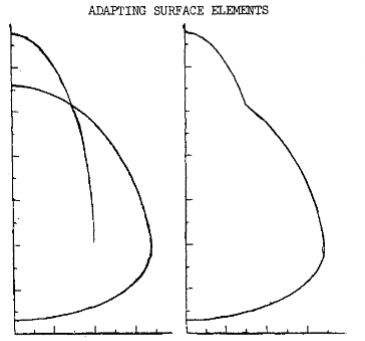
Introduction

I created a root-finding code to support my QUICK-GEOMETRY PROGRAM which was used to generate a detailed dynamic CAD model of the Space Shuttle.
The model is dynamic in that the program can deliver a complete cross-section at any point along the central axis of the model.

https://ntrs.nasa.gov/search.jsp?R=19760009728

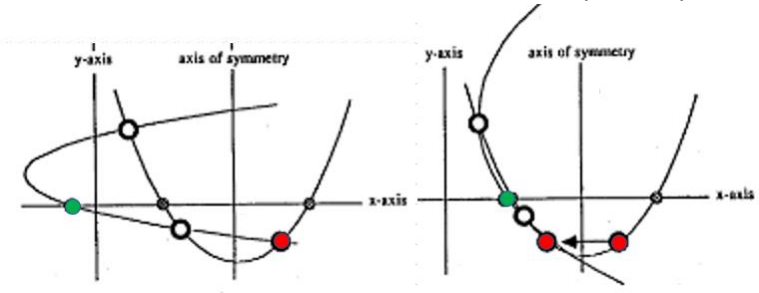
A featured element of the model required tracking the growth of the canopy as it emerges through the fuselage modeled by the intersection of two elliptical arcs.



My root-finding code follows the general algorithm defined by Brent, but with a geometric twist.
Link to a collection of the source codes

https://www.dropbox.com/s/df8tg90mcdzr03o/Collection%20of%20primary%20source%20codes%20for%20My%20Root%20Finder.txt?dl=0

I saw, that, that parabolic refinement would fail during the iteration cycles to refine the root, unless all the estimates fell on the same side of its axis of symmetry.



Mapping Estimates Across the Axis of Symmetry

In each iteration cycle, Point(x(2), y(2)) is the best estimate for the root. The axis of symmetry is located at the point where the local tangent has a zero slope.
Since the tangent of a parabola is a linear function, the location for that axis of symmetry can be calculated from tangents on either side of the Point(x(2),y(2)).

```
'-------------------------------------------------------------------
'Apply the mean value theorem: Secant Line is parallel to the Tangent Line
'-------------------------------------------------------------------
      If (x(3) - x(1) <> 0#) Then Secant_Left = (y(3) - y(1)) / (x(3) - x(1))
      If (x(3) - x(2) <> 0#) Then Secant_Right = (y(3) - y(2)) / (x(3) - x(2))
'-------------------------------------------------------------------
      x_Left = (x(1) + x(3)) / 2# 'located at mid point of the secant
      x_Right = (x(2) + x(3)) / 2# 'located at mid point of the secant
'-------------------------------------------------------------------
      If (Secant_Right - Secant_Left <> 0#) Then
            x_Symmetry = SECANT(x_Left, Secant_Left, x_Right, Secant_Right)
            x(5) = x_Symmetry
'-------------------------------------------------------------------
'  reflect x(1) and x(3) to the x(2) side of the symmetry axis
'-------------------------------------------------------------------
            For i = 1 To 3 Step 2
              Select Case True
                  Case x(2) > x_Symmetry And x(i) < x_Symmetry
                        xset(i) = x_Symmetry + x_Symmetry - x(i)
                  Case x(2) <= x_Symmetry And x(i) > x_Symmetry
                        xset(i) = x_Symmetry + x_Symmetry - x(i)
              End Select
            Next i
```

More details of the SECANT function will follow later.

The Algorithm is split into two parts: ESTNXT and SETNXT.

ESTNXT generates a new root estimate based on three previous estimates.
SETNXT integrates the new estimate with the previous estimates and returns the best three estimates for the next iteration.
The integration of ESTNXT and SETNXT can be seen from the code for the Solver.

```
'============================
Sub Get_Solution(ByRef Zero_Tolerance As Double, _
                 ByRef X_Root As Double, _
                 ByRef Y_Root As Double, _
                 ByRef Iteration_Count As Integer)
'============================
    Dim xval(5) As Double
    Dim yval(5) As Double
'-------------------------------------------
' Initialization - Collect Starting Points [x(1):x(2)]
'-------------------------------------------
    xval(1) = Range("X_1").Offset(1, 0)
    yval(1) = Get_Y_Value(xval(1))
    xval(2) = Range("X_2").Offset(1, 0)
    yval(2) = Get_Y_Value(xval(2))
'-------------------------------------------
    Range("Y_1").Offset(1, 0) = yval(1)
    Range("Y_2").Offset(1, 0) = yval(2)
'-------------------------------------------
' set point 3 by bi-section
'-------------------------------------------
    xval(3) = (xval(1) + xval(2)) / 2#
    yval(3) = Get_Y_Value(xval(3))
'-------------------------------------------
    Range("X_3").Offset(1, 0) = xval(3)
    Range("Y_3").Offset(1, 0) = yval(3)
'-----------------------
     Iteration_Count = 1
'-----------------------
Do Until Abs(yval(2)) <= Zero_Tolerance
'-----------------------
    Call ESTNXT(xval(), yval())
'-------------------------------------------
    yval(4) = Get_Y_Value(xval(4))
'-------------------------------------------
    Range("X_4").Offset(Iteration_Count, 0) = xval(4)
    Range("Y_4").Offset(Iteration_Count, 0) = yval(4)
'-------------------------------------------
    Call SETNXT(xval(), yval())
'-------------------------------------------
    Iteration_Count = Iteration_Count + 1
'-------------------------------------------
'Report iteration details
'-------------------------------------------
    Range("X_1").Offset(Iteration_Count, 0) = xval(1)
    Range("Y_1").Offset(Iteration_Count, 0) = yval(1)
    Range("X_2").Offset(Iteration_Count, 0) = xval(2)
    Range("Y_2").Offset(Iteration_Count, 0) = yval(2)
    Range("X_3").Offset(Iteration_Count, 0) = xval(3)
    Range("Y_3").Offset(Iteration_Count, 0) = yval(3)
    Range("X_4").Offset(Iteration_Count, 0) = xval(4)
    Range("Y_4").Offset(Iteration_Count, 0) = yval(4)
'-------------------------------------------
Loop
    X_Root = xval(2)
    Y_Root = yval(2)
End Sub
```

SETNXT – Housekeeping

```
'================================
Sub SETNXT(ByRef xval() As Double, yval() As Double)
'================================
' re-orders points for ESTNXT
' incorporate the new root estimate: (xval(4),yval(4))
' points 1 and 2 make the smallest interval bracketing the root
' point 2 is a better root estimate than point 1
' point 3 is closer to point 2 than point 4
'-----------------------------------------
' Initialization
' step zero - check that xval(4) is contained [xval(1):xval(2)]
'-----------------------------------------
' xval(4) falls outside [xval(1):xval(2)] set a new point 3 by bi-section and exit
'-----------------------------------------
            Case Sgn(yval(1)) = Sgn(yval(3)) And Sgn(yval(3)) <> 0  ' interchange pnt(1) and pnt(3)
            Case Sgn(yval(2)) = Sgn(yval(3)) Or Sgn(yval(3)) = 0 ' interchange pnt(2) and pnt(3)
'-----------------------------------------
' xval(4) falls inside [xval(1):xval(2)]: incorporate the new estimate
'-----------------------------------------
            Case Sgn(yval(1)) = Sgn(yval(4)) ' interchange pnt(1) and pnt(4)
            Case Sgn(yval(2)) = Sgn(yval(4)) ' interchange  pnt(2) and  pnt(4)
'-----------------------------------------
' step two - if xval(3) inside [xval(1):xval(2)]
'-----------------------------------------
' interchange of pnt(3) with pnt(1) or pnt(2) by matching SGN
'-----------------------------------------
               Case Sgn(yval(2)) = Sgn(yval(3)) ' interchange  pnt(2) and pnt(3)
               Case Sgn(yval(1)) = Sgn(yval(3)) ' interchange pnt(1) and pnt(3)
'-----------------------------------------
' step three - make pnt(2) a better estimate then pnt(1)
'-----------------------------------------
       If Abs(yval(2)) > Abs(yval(1)) Then 'interchange pnt(1) and  pnt(2)
'-----------------------------------------
' step four - makep nt(3) closer to pnt(2) than  pnt(4)
     If (xlng3 > xlng4) Then ' interchange pnt(3) and  pnt(4)
```

Evaluating the Inverse Quadratic Polynomial

The Lagrange formula for the Inverse Quadratic Polynomial is:

$$x = x_1\frac{(y-y_2)(y-y_3)}{(y_1-y_2)(y_1-y_3)} + x_2\frac{(y-y_1)(y-y_3)}{(y_2-y_1)(y_2-y_3)} + x_3\frac{(y-y_1)(y-y_2)}{(y_3-y_1)(y_3-y_2)}$$

Which reduces for x at y=0 to:

$$x_{y=0} = \frac{x_1y_2y_3}{(y_1-y_2)(y_1-y_3)} + \frac{x_2y_1y_3}{(y_2-y_1)(y_2-y_3)} + \frac{x_3y_1y_2}{(y_3-y_1)(y_3-y_2)}$$

Requiring 12 multiplications, 3 Divisions, 3 Subtractions and 3 additions.

Details of the SECANT function and its application to the evaluation Inverse Quadratic Polynomial.

Note the SECANT function is a modified instance of the FLINE function, used to solve for x when y=0. The FLINE function is simply a functional representation of the One Degree Lagrange Polynomial.

$$y = y_1\frac{(x-x_2)}{(x_1-x_2)} + y_2\frac{(x-x_1)}{(x_2-x_1)}$$

$$y = \frac{y_1(x_2 - x) + y_2(x - x_1)}{(x_2 - x_1)}$$

$$y = \mathrm{FLINE}(x_1, y_1, x_2, y_2, x)$$

```
'================================
Function FLINE(ByRef x_Left As Double, ByRef y_Left As Double, _
               ByRef x_Right As Double, ByRef y_Right As Double, _
               ByRef x_Fit As Double) As Double
'================================
   Dim x_Diff As Double
'--------------------------------------'
   x_Diff = x_Right - x_Left
   FLINE = 0.5 * (y_Left + y_Right)
   If (x_Diff <> 0#) Then
      FLINE = (y_Left * (x_Right - x_Fit) + y_Right * (x_Fit - x_Left)) / x_Diff
   End If
End Function
```

More dertails about FLINE and its connection to the Lagrange Polynomial and Aitken's Geometric Parabolic Construction can be found here:

https://www.linkedin.com/posts/alfredvachris_aitkens-graphical-construction-for-a-parabola-activity-6742091467516362752-_g34

Continuing with the SECANT function

Evaluating the One Dimensional Lagrange Polynomial for y=0

$$0 = y_1 \frac{(x - x_2)}{(x_1 - x_2)} + y_2 \frac{(x - x_1)}{(x_2 - x_1)}$$

$$x = \frac{y_2 x_1 - y_1 x_2}{y_2 - y_1}$$

$$x = \frac{y_2 x_1 - y_1 x_2 + y_2 x_2 - y_2 x_2}{y_2 - y_1}$$

$$x = x_2 - y_2 \frac{(x_2 - x_1)}{(y_2 - y_1)}$$

$$x = \mathrm{SECANT}(x_1, y_1, x_2, y_2)$$

```
Function SECANT(ByRef x_Left As Double, _
                ByRef y_Left As Double, _
                ByRef x_Right As Double, _
                ByRef y_Right As Double) As Double
'=============================================
' xsecant where 0=FLINE(x_Left,y_Left,x_Rright,y_Right,x_Secant)
'---------------------------------------------
    SECANT = x_Right - y_Right * (x_Right - x_Left) / (y_Right - y_Left)
End Function
```

ESTNXT – Inverse Quadratic Solution

```
'-----------------------------------------------------------------
' estimate x(4)=(p*y+q)*y+r at y=0. Using the Aitken Construction
'-----------------------------------------------------------------
            xl = (xset(3) + xset(1)) * 0.5
            If (y(1) - y(3) <> 0#) Then xl = SECANT(xset(3), y(3), xset(1), y(1))
            xr = (xset(3) + xset(2)) * 0.5
            If (y(2) - y(3) <> 0#) Then xr = SECANT(xset(3), y(3), xset(2), y(2))
'-----------------------------------------------------------------
            x(4) = (xl + xr) * 0.5
            If (y(2) - y(1) <> 0#) Then x(4) = SECANT(xl, y(1), xr, y(2))
    End If
```

Using SECANT to evaluate the Inverse Quadratic Polynomial

Requiring only 6 multiplications, 3 Divisions, and 9 Subtractions.