MCM 1995 Part A - Single Helix

Adam Michael, Garrett Shaffer and Yunhui Shi

Problem

Engineers rely on Computer Aided Geometric Design programs to design and manufacture their products. One common use of such programs is to calculate and visualize planar cross sections of the design. As such, it is important for the program to quickly compute the intersection of an arbitrary plane and each piece of the design. Finding closed form solutions for these intersections is not always possible so the programs often resort to approximation algorithms. Here we consider the problem of finding all intersections of a plane and a helix, a shape used frequently in chemical and medical apparatus.

Assumptions

- We assume that the helix is infinitely thin
- We assume that the plane is infinitely thin
- We assume that the helix is infinitely long
- We assume that the plane is infinite
- We assume that if there are an infinite number of intersections, the CAD software will only require and will pre-specify either a finite amount or a range to return

Model

To begin, we assume that our helix runs in the direction of vector *v*. Select a point *p* on this vector. We define our Cartesian coordinate system such that the z axis is along vector *v* and the x axis is the unique line that is orthogonal to *v*, passes through point *p* and intersects the helix. Finally we select the orthogonal direction for the y axis that gives our coordinate system positive orientation.

Next we need to define the general form of a plane which will be interacting with the helix. The first step is to choose the normal vector to the plane. This vector will be of the form: $< A, B, C >$. Then the users must also select a point on the plane: (x_0, y_0, z_0).

Using the defined coordinate system, we arrive at the following general equations for our helix and arbitrary plane:

Plane: $A\,(x-x_0)+B\,(y-y_0)+C\,(z-z_0)=0$

Helix:
$$x = R\,cos(\omega t)$$
$$y = R\,sin(\omega t) \qquad (1)$$
$$z = t$$

In this model A, B, C, x_0, y_0, z_0 are merely constants that define the arbitrary plane. R is the radius of the helix and ω is the frequency of the rotation of helix with respect to the central axis.

Reduction

To find the intersections of the helix and plane, we first reduce it to a simpler problem. We reduce the problem to finding all of the intersections of a line and a sinusoid in two dimensions.

We begin our reduction with the definitions of the plane and the helix as given by (1). First, substitute the helix equations into the plane equation.

$$A\,(R\,cos(\omega t)-x_0)+B(R\,sin(\omega t)\;-\;y_0)+C(t-z_0)=0$$

Move the components of the line to the right hand side

$$AR\,cos(\omega t)+BR\,sin(\omega t)=Ax_0+By_0+Cz_0-Ct$$

Recall the linear combinations trig identity,

$$a\,cos(x)+b\,sin(x)=\sqrt{a^2+b^2}sin(x+arctan2(a,b))$$

Note that the $arctan2$ function is simply the four quadrant version of arctangent. We can use this identity to simplify our equation with two sinusoids into one:

$$R\sqrt{A^2+B^2}sin(\omega t+arctan2(A,B))=Ax_0+By_0+Cz_0-Ct \qquad (2)$$

We have reached an equation that represents the intersection of a sinusoid and a line. The solutions for *t* of (2) are the same as the solutions for *t* of (1) and can therefore be used to find the intersection points of the plane and the helix.

Algorithm

The problem of finding the intersections of a line and a sinusoid can be broken into four cases. We will form a set of criteria to determine which case an arbitrary plane and helix

described by (1) resides in. For each case, we will determine either a closed form solution for *t* or an algorithm to approximate *t*. Geometrically, the cases are:

1. The line is parallel to the sinusoid and they do not intersect anywhere
2. The line is parallel to the sinusoid and they intersect at infinitely many points
3. The line is not parallel to the sinusoid and the intersect at finitely many points

By "the line is parallel to the sinusoid", we mean that the line is parallel to the axis that the sinusoid oscillates around. In our case, this means that the line has zero slope.

Case 1: No Intersections

For the line to be parallel to sinusoid, the slope of the line must be zero. Therefore we must have that $C = 0$. So the line is a constant function given by $Ax_0 + By_0 + Cz_0$. All sinusoids have a restricted range, specifically the range of our sinusoid is $[-R\sqrt{A^2+B^2}, R\sqrt{A^2+B^2}]$. So, there will be an intersection if and only if the constant value that is our line function is in that range. We can then formulize our criteria for this case as:

Criteria for case 1: $C = 0 \; and \; |Ax_0 + By_0| > R\sqrt{A^2+B^2}$

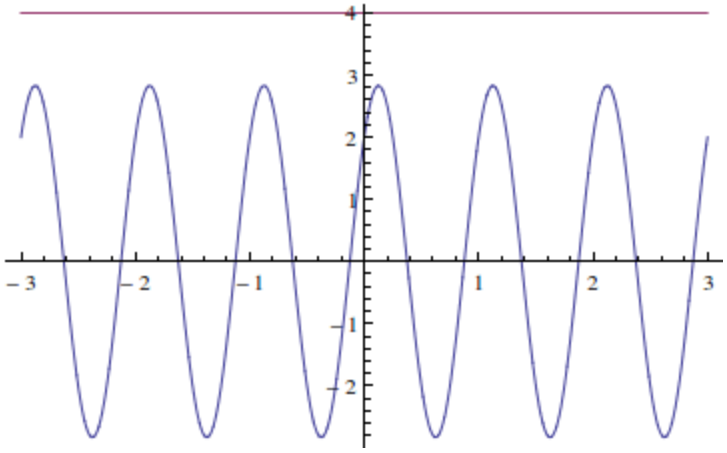

Figure 1: Example of Case 1

Case 2: Infinite Intersections

As in Case 1, in order for the line to be parallel to the sinusoid we must have that $C = 0$. However, in this case the constant value that is our line must be within the range of the sinusoid. So we can formulize the criteria for this case as:

Criteria for case 2: $C = 0 \; and \; |Ax_0 + By_0| \leq R\sqrt{A^2+B^2}$

We can find the intersections for this case by solving (2) with $C = 0$.

$$R\sqrt{A^2+B^2}\,sin(\omega t + arctan2(A,B)) = Ax_0 + By_0$$
$$sin(\omega t + arctan2(A,B)) = \frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}$$

Using our criteria for this case, we have that $|\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}| \leq 1$ so we can safely use *arcsin*. The infinite solutions stem from the fact that sin is a periodic function. Our case has period $2\pi/\omega$. So for all integers *k* we have that

$$\omega t + arctan2(A,B) = arcsin(\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}) + 2\pi k$$
$$t = \frac{1}{\omega}(\ 2k\pi + \ arcsin(\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}) - arctan2(A,B)\ )$$

There is a second infinite set of solution that results from the fact that sin realizes each value in its range twice in each period. So for all integers *k* we also have that

$$\omega t + arctan2(A,B) = \pi - arcsin(\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}) + 2\pi k$$
$$t = \frac{1}{\omega}(\ (2k+1)\pi - \ arcsin(\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}) - arctan2(A,B)\ )$$

We have produced a periodic description of all solutions *t* to (2) and therefore to (1) for this case. These values for *t* can be using in (1) to calculate the Cartesian coordinates of the intersection points of the plane and the helix.

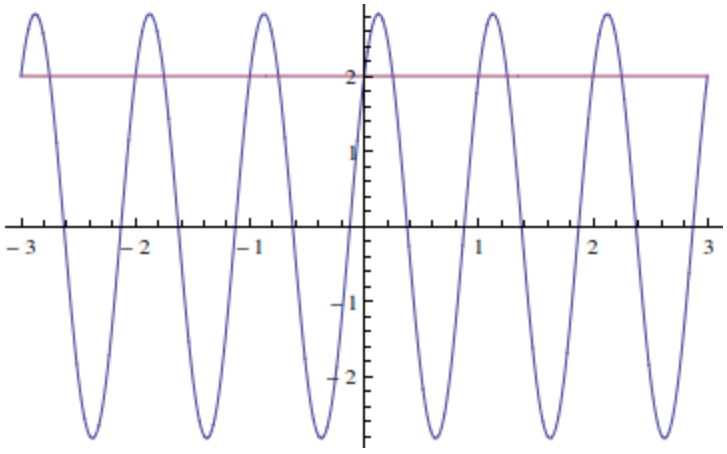

Figure 2: Example of Case 2

Case 3: Finite, Nonzero Number of Intersections

As there are only three possible cases, the criteria for this case is that neither the criteria for Case 1 nor the criteria for Case 2 have been met. Alternatively, since this case requires that the line and the sinusoid are not parallel, we can formulate the criteria as:

Criteria for Case 3: $C \neq 0$

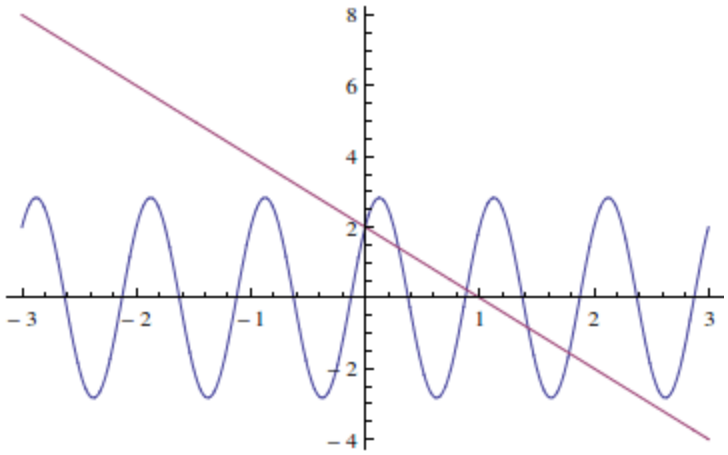

Figure 3: Example of Case 3

There is no closed form solution for the intersections using only elementary functions. Instead, we resort to approximation of the zeros of (3)

$$F(t) = R\sqrt{A^2 + B^2}sin(\omega t + arctan2(A,B)) - Ax_0 - B - Cz_0 + Ct = 0 \quad (3)$$

We begin with a few observations about this function that are necessary to formulate our algorithm to find the zeros

1. All intersections must occur when the line is within the range of the amplitude of the sinusoid. We calculate the upper and lower endpoints of that interval and call them t_1 and t_0, respectively.

$$Ax_0 + By_0 + Cz_0 - C\ t_0 = \pm R\sqrt{A^2 + B^2}$$

$$t_0 = min\{\tfrac{1}{C}(Ax_0 + By_0 + Cz_0 \pm R\sqrt{A^2 + B^2})\}$$

$$t_1 = max\{\tfrac{1}{C}(Ax_0 + By_0 + Cz_0 \pm R\sqrt{A^2 + B^2})\}$$

1. We can find $F'(t)$

$$F'(t) = R\omega\sqrt{A^2 + B^2}cos(\omega t + arctan2(A,B)) + C \quad (3)$$

2. $F'(t)$ is a sinusoid with a vertical offset. Therefore, it has no zeros or infinitely many periodic zeros.

3. $F'(t)$ has infinitely many zeros if and only if

$$-1 \leq \frac{C}{R\omega\sqrt{A^2+B^2}} \leq 1 \quad (4)$$

3. If $F'(t)$ has zeros, we can find all of them

$$R\omega\sqrt{A^2+B^2}cos(\omega t + arctan2(A,B)) + C = 0$$

$$cos(\omega t + arctan2(A,B)) = \frac{-C}{R\omega\sqrt{A^2+B^2}}$$

$$\omega t = 2\pi k \pm arccos(\frac{-C}{R\omega\sqrt{A^2+B^2}}) - arctan2(A,B)$$

$$t = \frac{1}{\omega}(2\pi k \pm arccos(\frac{-C}{R\omega\sqrt{A^2+B^2}}) - arctan2(A,B)) \text{ for any integer } k \quad (5)$$

4. The zeros of $F'(t)$ correspond to the minima and maxima of $F(t)$

5. If $F'(t)$ has no zeros, then $F(t)$ can have at most one zero and we already know that it has at least one, so $F(t)$ has exactly one zero.

Consider the case where $F'(t)$ has no zeros and therefore $F(t)$ has exactly one zero. We know that this zero must lie within $[t_0, t_1]$ so we can simply use a search method to locate it.

Now consider the case where $F'(t)$ has infinitely many zeros, calculated by (5). Then $F(t)$ has infinitely many periodic extrema which alternate between minimum and maximum. These minima and maxima partition the domain of $F(t)$ so each zero must be between or equal to a sequential minimum and maximum pair. Also, each pair of consecutive extrema can have at most one zero between them, or else they would not be consecutive extrema. If we were to plug these values into $F(t)$ we would see that for $F(t_{min})$ either strictly increases of strictly decreases as we iterate through our values for t_{min}. It is the same for the values of t_{max}. Therefore, all the zeros of $F(t)$ occur in consecutive intervals that are partitioned by our values of t_{min} and t_{max}. We can determine whether or not the interval contains a zero by comparing the signs of the function of its endpoints. We begin by searching the partition that contains t_0. We look through each partition, checking whether or not it contains a zero. Once we find the first interval that contains zero, we use a searching method to identify the zero. We do not know before hand how many zeros there are, so we continue checking endpoints and collecting zeros until we reach an interval that does not contain a zero, at which point we are done. Finally, we use (1) to return the Cartesian coordinates of each of the intersections points we just found.

The algorithm outlined above uses a searching method once it has identified that there is exactly one zero within that interval. We considered several searching methods and decided on the bisection method. The bisection method is not the fastest searching method, however it is guaranteed to converge whereas some similar methods, such as Newton-Raphson, are not. Furthermore, the bisection method can pre-calculate the number of iterations to reach a certain maximum error once you know the initial interval size. This removes one calculation from each iteration of the loop which provides a slight advantage. To improve the performance of the algorithm, a better searching method could be used, such as Brent-Dekker or Fibonacci Search.

The complete algorithm is described compactly in Appendix A.

Implementation

We implemented the algorithm described in Appendix A using Python 2.7. The implementation is included in Appendix B. The tool can be configured via a separate config.py file, also included in Appendix B, so that the end user does not need to modify the algorithm code. The Cartesian coordinates of the intersections are printed to stdout so that another program, such as a graphics renderer, could take the coordinates as an input using a Unix pipe and display the points.

Testing and Results

For our algorithm to be deemed useful, we must consider how it compares to other equation solving software. We tried two well-known general math equations solvers, Maple and Mathematica. Mathematica was able to solve the system of equations, but took a significant amount of time and only returned a subset of the intersection points. Maple return a large quantity of meaningless garble and did not arrive at any solutions. Since neither equation solver was able to handle the equations properly, we tested our algorithm by graphing the sinusoid and line functions on a TI-83 calculator and using the local intersection functions. We used test cases from each category to ensure that our script worked properly. The results from these tests can be seen in Appendix C.

To see the results in 3 dimensions, we plotted the plane, helix, and intersection points using Mathematica. The intersections of the helix and the plane match up with the points we had found, as shown by the plot in Figure 4.

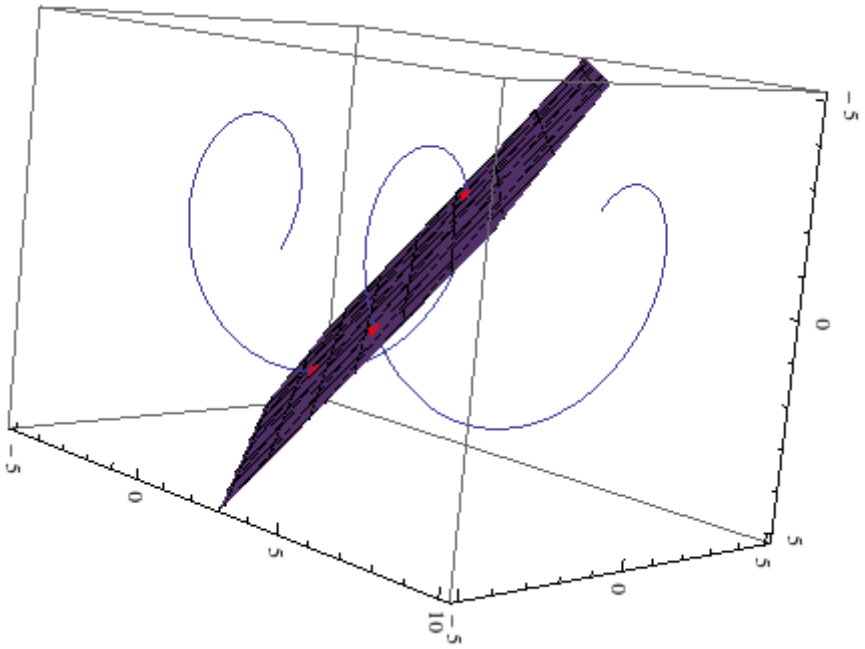

Figure 4: 3-D View of helix and plane with red dots marking the output of the algorithm

Further Studies

Although the constructed algorithm solves the model completely, many real world scenarios require extensions to our model and thus extensions to the algorithm. We considered some possible changes to the model and effects they would have on the analysis and the algorithm.

1. We assumed that both the plane and the helix are infinite. What would change if one or both were finite?

Given the plane and and helix, we can solve that with condition in it. For example, in our first case, we have infinite intersections. Based on the size of new limited plane, we can still use our algorithm to test the intersections. We may have less intersections, but the idea to determine the intersections is to add the multiple of the periods to find finite intersections. So, this implies that we can do the same ways to analyze other three cases.

2. What if our single helix has an elliptical shape?

Plane: $A\,(x-x_0)+B\,(y-y_0)+C\,(z-z_0)=0$

Helix: $x = R_1 cos(\omega t)$

$y = R_2 cos(\omega t)$

$$z = t$$

If we want to make our single helix with the shape of ellipse, we can just make R has different values for x and y. The reduction from a plane and a helix to a line and a sinusoid occurs in much the same way.

$$A\ (R_1\ cos(\omega t) - x_0) + B(R_2\ sin(\omega t)\ -\ y_0) + C(t - z_0) = 0$$

Move the components of the line to the right hand side

$$AR\ cos(\omega t) + BR\ sin = Ax_0 + By_0 + Cz_0 - Ct$$

Using the linear combinations trig identity,

$$\sqrt{(AR_1)^2 + (BR_2)^2}sin(\omega t + arctan2(AR_1, BR_2)) = Ax_0 + By_0 + Cz_0 - Ct$$

The difference here is that we cannot cancel out R_1 and R_2 in arctan2, since R_1 and R_2 are not equal. Besides that, R cannot pull out from the square root bracket. Essentially, stretching the circular helix into an elliptical one just changes the values of some constants in our equation for the sinusoid. The rest of the algorithm can easily be adjusted to reflect this change.

3. What if the helix does not move in the z-direction uniformly with time?

To compensate for this, we would only need to make minor adjustments to the algorithm. In each of our cases, we find t for the maximums and minimums using the equation (or some variation):

$$t = \frac{1}{\omega}(\ 2k\pi + \ arcsin(\frac{Ax_0 + By_0}{R\sqrt{A^2 + B^2}}) - arctan2(A, B)\ )$$

We would need to change from using ω, which is a constant to $\omega(t)$. As long as $t\ \omega(t)$ has an inverse, this will still be solvable for *t*. Our algorithm takes advantage of the fact that this function currently defines a sequence that increases at a constant rate. If we apply the inverse of $t\ w(t)$ to it, we may lose this property. However, the algorithm could still apply in certain cases where applying that inverse retains this property.

Bibliography

"Bisection method." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 1 Sep. 2014. Web. 8 Sep. 2014.

"Helix." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 5 Sep. 2014. Web. 8 Sep. 2014.

Appendix A - Algorithm

initialize parameters

if $C = 0 \; and \; |Ax_0 + By_0| > R\sqrt{A^2 + B^2}$ then

    return no solution

else if $C = 0 \; and \; |Ax_0 + By_0| \leq R\sqrt{A^2 + B^2}$ then

    construct sequence 1: $\frac{1}{\omega}(2k\pi + arcsin(\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}) - arctan2(A,B))$

    construct sequence 2: $\frac{1}{\omega}((2k+1)\pi - arcsin(\frac{Ax_0+By_0}{R\sqrt{A^2+B^2}}) - arctan2(A,B))$

    return (sequence 1 union sequence 2) to original model

else if $C \neq 0 \; and \; |\frac{C}{R\omega\sqrt{A^2+B^2}}| > 1$

    let $t_0 = min\{\frac{1}{C}(Ax_0 + By_0 + Cz_0 \pm R\sqrt{A^2 + B^2})\}$

    let $t_1 = max\{\frac{1}{C}(Ax_0 + By_0 + Cz_0 \pm R\sqrt{A^2 + B^2})\}$

    perform bisection method on interval [t0,t1]

    return resulting value to original model

else if $C \neq 0 \; and \; |\frac{C}{R\omega\sqrt{A^2+B^2}}| \leq 1$

    let t1= $min\{\frac{1}{C}(Ax_0 + By_0 + Cz_0 \pm R\sqrt{A^2 + B^2})\}$

    construct sequence: $\frac{1}{\omega}(2\pi k \pm arccos(\frac{-C}{R\omega\sqrt{A^2+B^2}}) - arctan2(A,B))$

    let t1 = greatest value of the sequence that is less than t0

    while true:

        if F(t0) has the same sign as F(t1) then

            if we have already found some solution then

                break loop, we have found them all

        else

            perform bisection method on interval [t0,t1]

            store result to be returned later

        set t0 = t1

        set t1 = next value from sequence

    return all solutions found

Appendix B - Implementation

To use this implementation, include the following two files in the same directory and run `python intersections.py`

Config File (`config.py`)

```
from math import *

# error tolerance for t
EPSILON=.00000001

# number of intersections to calculate if there are infinite
n=100

# If True, will output both the Cartesian coordinates and the t values
# If False, will output only the Cartesian coordinates
outputT = True

# Plane
# A(x - x0) + B(y - y0) + C(z - z0) = 0
# Helix
# x = R cos(w t)
# y = R sin(w t)
# z = t
A = 3
B = 4
C = -4
x0 = 2
y0 = 1
z0 = 1
R = 2
w = pi/2

A = float(A)
B = float(B)
C = float(C)
x0 = float(x0)
y0 = float(y0)
z0 = float(z0)
R = float(R)
w = float(w)
```

Appendix B (Continued)

Algorithm File (`intersections.py`)

```
#!/usr/bin/python
import sys
from math import *
from config import *

def bisection(a,b,f):
    midpt = (a+b)/2
    if ((b-a)/2<EPSILON):
        output(midpt)
    elif f(a) == copysign(f(a),f(midpt)):
        bisection(midpt,b,f)
    else:
        bisection(a,midpt,f)

def output(t):
    point = [R*cos(w*t),R*sin(w*t),t]
    if outputT:
        print t
    print point


# algorithm starts here
if (C == 0):
    if (abs(A*x0+B*y0+C*z0) > abs(R*sqrt(A*A+B*B))):
        print "none"
        exit(0)
    else:
        n = int(n)
        k = 0
        while k < int(n):
            t1 = 1/w * (2*k*pi+asin((A*x0+B*y0)/R/sqrt(A*A+B*B))-atan2(A,B))
            t2 = 1/w *
((2*k+1)*pi-asin((A*x0+B*y0)/R/sqrt(A*A+B*B))-atan2(A,B))
            if (t1<t2):
                output(t1) ; output(t2) ; k = k + 2
            elif (t1>t2):
                output(t2) ; output(t1) ; k = k + 2
            else:
                output(t1) ; k = k + 1
else:
    f = lambda t: R*sqrt(A**2+B**2)*sin(w*t+atan2(A,B))-A*x0-B*y0-C*z0+C*t
    period = 2*pi/w
```

```
amplitude=R*sqrt(A**2+B**2)
m = -C
b = A*x0+B*y0+C*z0
t0 = -amplitude / abs(m) - b / m
t1 =  amplitude / abs(m) - b / m

if abs(C/(R*w*sqrt(A**2+B**2))) <= 1:
    # extrema1 > extrema2
    extrema1 = 1/w*(acos(-C/(R*w*sqrt(A**2+B**2)))-atan2(A,B))
    extrema2 = 1/w*(-acos(-C/(R*w*sqrt(A**2+B**2)))-atan2(A,B))
    if (extrema2 > t0):
        while (extrema2 > t0):
            extrema1, extrema2 = extrema2, extrema1-2*pi/w
    elif (extrema1 < t0):
        while (extrema1 < t0):
            extrema1, extrema2 = extrema2+2*pi/w, extrema1
    if f(extrema1) == copysign(f(extrema1),f(extrema2)):
        extrema1, extrema2 = extrema2+2*pi/w, extrema1

    while f(extrema1) != copysign(f(extrema1),f(extrema2)):
        bisection(extrema2,extrema1,f)
        extrema1, extrema2 = extrema2+2*pi/w, extrema1
else:
    bisection(t0,t1,f)
```

Appendix C - Test Results

Case Type	No intersection	Infinite Intersections	Finite:		
R	1	3	3	2	2
A	3	3	3	3	3
B	4	4	4	4	4
C	0	0	2	3	-4
w	Pi/2	Pi/2	Pi/2	Pi/2	Pi/2
x0	2	2	2	1	1
y0	1	1	1	1	1
z0	4	4	4	1	1
Graphing Calculator					
	None	0.05489352	4.0493009	3.7100731	-2.222911
		1.1257754	5.2577616	5.9023312	0
			7.7011435	7	0.8279179
			9.6451656		
			11.384369		
			14.061988		
			15		
Our Algorithm (epsilon = 0.00000001)					
	None	0.054893525	4.049300938	3.710073137	-2.222910743
		1.125775416	5.257761621	5.902331165	3.17E-10
		8.054893525	7.701143508	7.000000006	0.8279179037
		9.125775416	9.645165588		
			11.3843686		
			14.06198839		
			15		

Figure 5: Comparison of Graphing Calculator Equation Solver and Single Helix Algorithm