helix"), new_line,
    get(major_radius),
    put("minor radius of helix"); new_line,
    get(minor_radius),
    put("type in pitch of the helix =>"); new_line,
    get(pitch),

  end get_helix_implicit ;

  procedure check_for_intersection_vertical ( s                         . in       three_vec    ;
                                              major_radius                in       double       .
                                              minor_radius              . in       double       ,
                                              intersection_possible       _ out    boolean       ) is

  radius      . double       ,

  begin

    intersection_possible  = true ;
    radius  = s(1) * s(1) + s(2) * s(2) ,
```

```
    if radius > ( major_radius + minor_radius ) ** 2  or radius < ( major_radius - minor_radius ) ** 2  then

      intersection_possible  = false ,

    end if;

  end check_for_intersection_vertical,

  procedure calculate_intersection_vertical ( s                   ·in     three_vec  ;
                                              major_radius        ·in     double     ,
                                              minor_radius        .in     double     ,
                                              pitch                in     double      ) is

  cos_phi           : double                ,
  phi               . array(1. 2) of radian ;
  theta             . double                ;
  cos_theta         - double                ,
  sin_theta           double                ;
  z                 - array(1. 2) of double ,
  z_step            - double                ;

  begin

    cos_theta := s(1) / sqrt( s(1) ** 2 + s(2) ** 2 ) ;
    sin_theta := s(2) / sqrt( s(1) ** 2 + s(2) ** 2 ) ,

    if sin_theta < 0 0 then
      theta ·= double( two_pi - acos( cos_theta ) );
    else
      theta  = double( acos( cos_theta ) );
    end if,

    cos_phi .= ( sqrt( s(1) ** 2 + s(2) **2 ) - major_radius ) / minor_radius ,

    phi(1) .= acos( cos_phi ) ;
    phi(2) := two_pi - phi(1) ;

    z(1)  = minor_radius * sin(phi(1)) + pitch * theta ,
    z(2)  = minor_radius * sin(phi(2)) + pitch * theta ;
    z_step  = double(two_pi) * pitch ;

    put(" Infinite number of roots between (infinite) straight line and (infinite) helix ") ; new_line ;
    put(" All intersections occur at x = "),  put( s(1) ), put(" , y = ") , put( s(2) ) , new_line,
    put(" One set of roots is at z = "), put ( z(1) ) , put (" plus or minus "), put ( z_step ) ; new_line ,
    put(" The other set of roots is at z= "); put ( z(2) ) ; put (" plus or minus "); put ( z_step ) , new_line ,

  end calculate_intersection_vertical ,

  procedure intersect_bounding_cylinder ( s                         . in          three_vec    ,
                                          t                         : in          three_vec    ;
                                          major_radius              . in          double       ;
                                          minor_radius              : in          double       ,
                                          point_intersection        :    out      boolean      ,
                                          intersection_possible     -    out      boolean      ;
                                          intersect_inner_cyl            out      boolean      ,
                                          lambda                         out      four_vec      ) is

  descrim_outer_cyl           :  double ;
  descrim_inner_cyl           .  double ,
  a, b, c                     .  double ,

  begin

  -- initialise the logical flags and calculate a few values that will be useful in future calculations
```

```
        intersect_inner_cyl  = false ;
        intersection_possible -= true,
        point_intersection   = false ,
        a .= s(1) ** 2 + s(2) ** 2 ,
        b  = s(1) * t(1) + s(2) * t(2) ,
        c  = t(1) ** 2 + t(2) ** 2 ,

   -- now check to see whether the line intersects the outer bounding cylinder of the helix

        descrim_outer_cyl := b * b - c * ( a - ( major_radius + minor_radius ) ** 2 ) ;

        if descrim_outer_cyl > 0 0 then

          lambda(1)  = ( -b + sqrt( descrim_outer_cyl ) ) /  c ,
          lambda(2)  = ( -b - sqrt( descrim_outer_cyl ) ) /  c ;

   -- check to see whether the line intersects the inner bounding cylinder as well

          descrim_inner_cyl .= b * b - c * ( a - ( major_radius - minor_radius ) ** 2 ) ,

          if descrim_inner_cyl > 0 0 then

            lambda(3) -= ( -b + sqrt( descrim_inner_cyl ) ) / c ,
            lambda(4) -= ( -b - sqrt( descrim_inner_cyl ) ) / c ,
            intersect_inner_cyl  = true ;

          end if,

        elsif descrim_outer_cyl = 0 0 then

   -- line is tangential to outer bounding cylinder   Only one point need be checked

          lambda(1) := -b / ( 2 0 * c ) ;
          point_intersection  = true ;
          intersection_possible  = true ;

        else

   -- no intersection between line and bounding cylinders of helix

          intersection_possible := false ;

        end if ,

      end intersect_bounding_cylinder ;

      procedure check_point ( s                   : in     three_vec   ;
                              t                   . in     three_vec   ,
                              major_radius        . in     double      ,
                              minor_radius        . in     double      ,
                              pitch               : in     double      ,
                              lambda              : in     double        ) is

      solution            . array(1..3) of double ,
      x                   . three_vec             ;
      point_not_on_helix  - boolean               ,
      theta               . double                ,
      phi                 . double                ,

        procedure get_parameters ( x                  : in        three_vec ;
                                   major_radius       . in        double    ,
                                   minor_radius       . in        double    ,
                                   pitch              : in        double    ,
                                   point_not_on_helix .      out  boolean   ,
                                   theta              . in   out  double    ;
```

```
                                   phi                  in   out   double      ) is

        theta_found           boolean          ;
        phi_found             boolean          ,

          procedure try_and_find_theta ( x                 . in        three_vec   ,
                                         pitch               in        double      ,
                                         minor_radius      . in        double      ;
                                         theta               in   out  double      ,
                                         theta_found         in   out  boolean       ) is

          cos_theta           - double  ,
          sin_theta           - double  ;
          point_not_on_helix : boolean  ,

          begin
            theta_found  = false ,
            point_not_on_helix .= false ;
            cos_theta .= x(1) / sqrt(x(1)*x(1) + x(2)*x(2)),
            sin_theta  = x(2) / sqrt(x(1)*x(1) + x(2)*x(2)),
            if sin_theta < 0 0 then
              theta  = double( two_pi - acos( cos_theta ) ),
            else
              theta -= double( acos( cos_theta ) ),
            end if;

            if ( ( x(3) - pitch*theta >= -minor_radius ) and ( x(3) - pitch*theta <= minor_radius ) ) then

              theta_found  = true ,

            elsif x(3) - pitch*theta > -minor_radius then

              loop
                exit when ( point_not_on_helix = true ) or ( theta_found = true ) ,
                theta .= theta + double(two_pi) ,
                if ( ( x(3) - pitch*theta >= -minor_radius ) and ( x(3) - pitch*theta <= minor_radius ) ) then
                  theta_found  = true ,
                elsif ( ( x(3) - pitch*theta > -minor_radius - double(two_pi) * pitch )
                          and ( x(3) - pitch*theta < -minor_radius ) ) then
                  point_not_on_helix  = true ;
                end if,
              end loop,

            else
             loop
                exit when ( point_not_on_helix = true ) or ( theta_found = true ) ;
                theta  = theta - double(two_pi) ,
                if ( ( x(3) - pitch*theta >= -minor_radius ) and ( x(3) - pitch*theta <= minor_radius ) ) then
                  theta_found := true ,
                elsif ( ( x(3) - pitch*theta > minor_radius )
                          and ( x(3) - pitch*theta < double(two_pi) * pitch - minor_radius ) ) then
                  point_not_on_helix .= true ,
                end if,
              end loop,

            end if ,

          end try_and_find_theta ;

          procedure try_and_find_phi ( x                 : in       three_vec   ,
                                       theta             : in       double      ,
                                       major_radius      . in       double      ;
                                       minor_radius      : in       double      ;
                                       pitch             . in       double      ;
                                       phi               .      out double      ;
                                   phi_found      :     out boolean        ) is
      cos_phi                        . double  ;
      sin_phi                        . double  ;
      distance_from_surface          : double  ,
      point_not_on_helix             . boolean ,

      begin

        point_not_on_helix := false;

        cos_phi := ( sqrt( x(1) ** 2 + x(2) **2 ) - major_radius ) / minor_radius ,

        sin_phi := ( x(3) - pitch * theta ) / minor_radius ;

-- now to check whether or not the point
-- is actually on the helix

        if 1 0 - cos_phi ** 2 - sin_phi ** 2 < 0 0 then

          point_not_on_helix  = true,

        else

          distance_from_surface .= abs(sqrt(1 0 - cos_phi ** 2 - sin_phi ** 2 )) ,
          if distance_from_surface <= 1 0e-8 then

            phi_found  = true,
            if sin_phi < 0 0 then

              phi  = double( two_pi - acos(cos_phi)),

            else

              phi := double(acos(cos_phi));

            end if;

          else

            point_not_on_helix := true,

          end if,

        end if,

      end try_and_find_phi ;

    begin

      theta_found := false ,
      phi_found := false ;
      point_not_on_helix  = false ,
      try_and_find_theta ( x, pitch, minor_radius, theta, theta_found ) ,

      if theta_found = false then

        point_not_on_helix  = true ,

      else

        try_and_find_phi ( x, theta, major_radius, minor_radius, pitch, phi, phi_found ) ,

        if phi_found = false then

          point_not_on_helix := true ,

         end if;

      end if,

    end get_parameters ;

  begin

    for countvar in 1. 3 loop
      x(countvar)  = s(countvar) + lambda * t(countvar) ;
    end loop ,

    get_parameters( x, major_radius, minor_radius, pitch, point_not_on_helix, theta, phi ) ,

    if point_not_on_helix = true then

      put(" Line touches helix at one point  Co-ordinates are => ") ; new_line ,
      for count in 1 .3 loop

        put(x(count)) , put("  ") ,

      end loop ;

    else

        put(" No intersection ") , new_line ,

    end if ,

  end check_point ,

  procedure check_lambda_in_range ( lower_limit  .  in         double    ,
                                    upper_limit  .  in         double    ;
                                    s            .  in         three_vec ,
                                    t            .  in         three_vec ,
                                    major_radius :  in         double    ;
                                    minor_radius .  in         double    ,
                                    pitch        :  in         double       )  is

  f1                             .  array(1  50) of double             ;
  f2                                array(1. 50) of double             ,
  a                                 double                             ,
  b                                 double                             ,
  c                                 double                             ,
  lambda                            double                             ,
  lambda1                        .  double                             ;
  lambda2                        .  double                             ,
  lambda_step                    :  double                             ;
  root                           .  double                             ;
  better_guess                   .  double                             ;
  no_of_roots                    :  int .= 0                           ,
  discontinuity                  :  boolean .= false                   ,
  x                              :  three_vec                          ;
  solutions                      .  array(int range 1 .300) of double  ,
  intersection                   :  boolean := false                   ,
  point_not_on_helix             :  boolean                            ,
  theta                          .  double                             ,
  phi                            :  double                             ,

    function  line_helix1 (lambda                       double    ,
                          s             .               three_vec ,
                           t            .           three_vec ;
                           major_radius :           double    ,
                           minor_radius .           double    ;
                           pitch        .           double     ) return double is

  a      . double              ,
  b        double              ;
  c        double              ;
  fun    · double              ;

  begin

    a  = s(1) ** 2 + s(2) ** 2 ,
    b  = 2 0 * (s(1) * t(1) + s(2) * t(2) ) ,
    c  = t(1) ** 2 + t(2) ** 2 ;

    fun  = s(2) + lambda * t(2) - ( s(1) + lambda * t(1) )
                 * tan(radian( s(3) + lambda * t(3) + sqrt( minor_radius ** 2 - a - major_radius ** 2 - lambda*b - c * lambda ** 2
                        + 2 0 * major_radius * sqrt( a + lambda * b + c * lambda ** 2 ) ) / pitch ) );
    return fun ;

  end line_helixi ;

  function  line_helix2 (lambda       :           double    ,
                         s                        three_vec ,
                         t            .           three_vec ,
                         major_radius             double    ;
                         minor_radius ·           double    ;
                         pitch        :           double     ) return double is

  a      · double              ;
  b        double              ;
  c      · double              ,
  fun    : double              ,

  begin

    a .= s(1) ** 2 + s(2) ** 2 ,
    b ·= 2 0 * (s(1) * t(1) + s(2) * t(2) ) ;
    c .= t(1) ** 2 + t(2) ** 2 ,

    fun  = s(2) + lambda * t(2) - ( s(1) + lambda * t(1) )
                 * tan(radian( s(3) + lambda * t(3) - sqrt( minor_radius ** 2 - a - major_radius ** 2 - lambda*b - c * lambda ** 2
                      .+ 2 0 * major_radius * sqrt( a + lambda * b + c * lambda ** 2 ) ) / pitch ) ),
    return fun ;

  end line_helix2 ;

  procedure get_parameters ( x                     in        three_vec ,
                             major_radius        · in        double    ,
                             minor_radius          in        double    ,
                             pitch               · in        double    ,
                             point_not_on_helix       out    boolean   ,
                             theta                 in out    double    ,
                             phi                   in out    double     ) is

  theta_found        :  boolean        ,
  phi_found          :  boolean        ;

    procedure try_and_find_theta ( x                 in       three_vec   ;
                                   pitch           : in       double      ,
                                   minor_radius    : in       double      ,
                                   theta           . in  out  double      ,
                                   theta_found     · in  out  boolean      ) is
```

```
      cos_theta            : double  ;
      sin_theta            · double  ,
      point_not_on_helix . boolean ,

      begin
        theta_found .= false ;
        point_not_on_helix  = false ,
        cos_theta  = x(1) / sqrt(x(1)*x(1) + x(2)*x(2)),
        sin_theta  = x(2) / sqrt(x(1)*x(1) + x(2)*x(2));
        if sin_theta < 0 0 then
          theta  = double( two_pi - acos( cos_theta ) ),
        else
          theta  = double( acos( cos_theta ) ),
        end if,

        if ( ( x(3) - pitch*theta >= -minor_radius ) and ( x(3) - pitch*theta <= minor_radius ) ) then

          theta_found  = true ,

        elsif x(3) - pitch*theta > -minor_radius then

          loop
            exit when ( point_not_on_helix = true ) or ( theta_found = true ) ;
            theta ·= theta + double(two_pi) ;
            if ( ( x(3) - pitch*theta >= -minor_radius ) and ( x(3) - pitch*theta <= minor_radius ) ) then
              theta_found ·= true ,
            elsif ( ( x(3) - pitch*theta > -minor_radius - double(two_pi) * pitch )
                      and ( x(3) - pitch*theta < -minor_radius ) ) then
              point_not_on_helix  = true ,
            end if;
          end loop;

        else
         loop
            exit when ( point_not_on_helix = true ) or ( theta_found = true ) ,
            theta  = theta - double(two_pi) ,
            if ( ( x(3) - pitch*theta >= -minor_radius ) and ( x(3) - pitch*theta <= minor_radius ) ) then
              theta_found  = true  ;
            elsif ( ( x(3) - pitch*theta > minor_radius )
                      and ( x(3) - pitch*theta < double(two_pi) * pitch - minor_radius ) ) then
              point_not_on_helix ·= true ,
            end if,
          end loop,

        end if ;

      end try_and_find_theta ,

      procedure try_and_find_phi ( x                · in      three_vec    ,
                                   theta              in      double       ,
                                   major_radius     · in      double       ,
                                   minor_radius       in      double       ,
                                   pitch              in      double       ,
                                   phi              ·     out double       ,
                                   phi_found        ·     out boolean       ) is

      cos_phi                  : double  ;
      sin_phi                    double  ,
      distance_from_surface    · double  ;
      point_not_on_helix       · boolean ,

      begin

        point_not_on_helix  = false,
          cos_phi .= ( sqrt( x(1) ** 2 + x(2) **2 ) - major_radius ) / minor_radius ,

          sin_phi  = ( x(3) - pitch * theta ) / minor_radius ;

  -- now to check whether or not the point
  -- is actually on the helix

          if 1 0 - cos_phi ** 2 - sin_phi ** 2 < 0 0 then

            point_not_on_helix  = true,

          else

            distance_from_surface := abs(sqrt(1 0 - cos_phi ** 2 - sin_phi ** 2 )) ;
            if distance_from_surface <= 1.0e-8 then

              phi_found -= true,
              if sin_phi < 0 0 then

                phi  = double( two_pi - acos(cos_phi)),

              else

                phi  = double(acos(cos_phi)),

              end if,

            else

              point_not_on_helix  = true,

            end if,

          end if,

        end try_and_find_phi ,

      begin

        theta_found  = false ;
        phi_found .= false ;
        point_not_on_helix .= false ;
        try_and_find_theta ( x, pitch, minor_radius, theta, theta_found ) ,

        if theta_found = false then

          point_not_on_helix  = true ,

        else

          try_and_find_phi ( x, theta, major_radius, minor_radius, pitch, phi, phi_found ) ;

          if phi_found = false then

            point_not_on_helix .= true ,

           end if,

        end if,

      end get_parameters ;

    begin
```

```
        a := s(1) ** 2 + s(2) ** 2 ,
        b  = 2 0 * ( s(1) * t(1) + s(2) * t(2) ) ,
        c := t(1) ** 2 + t(2) ** 2 ,
        lambda_step  = ( upper_limit - lower_limit ) / 49.0 ,

        for count in 1  50 loop

          lambda  = lower_limit + lambda_step * double ( count - 1 ) ,
          f1(count)  = line_helix1( lambda, s, t, major_radius, minor_radius, pitch ) ;
          f2(count)  = line_helix2( lambda, s, t, major_radius, minor_radius, pitch ) ;
        end loop ,

        for count in 1..49 loop

          if f1(count) * f1(count+1) < 0.0 then

  -- there has been a sign change.  Could be either a root or a discontinuity
  -- Use linear interpolation to get a better guess for the possible root

            lambda1 .= lower_limit + lambda_step * double( count -1 ) ,
            lambda2 .= lambda1 + lambda_step ;
            intersection  = false ,
            discontinuity := false ;

            loop
              exit when discontinuity = true or intersection = true ,
              better_guess := lambda1 - ( lambda2 - lambda1 ) *
                              line_helix1 (lambda1, s, t, major_radius, minor_radius, pitch ) /
                              (line_helix1 (lambda2, s, t, major_radius, minor_radius, pitch ) -
                               line_helix1 (lambda1, s, t, major_radius, minor_radius, pitch ) ) ,

              if line_helix1 (better_guess, s, t, major_radius, minor_radius, pitch )
                  * line_helix1 (lambda1, s, t, major_radius, minor_radius, pitch ) < 0 0 then

                lambda2 := better_guess ;

              else

                lambda1 := better_guess,

              end if ;

              if abs( line_helix1 (better_guess, s, t, major_radius, minor_radius, pitch ) ) <= 1 0e-8 then

                no_of_roots  = no_of_roots + 1 ;
                solutions(no_of_roots) -= better_guess ,
                intersection  = true ,

              elsif abs(line_helix1 (better_guess, s, t, major_radius, minor_radius, pitch ) ) > 10 0 then

                discontinuity  = true ,

              end if ,

            end loop ,

          elsif f1(count) * f1(count + 1) = 0 0 then

            if f1(count) = 0 0 then

              root .= lower_limit + lambda_step * double( count - 1 ) ;
              no_of_roots  = no_of_roots +1 ,
              solutions(no_of_roots)  = root ;

            else
          root := lower_limit + lambda_step * double( count ) ;
          no_of_roots := no_of_roots +1 ;
          solutions(no_of_roots) := root ;

        end if ;

      end if ;

    end loop ;

    for count in 1..49 loop

      if f2(count) * f2(count+1) < 0.0 then

-- there has been a sign change.  Could be either a root or a discontinuity
-- Use linear interpolation to get a better guess for the possible root

        lambda1  := lower_limit + lambda_step * double( count -1 ) ;
        lambda2  := lambda1 + lambda_step ;
        intersection := false ;
        discontinuity := false ;

        loop
          exit when discontinuity = true or intersection = true ;
          better_guess  := lambda1 - ( lambda2 - lambda1 ) *
                           line_helix2 (lambda1, s, t, major_radius, minor_radius, pitch ) /
                           (line_helix2 (lambda2, s, t, major_radius, minor_radius, pitch ) -
                            line_helix2 (lambda1, s, t, major_radius, minor_radius, pitch ) ) ;

          if line_helix2 (better_guess, s, t, major_radius, minor_radius, pitch )
              * line_helix2 (lambda1, s, t, major_radius, minor_radius, pitch ) < 0.0 then

            lambda2 := better_guess ;

          else

            lambda1  := better_guess;

          end if ;

          if abs( line_helix2 (better_guess, s, t, major_radius, minor_radius, pitch ) ) <= 1.0e-8 then

            no_of_roots := no_of_roots + 1 ;
            solutions(no_of_roots)  := better_guess ;
            intersection  := true ;

          elsif abs(line_helix2 (better_guess, s, t, major_radius, minor_radius, pitch ) ) > 10.0 then

            discontinuity := true ;

          end if ;

        end loop ;

      elsif f2(count) * f2(count + 1) = 0.0 then

        if f2(count) = 0.0 then

          root := lower_limit + lambda_step * double( count - 1 ) ;
          no_of_roots  := no_of_roots +1 ;
          solutions(no_of_roots) := root ;

        else
```

```
          root := lower_limit + lambda_step * double( count ) ;
          no_of_roots := no_of_roots +1 ;
          solutions(no_of_roots) := root ;

        end if ;

      end if ;

    end loop ;

    if no_of_roots = 0 then

      put (" No intersection "); new_line ;

    else

      for count in 1..no_of_roots loop

        lambda  := solutions(count) ;
        for countvar in 1..3 loop
          x(countvar) := s(countvar) + lambda * t(countvar) ;
        end loop;
        get_parameters ( x, major_radius, minor_radius, pitch, point_not_on_helix, theta, phi ) ;

        if point_not_on_helix = false then

          put ("Co-ordinates of intersection "),  new_line;
          put (x(1)); put ("  "); put (x(2)); put ("  "); put (x(3)); put ("  ");
          put (" lambda = ");  put(lambda) ; new_line ;
          put (" theta = ");  put(theta) ; new_line ;
          put (" phi = ");  put(phi) ; new_line ;

        end if ;

      end loop ;

    end if ;

  end check_lambda_in_range ;

begin

  get_line ( s, t ) ;
  get_helix_implicit ( major_radius, minor_radius, pitch ) ;

  put(" Data accepted "); new_line ;

-- check to see whether the line is vertical ( i.e. parallel to the axis of the helix ), in which
-- case it will not intersect the bounding cylinder of the helix but it may intersect the helix .

  if ( t(1) ** 2 + t(2) ** 2 ) = 0.0 then

    check_for_intersection_vertical ( s, major_radius, minor_radius, intersection_possible ) ;

    if intersection_possible = true then

      calculate_intersection_vertical ( s, major_radius, minor_radius, pitch ) ;

    else

      put(" No intersection ");

    end if ;
```

```
      else

        intersect_bounding_cylinder( s, t, major_radius, minor_radius, point_intersection, intersection_possible,
                                     intersect_inner_cyl, lambda ) ;

        put(" Cylinder intersected "); new_line ;

        if intersection_possible = false then

          put(" No intersection ");

        else

          if point_intersection = true then

            check_point ( s, t, major_radius, minor_radius, pitch, lambda(1) ) ;

          else

            if intersect_inner_cyl = true then

              check_lambda_in_range ( lambda(2), lambda(4), s, t, major_radius, minor_radius, pitch ) ;
              check_lambda_in_range ( lambda(3), lambda(1), s, t, major_radius, minor_radius, pitch ) ;

            else

              check_lambda_in_range ( lambda(2), lambda(1), s, t, major_radius, minor_radius, pitch ) ;

            end if ;

          end if ;

        end if ;

      end if ;

    end line_intersect_helix ;
```

Appendix 2

Program to Calculate the Intersection Between a Plane and a Helix

This program uses the following Ada packages :-

pafec_base_types
pafec_standard_types
pafec_int_io
pafec_double_io
text_io
pafec_maths

The program was written in Ada (ANSI / MIL-STD 1815A) and run on Apollo DN3000 and DN4000 workstations.

1. Name of Procedure

plane_helix_parametric

2. Purpose

To calculate points of intersection between a plane and a helix. The points may then be interpolated to give the curves of intersection.

3. Keywords

helix, theta.

4. Language

Ada. ANSI / MIL - STD 1815A

5. Description

This procedure reads in data specifying a plane and a helix. The helix is specified by its major and minor radii, pitch, and the number of coils in the helix. The plane is specified by the direction of its normal and the perpendicular distance from the plane to the origin. The procedure calculates the intersection by the method detailed in section 3.6, that is, for each value of theta it solves an equation in phi to find two points of intersection. For more details see sections 3.6 and 4.2.

6. Parameters

None.

7. Error Indicators

None.

8. Auxiliary Routines

None.

9. Accuracy

All arithmetic is carried out in double precision.

```
with pafec_base_types                 , use pafec_base_types         ,
with pafec_standard_types             ; use pafec_standard_types     ,
with pafec_int_io                     , use pafec_int_io             ,
with pafec_double_io                  , use pafec_double_io          ;
with text_io                          , use text_io                  ,
with pafec_maths                      , use pafec_maths              ,
------------------------------------------------------------------------
procedure plane_helix_parametric is
  n                            . array (1. 3) of double;
  s                            : array (1. 3) of double,
  t                            . array (1  2) of double;
  theta                        · radian ;
  theta_step                     radian ,
  pitch                        : double ;
  d,a,b,k1,k2                  · double ,
  major_radius                 : double ;
  minor_radius                 . double ,
  discrim                      . double ,
  num                          : double ,
------------------------------------------------------------------------
begin
  put("type in co-ordinates of direction vector of normal to plane =>");
  get(n(1)),   get(n(2)),  get(n(3)); new_line,
  put("perpendicular distance from plane to origin ( d ) =>"); new_line,
  get(d),
  put("major radius of helix"); new_line,
  get(major_radius),
  put("minor radius of helix"), new_line,
  get(minor_radius);
  put("pitch factor ( k ) =>"), new_line;
  get(pitch),
  put("number of rotations of section around helical centre line =>"), new_line;
  get(num);
  put("theta varies between 0 0 and ");  put(2 0 * double(pi) * num),   new_line,
  put("type in steplength of theta variation");  new_line   ,
  get(theta_step),
  theta  = 0 0 ,

--  solve quadratic equation in t = tan( phi/2 )   Check for no real roots or a
--  double root

  while double(theta) < 2 0 * double(pi) * num loop

    k1 .= n(1) * cos(theta) - n(2) * sin(theta),
    k2  = n(3) * pitch * double(theta) + d,
    discrim  = minor_radius ** 2  * n(3) ** 2
                 - ( (major_radius - minor_radius) * k1 + k2) * ((major_radius + minor_radius) * k1 + k2 ),

-- first check for complex roots

    if discrim < 0.0 then

      new_line   ,put("no real roots for theta = "),   put(double(theta)),

-- now check for a double root

    elsif discrim = 0 0 then

      new_line  ,put("double root for theta = "),     put(double(theta)),   new_line,

      if ( major_radius - minor_radius ) * k1 + k2 = 0.0 then

        s(1) ·= ( major_radius - minor_radius ) * cos(theta),
        s(2) .= -( major_radius - minor_radius ) * sin(theta),
        s(3) := pitch * double(theta),
```

```
      else

        t(1)  = ( -minor_radius * n(3)) / (( major_radius - minor_radius ) * k1 + k2 );
        s(1) .= ( major_radius + minor_radius *( 1 0 - t(1) * t(1) ) / ( 1 0 + t(1) * t(1) ) ) * cos(theta);
        s(2)  = - ( major_radius + minor_radius * ( 1 0 - t(1) * t(1) ) / ( 1 0 + t(1) * t(1) ) ) * sin(theta),
        s(3) .= 2 0 * t(1) * minor_radius / (1.0 + t(1) * t(1) ) + pitch * double(theta),

      end if,

        put(s(1)) ; put("  ")  ,put(s(2))  ,put("  ")  ,put(s(3))  ,new_line  ;
------------------------------------------------------------------------
-- two distinct roots.  Check for one root at infinity.
------------------------------------------------------------------------
    else

      if ( major_radius - minor_radius ) * k1 + k2 = 0 0 then
        put("roots for theta = "),           put(double(theta)),   new_line,
        s(1)  = ( major_radius - minor_radius ) * cos(theta);
        s(2) ·= -( major_radius - minor_radius ) * sin(theta),
        s(3)  = pitch * double(theta),
        new_line,
        put(s(1))   ,put("  ")  ,put(s(2))  ,put("  ")  ,put(s(3))  ,new_line  ;

--    now the other root, which is the root of the linear equation formed by putting the
--    leading coefficient of the quadratic equal to zero

        t(1)  = - ( ( major_radius + minor_radius ) * k1 + k2 ) / ( 2.0 * minor_radius * n(3) ),
        s(1)  = ( major_radius + minor_radius *( 1 0 - t(1) * t(1) ) / ( 1 0 + t(1) * t(1) ) ) * cos(theta),
        s(2)  = - ( major_radius + minor_radius * ( 1 0 - t(1) * t(1) ) / ( 1 0 + t(1) * t(1) ) ) * sin(theta),
        s(3)  = 2.0 * t(1) * minor_radius / (1.0 + t(1) * t(1) ) + pitch * double(theta),

--    two distinct, finite roots

      else

        t(1)  = ( -( minor_radius * n(3) ) + sqrt(discrim) ) / ( ( major_radius - minor_radius ) * k1 + k2 ),
        t(2) .= ( -( minor_radius * n(3) ) - sqrt(discrim) ) / ( ( major_radius - minor_radius ) * k1 + k2 ),
        new_line,
        put("roots for theta = "),          put(double(theta))  ;     new_line,


        for i in 1. 2 loop

          s(1)  = ( major_radius + minor_radius *( 1 0 - t(i) * t(i) ) / ( 1 0 + t(i) * t(i) ) ) * cos(theta),
          s(2) ·= - ( major_radius + minor_radius * ( 1 0 - t(i) * t(i) ) / ( 1 0 + t(i) * t(i) ) ) * sin(theta),
          s(3)  = 2 0 * t(i) * r2 / (1.0 + t(i) * t(i) ) + pitch * double(theta),
          put(s(1))   ,put("  ")  ,put(s(2))  ,put("  ")  ,put(s(3))  ,new_line  ,

        end loop,

      end if,

    end if,

  theta  = theta + theta_step ,

  end loop,
  new_line;

end plane_helix_parametric,
```

Appendix 3

Suggested Reading

The following books are suitable background reading :-

"Computational Geometry for Design and Manufacture" by I. D. Faux and M. J. Pratt (Ellis Horwood, 1979)

"Geometric Modelling" by M. E. Mortensen (John Wiley & Sons, 1985)

"Geometry of Spatial Forms" by P. C. Gasson (Ellis Horwood, 1983)