Statics with Robotics to Get the Least-squares Fit of Profiles for Evaluating FEA Simulations of Flexible Components and Assemblies

by

Sai Chandu Sunkara

A Thesis Presented in Partial Fulfillment

of the Requirements for the Degree

Master of Science

Approved April 2023 by the

Graduate Supervisory Committee:

Joseph Davidson, Chair

Jami Shah

Yi Ren

ARIZONA STATE UNIVERSITY

May 2023

ABSTRACT

Least squares fitting in 3D is applied to produce higher level geometric parameters that describe the optimum location of a line-profile through many nodal points that are derived from Finite Element Analysis (FEA) simulations of elastic spring-back of features both on stamped sheet metal components after they have been plasticly deformed in a press and released, and on simple assemblies made from them. Although the traditional Moore-Penrose inverse was used to solve the superabundant linear equations, the formulation of these equations was distinct and based on virtual work and statics applied to parallel-actuated robots in order to allow for both more complex profiles and a change in profile size. The output, a small displacement torsor (SDT) is used to describe the displacement of the profile from its nominal location. It may be regarded as a generalization of the slope and intercept parameters of a line which result from a Gauss-Markov regression fit of points in a plane. Additionally, minimum zone-magnitudes were computed that just capture the points along the profile. And finally, algorithms were created to compute simple parameters for cross-sectional shapes of components were also computed from sprung-back data points according to the protocol of simulations and benchmark experiments conducted by the metal forming community 30 years ago, although it was necessary to modify their protocol for some geometries that differed from the benchmark.

ACKNOWLEDGEMENTS

My most sincere appreciation goes to my advisor and committee chair, Dr. Joseph K. Davidson, for his guidance, support, and belief in me, which made this work possible.

I would also like to thank other committee members, Dr. Jami J. Shah and Dr. Yi Ren for their time and guidance.

I would also like to thank former student Abhishek Joshi who undertook the Finite Element Analysis simulation for a J-section welded profile and who did the preliminary work to extract from it the data points for fitting.

Finally, I wish to acknowledge my appreciation for funding provided by the National Science Foundation, GOALI Award No. 2029905.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figures Page

Figures Page

Figures Page

LIST OF NOMENCLATURE

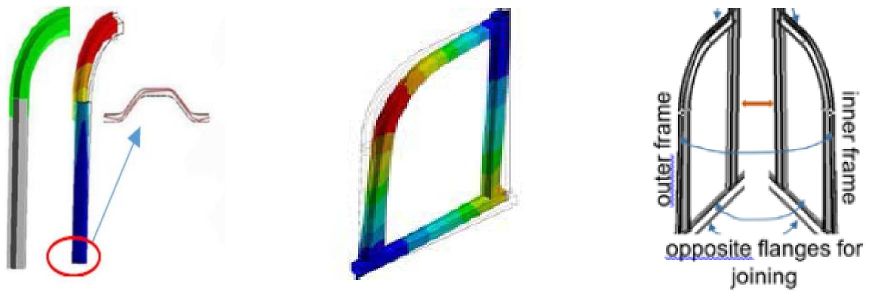
SDT:	Small Displacement Torsor
LSF:	Least Squares Fit
FEA:	Finite Element Analysis

CHAPTER 1. Introduction and Problem Statement

Large assemblies, such as an automobile body or door, are made from flexible parts, usually sheet metal stampings. Each assembly is made of many flexible subassemblies that are assembled and joined progressively. These subassemblies are also built progressively, as shown in Figure 1.1. When two individually stamped parts are brought together to be joined into a subassembly, they often do not match up exactly, and so special tooling and clamping are required to bring them into alignment. Thus, for good design, variable gaps between proximal assemblies must be predicted precisely and be related to input variables to be controlled. As subassemblies of parts are stacked, errors accumulate to cause larger variations in the gaps. Sources of manufacturing variations in sheet metal assemblies include non-isotropic material properties from cold rolling, spring back from stamping, and distortion from residual stresses when components are clamped, then spot welded. The workflow of FE simulations to capture these sequential events are: (1) component stamping that captures spring back, (2) clamping of the components for assembly, and (3) assembly joining, then release to the free state.

Variations in the gaps between assemblies depend on the geometric spring back variations of nodal points along each of two matching line profiles, such as the edge of an auto door and its matching opening in the body. As a first step in producing extracted geometric parameters related to a gap, this thesis utilizes data along the profiles located at the extreme longitudinal edges for the simple assembly shown in Fig. 1.1. (See the highlighted edges in Fig. 1.2(A).) The data are from the FEA analysis conducted by A. Joshi [1] in 2020. In addition to getting parameters for the two J-shaped profiles (edges) of the assembly, data for each profile were partitioned so points confined to a straight part could be used to simulate spring-back for the edges of a straight component, and points confined to an arc could be used to simulate spring-back, including change in radius, for the edges of an arc component. Spring-back for each of these six profiles is quantified with a zone magnitude and with six coordinates of small displacement (Fig. 1.2(B)) to give its overall geometric location relative to its nominal location (zero spring-back). Two zone magnitudes were computed: a zone constrained to the location of the LS profile, and an unconstrained zone. (See Chap. 5.)

The parameters are being used as part of curated data that represent a design space for flexible two-part assemblies which will be available for machine learning (ML) algorithms so optimal designs of subassembly fixtures and weld patterns can be predicted computationally, instead of empirically (Adrian, *et al.*, 2022) [2]. The parameters will also be used in the future for more elaborate assemblies [3].

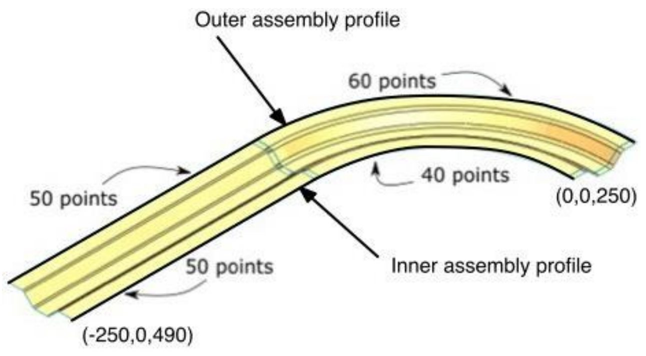


Simple Assembly   Closed Loop Assembly   Matched Pair Assembly

Fig 1.1 Common Types of Assembly Structures

Some simple parameters for cross-sectional shapes of components were also computed from sprung-back data points according to the protocol of the NUMISHEET

93 simulations and benchmark experiments [4], although it was necessary to modify the protocol for some of our geometries that differed from the one in NUMISHEET 93. (See sect. 6.1.)



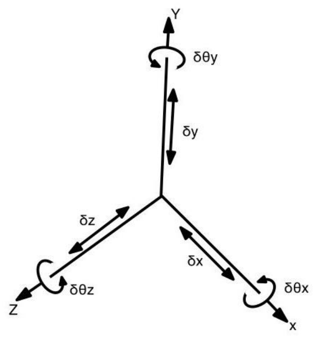


Fig 1.2(A) Nominal Dimensions of Assembly, mm   Fig 1.2(B) Small Displacement Coordinates

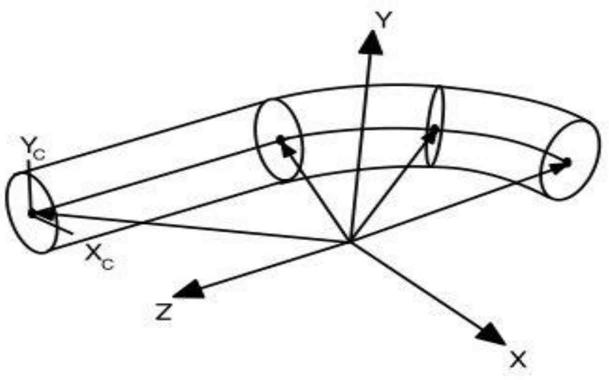


Fig 1.2(C) LSF Zone of the Profile

1.1 Problem Statement and Methods

The novel contribution in this thesis is the application of virtual work, in combination with parallel (platform) robotics to construct the set of linear equations for which the least squares (LS) best-fit solution may be obtained. This construction allows for both size change and the line-profiles to be of any shape. For any profile, the linear equations in the LSF formulation are derived from force balances applied to the virtual work expressions at the linear actuators of a parallel-actuated robot. Every actuator is linear, i.e., can only extend or contract, and so is represented geometrically with the six coordinates of a line [5]. Inputs to the fitting are the deviations of the sprung-back nodal points from the nominal profile (shape with zero spring-back). The output is a small displacement torsor (SDT) that describes the displacement of the profile from its nominal location. The SDT coordinates $\$ = \left(\delta\theta_x\ \delta\theta_y\ \delta\theta_z\ \delta x\ \delta y\ \delta z\right)$ represent the small displacement of the robot platform that has etched in it the profile of interest. The coordinates also are a generalization of the slope and intercept parameters of a line which result from a Gauss-Markov regression fit of points in a plane. The relatively small displacements of different profiles, e.g., opposite edges on a component or subassembly, may be used to correlate different combinations of inputs (e.g., material thickness, strength, blank orientation, and both spot weld locations and quantity) with the final shape of the component or subassembly in its free state.

By converting the SDT to an equivalent transformation matrix, it is then used to express the coordinates of the sprung-back nodal points in a new reference frame that is aligned with the LS profile. From these new coordinates, the magnitude of a LS envelope, centered on the LSF profile, is computed that just captures all the sprung back points. As an alternative, we also computed the magnitude of the true minimum zone using the minimum circumscribing cylinder algorithm from Mohan, *et al*. [6].

1.2 Literature Review

There has been a lot of development and research going on the fitting of the various shapes like polynomial curves, circles, planes, cylinders, spheres [7,8,9,10], and other surfaces [11] to an array of points. One of the ways to fit a straight line in 3D space is using the Singular Value Decomposition (SVD) which gives the average point and direction of a fitted straight line [12]. These methods work well when the point sampling is controlled and can be made uniform but when dealing with non-uniformly sampled points this doesn't work well, so there is a need to incorporate the weighted least squares fit that accounts for that discontinuities in the sampled points [13]. An example 3D fitting application, which is relevant to this NSF project [3], is included in [14]: the fitting of an auto body opening (the profile) to points measured around a prospective door in order to determine optimum hinge-mounting adjustments when assembling the door to the auto body.

Change in size of an arc or a closed planar profile was included with least-squares fitting by Davidson, *et al*. [15,16,17]. The linear equations to be solved were created using virtual work applied to parallel actuated platform robots, the method used in this thesis (see section 5.1). When the Moore-Penrose inverse (pseudoinverse) [12,17] is applied to the matrix form of these equations, the optimum LS solution is the set of small displacement torsor (SDT) coordinates that represent the displacement of a point on the robot platform and its angular orientation.

Additional parameters that will be important potentially are the true minimum zone for a set of points arrayed along a straight line. Both the zone magnitude and the location of the corresponding axis are of potential use. The method for getting these results is iterative, but it begins from the LSF of a line to the points [6]. Further, by transforming points along a curved profile to a line, the method may be applied to line-profiles containing curved portions also (See Sect. 5.2- Fig 5.1).

This thesis also contains software for computing geometric parameters to validate the Ohio State University FEA simulations [2] for stamping hat-section components. Verification was achieved when the parameters agreed with experimental values presented in the NUMISHEET 1993 U-draw/bending benchmark [4]. Since the sprung-back points defining the radius parameter were often ill-conditioned, a refined algorithm [18] was used for it.

CHAPTER 2. Plücker Coordinates of a Line and Coordinates of a Small Displacement Torsor (SDT)

The entities lines, screws, wrenches, and small displacement torsors (SDTs) are used in the following chapters both to characterize locations of LSF profiles and to create the linear equations needed for a matrix optimization. All may be represented in 3D space with two vectors, or six scalar coordinates (L, M, N; P, Q, R). For all four of them, the first vector gives the direction i.e., a spatial field of parallel lines. When a line is represented, the two vectors are at right angles and the second vector selects a specific line from the spatial field. When a screw is represented, the second vector P^*, Q^*, R^* contains additional information (vector τ in Fig. 2.1) and so violates the right-angle property. For wrenches and SDTs, the entire set of six coordinates also contains amplitude information (Table 2.1 So, one of the prominent ways to do this is to use Plücker coordinates. The Plücker coordinates (L, M, N; P, Q, R) are usually used to locate a line or a screw axis in space. They comprise a pair of three-dimensional vectors of which the first vector (L, M, N) determines the direction of the axis, and the second vector (P, Q, R) locates the line in space. Table 2.1 shows the different interpretations of these coordinates as used in this thesis.

Table 2.1 Entities That Can Be Interpreted Using the Plücker Coordinates Method. [5]

Entity	**Symbol**	**Coordinates**
Line	\$	(L, M, N; P, Q, R)
Screw	\$	(L, M, N; P^*, Q^*, R^*)
Wrench	F \$	F (L, M, N; P^*, Q^*, R^*)
Small Displacement Torsor (SDT)	δθ\$	$\delta\theta$ (L, M, N; P^*, Q^*, R^*)

We are currently using the wrench coordinates to represent forces, line coordinates to represent actuators, and SDT coordinates to describe the displacement of the profile from its nominal location (Chap. 3).

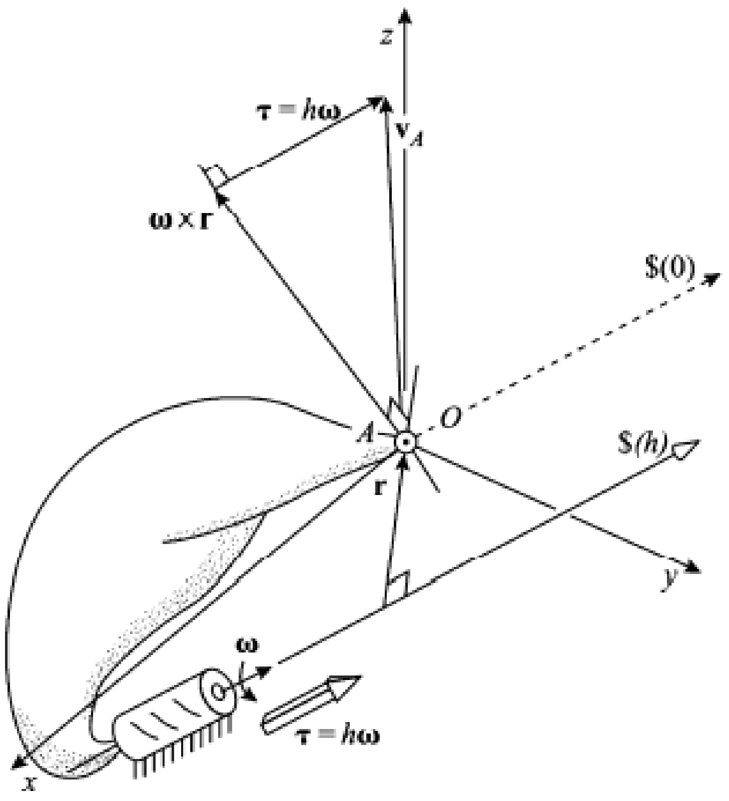


Fig 2.1 Line Coordinates [5].

In general, the (P, Q, R) coordinates of the SDT will include the vector parallel to the twist $(*h*) = $\delta\theta$$ = $\delta\theta$ (L, M, N; P*, Q*, R*) in Fig. 2.1. Therefore, using equations from [5], pitch *h* may be extracted from the coordinates as the ratio,

$$h = \frac{LP^* + MQ^* + NR^*}{L^2 + M^2 + N^2}$$

and amplitude $\delta\theta = (L^2 + M^2 + N^2)^{1/2}$. (2.1)

It often is helpful to identify the location of the line that carries the SDT. This may be found by computing $P = P^* - hL$, $Q = Q^* - hM$, and $R = R^* - hN$ for the line, then using twice the condition

$$\begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \begin{bmatrix} L \\ M \\ N \end{bmatrix} \tag{2.2}$$

that a point (*x*, *y*, *z*) lies on the line (L, M, N; P, Q, R). Helpful points lie on the coordinate planes *x*=0, *y*=0, and *z*=0.

CHAPTER 3. Least Squares Fits using Virtual Work Applied to Platform Robots.

Several geometric characteristics of nodal points from FEA simulations of plastic forming and elastic spring-back may be represented by using, or in some cases extrapolating, measures and methods already established for scanned points on surfaces of machined parts. The methods that are used in this thesis are (1) the computation of a minimum zone, which just captures all partitioned set of points; (2) the substitute feature associated with the zone; (3) least-squares fits (LSFs) of substitute features for the same points; and (4) the boundaries parallel to them which just capture all the points. The computations for minimum zones and their associated substitute features follow the methods from metrology described in Mohan, *et al*. [6].

However, several features for the components, and for assemblies of them, are *line* profiles, some of them with changing curvature. Examples are the free edges of components and assemblies. For these reasons, we have not found a satisfactory LSF model. One approach to this was undertaken by Nassef & ElMaraghy, [14] in which they minimized the squares of the closest proximities of pairs of opposite points on two similar 3D line profiles, one on the edge of the door of an automobile and the other around the corresponding opening in the auto body. The resulting profile was used to optimize the mounting location for the door. It is not clear from the paper whether these distances were treated as scalar values or vectors, but, as seen in what follows, for points arranged in 3D around a nominal profile, the closest distances need to be treated as vectors [7,8,9,10]. To incorporate the vector feature in combination with changing in curvature in this thesis, an alternative method is suggested: virtual work applied to a platform robot that is customized to the specific profile [15,16,17]. The method also allows the fitted profile to include change in size.

3.1 Planar Example to Validate the Virtual Work Method.

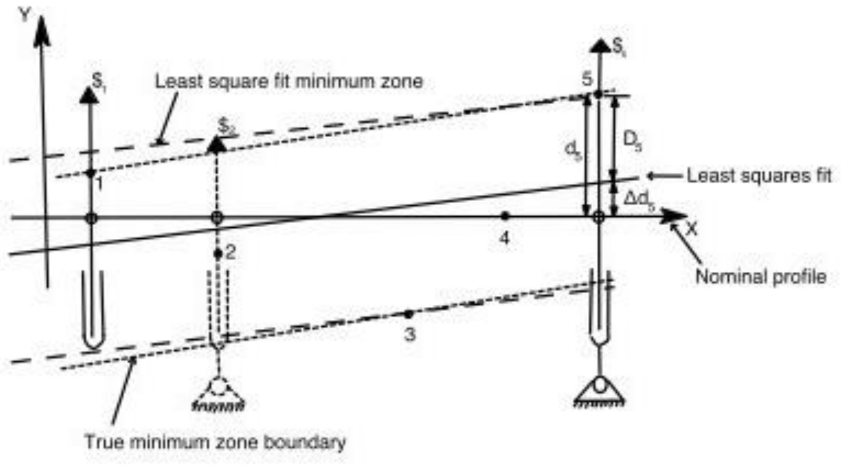


Fig 3.1 The Regression fit of line in 2D using line coordinates.

To validate LSFs using the method of virtual work applied to a parallel robot, consider the slope fit of a straight line to the five nodal points shown in Fig. 3.1 and which have coordinates listed in Table 3.1. The five nodes are shown deviating from the *x*-axis,

Table 3.1. The Uniformly Distributed Sample Points for Fitting a Line in 2d Space.

Points	1	2	3	4	5
X, mm	1	5	8	10	12
Y, mm	1	-1	-2	0	2

here taken to be the nominal profile shape (straight) and location. Their positions are fixed in the *XY*-plane. Coinciding with the nominal profile is a line etched in the platform of a virtual planar robot that is actuated redundantly and in-parallel with five actuators, one for each of the five points. Each actuator is linear (can only extend or contract), acts through one of the points, exerts force between the base and the platform (hence, act in-parallel on the platform), and, for small displacements of the platform, exerts force only in the *y*-direction. For the geometry in Fig. 3.1, the platform displacement is constrained to be only the rotation in the plane $\delta\theta_z$ and the vertical translation δy of that point on the platform initially coincident with the origin *O*. Also, all the actuators are connected to both the platform and the base with rotary joints except for the first actuator, which is rigidly attached to the base (Fig. 3.1).

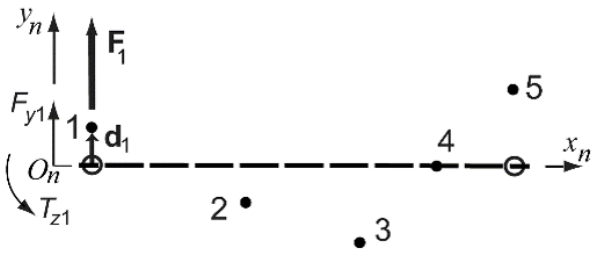


Fig 3.2 Force Balance for the Actuator at one Nodal Point

Each actuator, represented by its line $\$_i$, exerts a force of magnitude F_i on the robotic platform, and, when each act alone, the platform is a two-force member and exerts the same force on the environment. Therefore, presuming temporarily that the actuator at $\$_1$ acts alone and the other actuators are unconstrained, the force $F_1\$_1$ is the same as the wrench (T_{z1} F_{y1}) that the platform exerts on the environment (Fig. 3.2), where $F_{y1} = F_1$ and $T_{z1} = x_1F_1$. Putting this together $F_1 * \$_1 = [T_{z1}\ F_{y1}] = F_1[x_1\ 1]$, and the force magnitude cancels to leave $\$_1 = [x_1\ 1]$ to represent $\$_1$. When this result is compared to the representation of a wrench in chapter 2, $R_1 = x_1$ and $M_1 = 1$ are the normalized line coordinates for $\$_1$ in $F_1\ \$_1 = F_1\ [P_1\ Q_1\ R_1;\ L_1\ M_1\ N_1]$. Since all the nodal points lie in a plane and the x-axis of the coordinates system is aligned with the nominal profile in Fig.1, $P_1 = Q_1 = L_1 = N_1 = 0$. Resuming the special case when the actuator at $\$_1$ acts alone, the input virtual work is the scalar product F_1 d_1 because the vector deviation d_1 is measured on the line of action for F_1. Further the output virtual work is the matrix product of the wrench representation of $F_1 = \left[T_{z1}\, F_{y1}\right] = F_1[x_1\ 1]$ and the SDT [\$] = $[\delta\theta_z\ \delta y]^T$, where $\delta\theta_z$ is the rotation of the platform and δy is the y-displacement of the point on the platform initially coincident with the origin of coordinates. Equating these two forms of virtual work gives $F_1\, d_1 = F_1\ [x_1\ 1]\ [\delta\theta_z\ \delta y]^T$ at the actuator $\$_1$. Cancelling force F_1 now leaves the geometric equation $d_1 = [x_1\ 1]\ [\delta\theta_z\ \delta y]^T$. Corresponding equations occur for the remaining nodes. So, the set of linear equations that formulate the least-squares fit profile are.

$$[\mathbf{d}_\mathrm{i}] = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \equiv [\mathbf{K'}][\$] = \begin{bmatrix} R_1 & 1 \\ R_2 & 1 \\ \vdots & \vdots \\ R_n & 1 \end{bmatrix} [\$] \qquad (3.1)$$

where the $[d_i]$ are the deviations measured at right angles to the nominal profile, which is the *x*-axis, the matrix [k’] lists the unit forces acting along the linear actuators and the moment of each about the origin of coordinates (each pair representing the normalized line coordinates of a $\$_i$), and [\$] represents the SDT of the platform. Since, the fine linear equations (3.1) are superabundant and inconsistent set of linear equation, so there is no single vector [\$] that satisfies them all. Hence, we go for the least squares fit solution, $\$_{LS,}$ which minimizes the sum of the deviations of the points from the least squares profile. This best possible vector for $[\$_{L\text{-}S}]$ can be found by the pseudoinverse equation [12]

$$[\$_{\text{L-S}}] = [k']^{\#}[d] = (([k']^T[k'])^{-1}[k']^T)[d] \tag{3.2}$$

Table 3.2. The Inputs and the Results of 2d Least Squares Fitting Using Line Coordinates.

Point	R_i = x_i	M_i	d_i (mm)	Δd_i (mm)	D_i (mm)
1	1	1	1	-0.331	1.331
2	5	1	-1	-0.117	-0.882
3	8	1	-2	0.043	-2.043
4	10	1	0	0.150	-0.150
5	12	1	2	0.257	1.7433

After substituting R_i, M_i, and d_i from Table 3.2 into Eqs. (3.2), we get the $\$_{\text{L-S}}$ value to be ($\delta\theta z$ δy) = (3.0639, -0.385). From the values of the $\$_{L-S}$ found from solving the above equations we get a new set of deflection values Δd with which we can find the change in deflections D by finding the difference between the original deflections d and the deflections found from the least squares fit result Δd.

One measure of zone magnitude is the LSF minimum zone that is obtained as the perpendicular distance between the pair of lines parallel to the LSF line (Fig. 3.1) that just capture all the nodal points. This zone may be found in two stages. First, when the SDT [$\$_{\text{L-S}}$] is substituted into Eq. (3.1), the new values of d_i that result, here called Δd_i, measure the distances between the LSF profile and the nominal profile at the respective nodal points (Fig. 3.1). The accuracy of these distances presumes that the original deviations d_i are small relative to profile dimensions, an assumption that is valid for the FEA springback data used in this thesis. (It is not valid for the example shown in Fig. 3.1, but this example is included so definitions and terminology may be shown visually nearly at scale.) Second, it evident in Fig. 3.1 that the deviation D_i from the LSF profile at each nodal point i is $D_i = d_i - \Delta d_i$. Therefore, the L-S minimum zone MZ $_{\text{L-S}}$ is found to be the difference between the algebraically largest and smallest values of D_i which are obtained from the array of values at all the points, i.e.

$$\text{MZ}_{\text{L-S}} = D_{i,max} - D_{i,min} \,.$$

For the nominal profile and five nodal points shown in Fig. 1, MZ $_{\text{L-S}}$ =3.786 mm.

CHAPTER 4. Fitting of Line-profiles in 3d

In this study we are fitting the sprung back points along two profiles from the FEA simulations of component stamping and subsequent spot welding into a J-Shaped assembly (Fig. 4.1). The image below are the parts that were simulated by A. Joshi [1]. The two profiles are the outer line-profile that includes the larger arc with 60 points (59 intervals) and the inner profile that contains the smaller arc with 40 points (39 intervals). The number of nodal points on each edge of the profile are shown in Figure 4.1, and these are distributed uniformly along each profile. The data is taken from [1] for the case of five spot welds made one at a time, progressing from the smaller arc to the larger arc. Each weld is in one of the five straight portions of the hat-shaped cross-section.

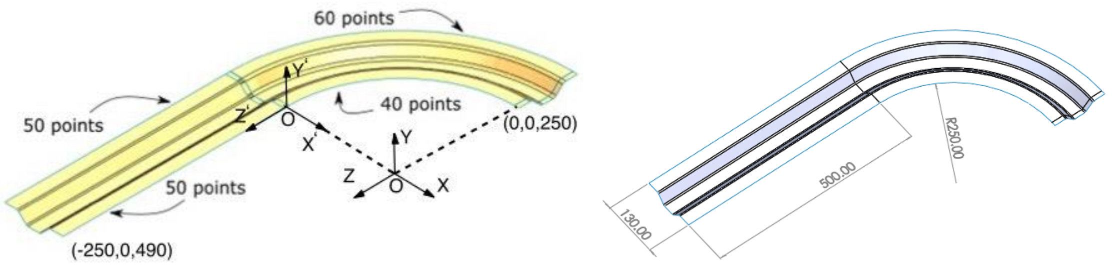


Fig 4.1 The Partitioned Points from the Assembly of Straight and Curved Hat-section Components and the Global Coordinate Frame. The Ending Coordinates Are for Zero Spring-back (Nominal Profiles) [1].

All the points in this chapter are taken from the J-shaped spot-welded assembly in Fig. 4.1. To apply the statics with robotics method for generating the linear equations, the points of the two profiles are partitioned into sets of points for edges of the straight hat section and sets for the inner and outer arcs. Then, for the assembly, these are combined for each profile. The straight component is 500mm long, but to simulate the spot-welding of the two components together for the assembly, there is an overlap of 10mm along both profiles. Consequently, there are 49 intervals for the straight profiles, and in the coordinate-frame shown in Fig. 4.1, the points at the ends are z = 0 and z = 490 mm.

Three least-squares fits were undertaken for each profile: the straight part, to simulate fitting edges to a component; the arc, another component, and the full J-shape for the welded assembly. Note, however, that the 50th point for each straight profile is also the first point for the respective arc. Therefore, for fitting each of the J-shaped assembly profiles, this duplicated point is used just once.

Two quantities of points were used for the same data: one was to use all 89 points for the inner profile containing the smaller radius arc and 109 points for the outer arc. The other quantity, to reduce computation time for use with large amounts of simulation data, was to use a sampling method that leads to 19 points for the inner profile and 23 points for the outer profile. For both profiles with sampling, 11 points were used for the straight portions with nine intervals of nominally 50mm each and the last interval (at the 490mm limit) of 40mm.

4.1 Straight Line Fitting in 3d Space with Least Squares

For points arranged along a nominal straight line-profile, such as those shown in Fig 4.2, the displacement of the LSF profile will be in the x- and y-directions with allowance for rotations about these axes also. Consequently, actuators are required in these two directions (Figs. 4.2 and 4.3) at each nodal point to produce the required displacements of the robot platform that carries the duplicate movable line-profile.

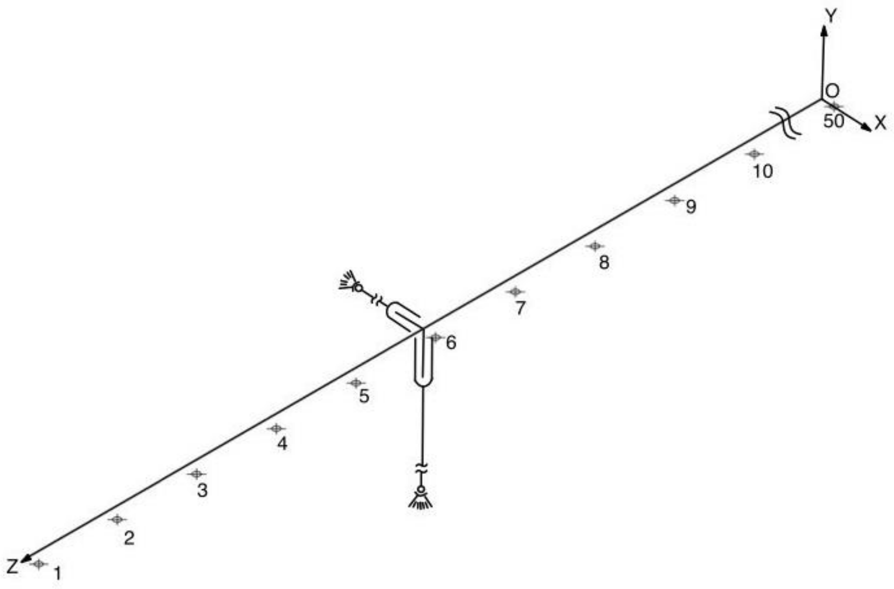


Fig 4.2. The Regression Fit of Line in 3d.

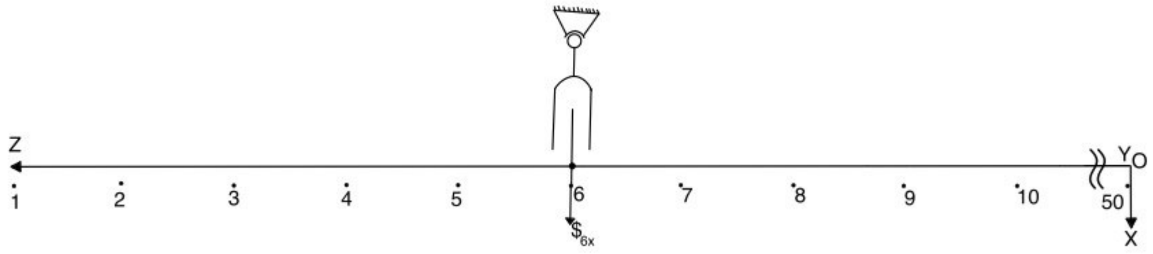


Fig 4.3(a) The XZ - Plane View of the Straight Line Profile

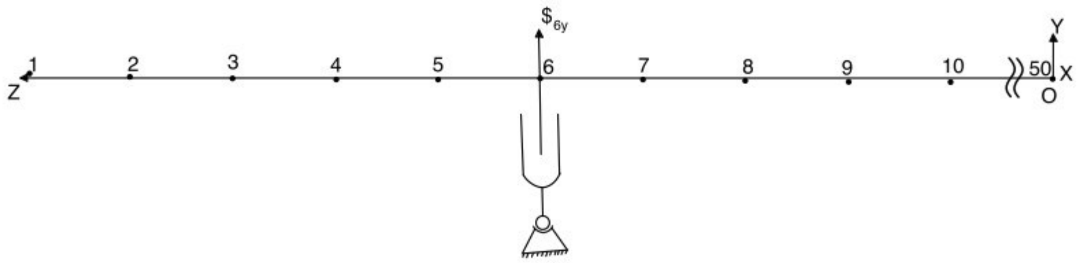


Fig 4.3(b) The YZ - Plane View of the Straight Line Profile

To capture all the deviations in 3D we need to find the line coordinates (P, Q, R, L, M, N) for each of the two perpendicular linear actuators in Fig. 4.2. The results are (0, Z_i, 0, 1, 0, 0) and (-Z_i, 0, 0, 0, 1, 0) for actuators lying in the XZ and YZ planes, respectively. Since the R and N coordinates are consistently zero, they may be ignored, and we get (P, Q, L, M) = (0, Z_i, 1, 0) and (-Z_i, 0, 0, 1) as the line coordinates for all the actuators at the nodal points along the straight portion of the profile.

Table 4.1. Sample Points from the Inner Sprung Back 3d Line Profile

Points	X, mm	Y, mm	Z, mm	d_i, mm
1	-241.90	0.743	490.02	8.10
2	-241.84	0.535	451.02	8.16
3	-241.79	0.303	401.03	8.21
4	-241.75	0.010	351.03	8.25

Table 4.2. Sample Entries for Matrix [K'] along the Inner Straight Profile

Points	P	Q	L	M	d_i
1x	0	490.2	1	0	8.10
1y	-490.2	0	0	1	0.743
2x	0	451.02	1	0	8.16
2y	-451.02	0	0	1	0.535
3x	0	401.03	1	0	8.21
3y	-401.03	0	0	1	0.303
4x	0	351.03	1	0	8.25
4y	-351.03	0	0	1	0.010

Using Fig 4.3(a) for actuators in the *ZX*-plane, we can deduce the virtual work balance equations at each node to be:

$$\begin{aligned} F_{ix}d_{ix} &= [0 \quad T_{iy} \quad F_{ix} \quad 0][\Delta\theta_x \quad \Delta\theta_y \quad \Delta x \quad \Delta y]^{\mathrm{T}} \\ &= F_{ix}[0 \quad z_i \quad 1 \quad 0][\Delta\theta_x \quad \Delta\theta_y \quad \Delta x \quad \Delta y]^{\mathrm{T}} \end{aligned} \tag{4.1}$$

And using Fig. 4.3(b) for actuators in the *YZ*-plane, each is

$$\begin{aligned} F_{iy}d_{iy} &= [T_{ix} \quad 0 \quad 0 \quad F_{iy}][\Delta\theta_x \quad \Delta\theta_y \quad \Delta x \quad \Delta y]^{\mathrm{T}} \\ &= F_{iy}[-z_i \quad 0 \quad 0 \quad 1][\Delta\theta_x \quad \Delta\theta_y \quad \Delta x \quad \Delta y]^{\mathrm{T}} \end{aligned} \tag{4.2}$$

These Eq. (4.1) and Eq. (4.2) at each node can be formulated into the single matrix expression.

$$[\mathbf{d}_{\mathrm{i}}] = \begin{bmatrix} d_{1x} \\ \vdots \\ d_{ix} \\ d_{iy} \\ \vdots \\ d_{ny} \end{bmatrix} \equiv [\mathbf{K'}][\$] = \begin{bmatrix} 0 & Z_1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & Z_i & 1 & 0 \\ -Z_i & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -Z_n & 0 & 0 & 1 \end{bmatrix} [\$] \tag{4.3}$$

that captures all the linear equations. The matrix [d] is a list of deviations measured at right angles in x- and y-directions from the nominal profile at the respective nodal points, the [**K'**] gives the line coordinates of the nodal points, and $[\$]_{\mathrm{SDT}} = [\delta\theta\mathrm{x}, \delta\theta\mathrm{y}, \delta\mathrm{x}, \delta\mathrm{y}]^{\mathrm{T}}$ is the SDT of the platform. Sample values of these are given in the table [4.2]. When these and all the remaining values are substituted in Expression (4.3), we end up with this superabundant and inconsistent set of linear equations formed from all the nodal points. There is no single vector [\$] that satisfies all the linear equations (4.3). However, we can get an optimum least-squares solution, $[\$_{\mathrm{L\text{-}S}}]$, by using the pseudoinverse $[\mathbf{K'}]^{\#}$ of the rectangular matrix [**K'**], defined as

$$[\$_{L-S}] = [\mathbf{K'}]^{\#}[d] = (([\mathbf{K'}]^T[\mathbf{K'}])^{-1}[\mathbf{K'}]^T)[d] \tag{4.4}$$

For the 11 points of the sample data along the straight part of the inner profile, we get the $\$_{\mathrm{L\text{-}S}}$ value to be $[\delta\theta_x \quad \delta\theta_y \quad \delta x \quad \delta y]$ = (-0.054, -0.045, 8.516, -0.140) in units of radians and mm. Using these we can find (see Chap. 2) the magnitude of rotation $\delta\theta = \left(\delta\theta_x^2 + \delta\theta_y^2\right)^{1/2}$ = 0.070 radians and pitch h = -91.808 mm/rad for the small displacement torsor (SDT). Then the axial progression of the platform along $\$_{\mathrm{L\text{-}S}}$ is $s = h\,\Delta\theta$. (See

Chap. 2) Using these we can find the line coordinates (L, M, N; P, Q, R) on which the SDT lies. (See Chap. 2 and Tables 7.7 – 7.10.)

In Section 5.1 the SDT coordinates are converted to a 4x4 homogeneous transformation matrix [5] that is used to transform the nominal profile to its least squares location relative to the original sprung back points. This result will then be used to find the boundaries for the least squares zone and true minimum zone that just captures all the points.

4.2 Least Squares fitting of an arc profile in 3D space.

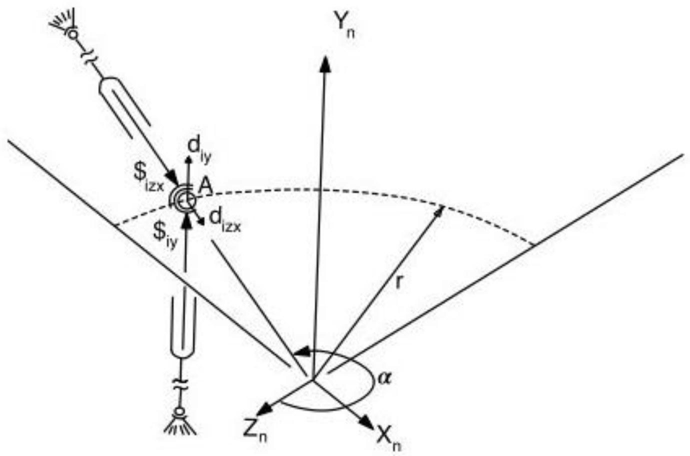


Fig 4.4. The arrangement of actuator for an Arc Profile

When the coordinate system in Fig. 4.5 is used to represent the arc, the two actuators normal to the nominal profile at a sprung-back data point must lie in a plane that contains both the y-axis and the point. We will use the arc of nominal radius 250 mm on the inner J-shaped profile as an example (Fig. 4.1). To formulate the linear equations, we first need angle α as

$$\alpha = \text{atan2}(x_i/z_i) \tag{4.5}$$

where both the *x* and *z* arguments of the atan2 function are signed. The necessary components of each deviation vector $\mathbf{d} = \overrightarrow{AB}$ (Fig. 4.5) are obtained from its projections onto the *zx*-plane and on the *y*-axis. Now the corresponding torques and forces in each row (P, Q, R; L, M, N) of matrix [**K'**] are given by k_{zx} = (0, 0, 0; L, 0, N) and k_y= (P, 0, R; 0, 1, 0). Since coordinates Q (the y-component of moment of an actuator force in the y-direction) is consistently zero, we can neglect the Q coordinate and just consider the (P, R, L, M, N) values to represent the actuator forces at each node of the profile. The representation for these are M_y = 1 for every point, and

$$P_y = -250\cos(\alpha), \quad R_y = 250\sin(\alpha),$$
$$L_{izx} = -x_i/\sqrt{x_i^2 + z_i^2}, \qquad N_{izx} = -z_i/\sqrt{x_i^2 + z_i^2} \tag{4.6}$$

The force balance at each node in the ZX-plane and the y direction leads to

$$\begin{aligned} F_{izx}d_{izx} &= [0 \quad 0 \quad F_{ix} \quad 0 \quad F_{iz}][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ &= F_{izx}[0 \quad 0 \quad L_{zx} \quad 0 \quad N_{zx}][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \end{aligned}$$

and

$$\begin{aligned} F_{iy}d_{iy} &= [T_{ix} \quad T_{iz} \quad 0 \quad F_{iy} \quad 0][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ &= F_{iy}[P_y \quad R_y \quad 0 \quad 1 \quad 0][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \end{aligned} \tag{4.7}$$

where the input virtual work terms on the left are dependent on values $d_{iy} = y_i$ and $d_{iz} = (x_i^2 + z_i^2)^{1/2}$. Representing the above equations Eq (4.6) for all the nodes in a matrix form gives us

$$[\mathbf{d}_i] = \begin{bmatrix} d_{1zx} \\ \vdots \\ d_{izx} \\ d_{iy} \\ \vdots \\ d_{ny} \end{bmatrix} \equiv [\mathbf{K'}][\mathbf{\$}] = \begin{bmatrix} 0 & 0 & L_{1zx} & 0 & N_{1zx} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & L_{izx} & 0 & N_{izx} \\ P_{iy} & R_{iy} & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ P_{ny} & R_{ny} & 0 & 1 & 0 \end{bmatrix} [\mathbf{\$}] \tag{4.8}$$

where [$\mathbf{d}_i$] is a column list of small normal distances at the respective actuators which are measured from the nominal profile to the points along both the directions of force applied by the actuators, matrix [**K'**] contains the pertinent line coordinates for the unit forces acting along those linear actuators and their moments about the origin, and [\$] = [$\delta\theta_x\ \delta\theta_z\ \delta x\ \delta y\ \delta z]^T$ is the SDT of the platform that carries the nominal profile shape and all other arcs concentric to it. (Note that the actuators all still exert force on the nominal profile (Fig 4.5)). The matrix expression (4.7) is a set of linear equations without a solution for [\$]. But an optimum LSF solution is given by the Moore-Penrose pseudoinverse.

$$[\$_{L-S}] = [k']^{\#}[d] = (([k']^T[k'])^{-1}[k']^T)[d]$$

to Eq. (4.7) gives the SDT [\$] = [$\delta\theta_x\ \delta\theta_z\ \delta x\ \delta y\ \delta z$] which minimizes the least squares sum of the deviations. From the coordinates in [\$] we can find the values of $\delta\theta$ and pitch h for the equivalent SDT and the line coordinates (L, M, N; P, Q, R) on which it lies. (See Chap. 2).

In section 5.1 the SDT coordinates are converted to a transformation matrix [5] that is used to obtain the coordinates of all the nodal points relative to the LSF profile. The result is used to find the boundaries of the least squares and true minimum zones that just captures all the points.

4.3 Least Squares Fitting of an Arc with Size Change.

Since the FEA model of forming the curved hat-section component constrained the median radius r = 315 mm (Fig. 4.1) to move straight down, the smaller radius flange was plastically deformed in tension and then sprang back to a smaller radius upon release from the press. In a like manner, the larger radius flange was plastically deformed in compression and then sprang back to a larger radius. Consequently, any curved component in its free state has a more open hat-section than the die where it was formed, and all of the sprung back nodal points of an arc-profile will be at a radius further from the median radius than its nominal radius. Since only those actuators in the zx-plane include the change in radius δr, values of deviations d_{izx} all contain a change in radius δr between that of the nominal profile and the least-squares profile. A distinct advantage of the statics-with-robotics method is that it allows radius change to be included selectively at actuators $\$_{izx}$ in the *zx*-plane, while leaving actuators $\$_y$ unaffected. Therefore, the first of Eqs. (4.6) should be modified to

$$\begin{aligned} d_{izx} &= [0 \quad 0 \quad L_{izx} \quad 0 \quad N_{izx}][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} - \delta r \\ &= [0 \quad 0 \quad L_{izx} \quad 0 \quad N_{izx} \quad -1][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z \quad \delta r]^{\mathrm{T}} \\ d_{iy} &= [T_{ix} \quad T_{iz} \quad 0 \quad F_{iy} \quad 0][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} - 0 \\ &= [T_{ix} \quad T_{iz} \quad 0 \quad F_{iy} \quad 0 \quad 0][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z \quad \delta r]^{\mathrm{T}} \end{aligned} \tag{4.9}$$

The radius change is inserted in Eq. (4.6) with a negative sign because all the d_{izx}-values are directed inward in Fig. 4.5, corresponding to a reduction in size. Now the SDT contains the added element δr and becomes [\$] = $[\delta\theta_x, \delta\theta_z, \delta_x, \delta y, \delta z, \delta r]^{\mathrm{T}}$, and, for each data point, the new row-pairs of [**K'**] are augmented to k_{zx} = (0, 0; L, 0, N, -1) and k_y = (P, R; 0, 1, 0, 0). As an example, for the inner arc-radius of 250 mm, the full array of linear equations takes the form,

$$[\mathbf{d}_{\mathrm{i}}] = \begin{bmatrix} d_{1zx} \\ \vdots \\ d_{izx} \\ d_{iy} \\ \vdots \\ d_{ny} \end{bmatrix} \equiv [\mathbf{K'}][\mathbf{\$}] = \begin{bmatrix} 0 & 0 & L_{1zx} & 0 & N_{1zx} & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & L_{izx} & 0 & N_{izx} & -1 \\ P_{iy} & R_{iy} & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ P_{ny} & R_{ny} & 0 & 1 & 0 & 0 \end{bmatrix} [\mathbf{\$}] \tag{4.10}$$

Lastly, the least-squares (pseudoinverse) solution to (Eqs 4.9) minimizes the sum.

$$\sum_{i=1}^{n} \left[\begin{array}{c} [d_{izx} - \{ + L_i\delta_x + N_i\delta_z - \delta r \}]^2 \\ +\left[d_{iy} - \{P_{iy}\delta\theta_x + R_{iy}\delta\theta_z + M_i\delta_y \}\right]^2 \end{array} \right]. \tag{4.11}$$

4.4 Least Squares Fitting of the J-shaped Profile in 3d Space.

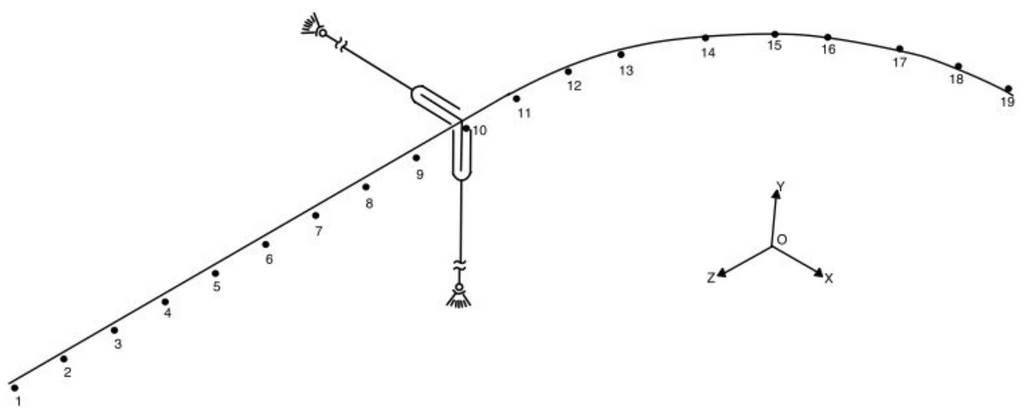


Fig 4.5. An Array of Selected Data Points Along One Edge Profile of the Assembled Structure with Component Linear Actuators at One Point.

The SDT for the J-shaped profile is represented with the Boolean union of the coordinates for the straight profile ($\Delta\theta_x \quad \Delta\theta_y \quad \Delta x \quad \Delta y$) and the coordinates for the arc profile ($\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z$) . Consequently, the SDT [\$] of this profile requires contains all six coordinates ($\Delta\theta_x$, $\Delta\theta_y$, $\Delta\theta_z$; Δx, Δy, Δz). Further, the six coordinates for each of the actuator forces $F_{izx}\$_{izx}$ and $F_y\$_y$ now must all take values. Therefore, in matrix [**K'**] we need to include a value for every coordinate (P, Q, R; L, M, N) at each actuator, some of which will be zero. For the straight profile, the entries are K'_x = (0, Q, 0; 1, 0, 0) and K'_y = (-P, 0, 0; 0, 1, 0), and, for the arc profile, the entries are K'_{zx} = (0, 0, 0; L, 0, N) and K'_y = (P, 0, R; 0, 1, 0). Both sets describe unit actuator forces depending on the geometry of the structure.

As a reminder, the force-balance equations at nodes of the straight part of one J-shaped profile then becomes.

$$\begin{aligned} F_{ix}d_{ix} &= [0 \quad T_{iy} \quad 0 \quad F_{ix} \quad 0 \quad 0][\Delta\theta_x \quad \Delta\theta_y \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ &= F_{ix}[0 \quad z_i \quad 0 \quad 1 \quad 0 \quad 0][\Delta\theta_x \quad \Delta\theta_y \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ F_{iy}d_{iy} &= [T_{ix} \quad 0 \quad 0 \quad 0 \quad F_{iy} \quad 0][\Delta\theta_x \quad \Delta\theta_y \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ &= F_{iy}[-z_i \quad 0 \quad 0 \quad 0 \quad 1 \quad 0][\Delta\theta_x \quad \Delta\theta_y \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \end{aligned} \tag{4.12}$$

and, for the curved portion,

$$\begin{aligned} F_{izx}d_{izx} &= [0 \quad 0 \quad F_{ix} \quad 0 \quad F_{iz}][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ &= F_{izx}[0 \quad 0 \quad L_{izx} \quad 0 \quad N_{izx}][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ F_{iy}d_{iy} &= [T_{ix} \quad T_{iz} \quad 0 \quad F_{iy} \quad 0][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \\ &= F_{iy}[-250\,\mathrm{C}\alpha_i \quad 250\,\mathrm{S}\alpha_i \quad 0 \quad 1 \quad 0][\Delta\theta_x \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]^{\mathrm{T}} \end{aligned} \tag{4.13}$$

where L_{izx} and N_{izx} are defined in Eqs. (4.5). Here d_{ix}, d_{iy}, dizx, and d_{iy} are an ordered list of deviations, measured from the nominal profile, of nodal points that form vector **[d**$_i$**]**. When the force amplitudes are cancelled out, the pairs of rows in matrix [**K'**] for each nodal point result. Substituting the equations, we get the matrix [**K'**] and, again applying the Gauss-Markov theorem, we get the $\$_{L\text{-}S}$ which minimizes the least squares sum.

$$[\$_{L-S}] = [k']^{\#}[d] = (([k']^T[k'])^{-1}[k']^T)[d]\ .$$

the SDT now consists of the six coordinates $[\Delta\theta_x \quad \Delta\theta_y \quad \Delta\theta_z \quad \Delta x \quad \Delta y \quad \Delta z]$ from which we can find the values of δθ and pitch h, and even further, we can compute the line coordinates (L, M, N; P, Q, R) of the small displacement torsor (SDT) (See Chap. 2).

In Section 5.1 methods are described (a) for transforming the given original points to a reference frame attached to the LSF profile and, from the points in this new frame,

(b) for finding the boundaries for the least squares and true minimum zones that just capture the nodal points along an arc-profile.

4.5 Least Squares Fitting of J-shaped Profile with Size Change.

Looking to the time when simulations exist for the frames of a matched pair assembly (Fig. 4.7), each of which includes the straight-and-arc subassembly in Fig. 4.1, values of deviations in the *ZX*-plane of Fig. 4.6 would all contain a change in feature size δr (radius δr along the arc) between that of the nominal profile and the least-squares profile. Deviations affected are d_{izx} along the arc in Fig. 4.5 and d_{ix} along the straight portion of the profile (Fig. 4.2). Focusing on the J-shaped subassembly in Fig. 4.7 and the parallel robot model for LS fitting, the platform carries the entire nominal profile and all profiles a constant distance from it and lying in the same plane. On one side of the nominal profile are J-shaped profiles with larger radius arcs, and on the other side lie profiles with smaller arc-radii.

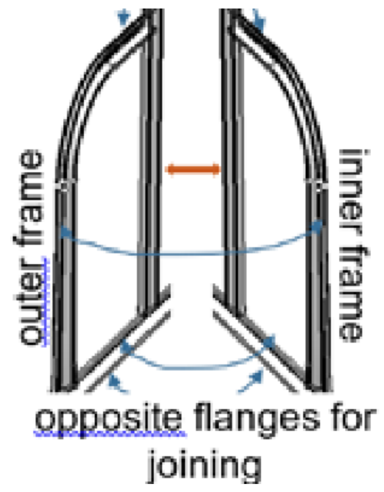


Fig. 4.6. Forming a Matched Pair Assembly by Joining Two Frame Subassemblies [2].

To include the change in size in the linear equations, the SDT in Sect. 4.4 is augmented to [$] = [$\Delta\theta_x$ $\Delta\theta_y$ $\Delta\theta_z$ Δx Δy Δz Δr], and the rows of [**K'**] must also be augmented with a seventh element. For all actuators lying in the *zx*-plane, the seventh element is -1, resulting in $K'_{izx} = (P_{izx}\ Q_{izx}\ R_{izx}\ L_{izx}\ M_{izx}\ N_{izx}\ -1)$, and for all the actuators parallel to the *y*-axis, the seventh element is zero, resulting in $K'_{iy} =(P_{iy}\ Q_{iy}\ R_{iy}\ L_{iy}\ M_{iy}\ N_{iy}\ 0)$. Now substituting these equations, we get the matrix to which Gauss Markov theorem is applied and the $\$_{L\text{-}S}$ could be obtained.

This SDT would now consist of 7 coordinates which account for the angular orientation, position, and the size change (from the nominal profile) of the least squares profile.

CHAPTER 5. Least Squares Zone and True Minimum Zone of a Profile

5.1 Finding the Coordinates of the Nodal Points in the Least Squares Fit Frame.

As mentioned above to find the coordinates of the nodal points in a reference frame attached to the least squares profile, we first need to find the homogeneous transformation matrix $[\mathrm{A}_{nL}]$ relating the frame of least squares profile to the frame of the original nominal profile. Given the SDT $\$_{\text{L-S}}$ in terms of its small rotation and amplitude ϕ and small axial translation $s = h\,\phi$, and given the normalized coordinates $(L\,,\ M,\ N;\ P_\ell,\ Q_\ell,\ R_\ell)$ of the line on which it lies, the required transformation is

$$[\mathrm{A}_{nL}] = \begin{bmatrix} & [\mathrm{R}_{nL}] & & x_{nL} \\ & & & y_{nL} \\ & & & z_{nL} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

in which in which $[\mathrm{R}_{nL}]$ is the 3x3 rotation matrix

$$[\mathrm{R}_{nL}] = \begin{bmatrix} L^2V_\phi + \mathrm{C}_\phi & MLV_\phi - NS_\phi & NLV_\phi + MS_\phi \\ LMV_\phi + NS_\phi & ML^2V_\phi + \mathrm{C}_\phi & NMV_\phi - LS_\phi \\ LNV_\phi - MS_\phi & MNV_\phi + LS_\phi & N^2V_\phi + \mathrm{C}_\phi \end{bmatrix} \tag{5.2}$$

and

$$\left.\begin{aligned} x_{nL} &= V_\phi(MR_\ell - NQ_\ell) + P_\ell \mathrm{S}_\phi + Ls \\ y_{nL} &= V_\phi(NP_\ell - LR_\ell) + Q_\ell \mathrm{S}_\phi + Ms \\ z_{nL} &= V_\phi(LQ_\ell - MP_\ell) + R_\ell \mathrm{S}_\phi + Ns \end{aligned}\right\} \tag{5.3}$$

in which $V_\phi =$ versine $\phi = 1 - \cos\phi$, $S_\phi = \sin\phi$, and $C_\phi = \cos\phi$. (5.4)

See equations (4.62) and (4.63) in §4.6.3 of [5].) Matrix $[\mathrm{A}_{nL}]$ transforms any point (x_L, y_L, z_L) in the LS frame to its representation (x_n, y_n, z_n) in the nominal frame. So, the original points can be represented relative to the new LSF profile by transforming them all with the *inverse* matrix $[\mathrm{A}_{Ln}]$.

5.2 Least Squares Zone Magnitude

Two single-parameter measures for spring-back variation along the LSF line-profile for a circular-arc, or for the J-shape in Fig. 4.6, are the least-squares (LS) minimum zone and the true minimum zone. The magnitude of the LS minimum zone is the largest deviation (radius) of any nodal point *i* from the LSF profile, and, is obtained as the largest of values $(x_{iL}^2 + y_{iL}^2 + z_{iL}^2 - (r_n + \Delta \mathrm{r})^2)^{1/2}$ from any arc portion, and $(x_{iL}^2 + y_{iL}^2 - (r_n + \Delta \mathrm{r})^2)^{1/2}$ from any straight portion of a J-profile, for all points i. If every nodal point, together with its distance to closest point on the LS profile, were projected along the LS profile to one plane with all the closest points on the profile coinciding, the largest distance becomes the radius of the LS minimum zone.

The second parameter is the true minimum zone that is an unconstrained cylindrical metrological zone, the magnitude for which may be found using a traditional metrological algorithm. However, for both the arc-profile and the J-shaped profile, a preliminary transformation must take place to unwrap the LS profile and associated nodal points to a straight line.

5.3 Unwrapping of the Points along the Arc Profile

The unwrapping transformation is obtained by unwrapping the LS profile, with all the nodal points attached, until the profile is straight (Fig. 5.1). This amounts to projecting the arc portion, together with associated nodal points, to the single straight line that (a) is obtained from the SDT, (b) is parallel to the Z_L-axis (Fig. 5.1) and (c) is tangent to the LSF arc of radius $r_n + \Delta r$. The projection is accomplished by representing the nodal points in the reference frame $O'x'y'z'$ that is (a) parallel to frame Oxyz in Fig. 5.1 and (b) with its origin O' at the tangent point. The transformation equations are

$$X_L' = (r_n + \Delta r) - ({x_{iL}}^2 + {z_{iL}}^2)^{1/2}$$

$$Y_L' = y_{iL}$$

$$Z_L' = (r_n + \Delta r)\left({}^{\pi}/_{2} - \alpha\right) \tag{5.1}$$

$$\text{where, } \alpha = \text{atan2}(x_{iL}/z_{iL}). \tag{5.2}$$

These equations were found by first getting the angle as given in Eq. (5.2), then multiplying this angle with the radius of the Least Squares fit profile to get the Z_L' coordinate, i.e., the distance of the points along the arc. The Y_L' values remain the same as they are unaffected by the unwrapping and projecting of points, The X_L' values are taken by finding the distance of the given points from the origin along the *zx*-plane, and these points are used to find the least squares zone size.

The next section shows how a minimum circumscribing cylinder may be found as a parameter for points along an edge of a straight component, an arc component, and the J-shaped assembly.

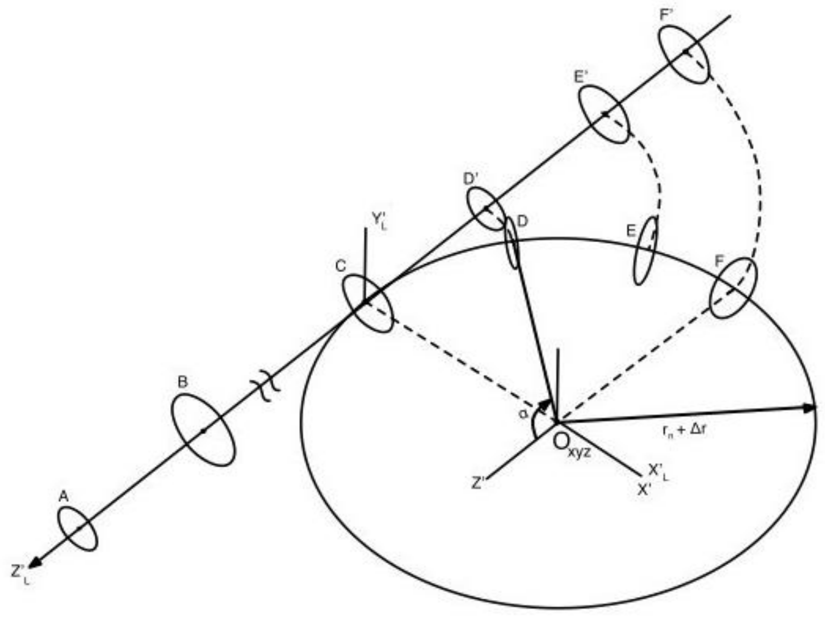


Fig 5.1. Unwrapping of the LSF Arc Profile onto a Straight-line Profile

5.4 True Minimum Zone

Known metrological methods [6] may be applied to obtain the unconstrained minimum circumscribing cylinder whose radius is the true minimum zone. The true minimum zone is always smaller than the least squares zone magnitude. To find the true minimum zone for sprung back points along the edge of a component (e.g., the 500 mm straight edge in Fig. 4.1), or any unwrapped arc-profile, we can use an algorithm listed in the Section 5.2.4 of [6] and first developed by [21]. The algorithm first finds the least squares axis to the points (Z_L' -axis for the arc or the J-shaped profile). Then, at each end of the least squares line, one creates hexagons (Fig. 5.3(b)) that are circumscribed by circles of radius 10% of the least squares zone radius. Next, 36 tentative axes are formed by joining every vertex on one hexagon to every vertex on the other hexagon (Fig. 5.3(c)). A new coordinate frame is established with its z-axis on each of the 36 axes, the coordinates for all the points are transformed to each new frame, and 36 new zone radii are produced (largest radius from all the points to each axis). Of the 36 axis-and-zone-radius pairs, retain only the one with the smallest zone radius. The process is repeated with the size of hexagons being reduced by half until the total size of the hexagon is less than the 0.01% of the least squares zone magnitude. Thus, we can find the true minimum zone radius and axis. We can directly incorporate it for the 11-point straight line mentioned above but, while doing it for the curved portion, we need first to unwrap the arc to straight line, then apply the algorithm.

For this thesis, a few modifications were made to the algorithm for it to be written as a program. The first is we have shifted the coordinate frame origin to one of the ends of the Least Squares Fit (LSF) line to the origin so it would be easier to find the transformation matrix during the rotation of axis to make it parallel to the z-axis and to transform the point with it, which can be translated back to the original position once we end up with the resultant true minimum zone radius and axis. Also, a change was made to the rate at which the size of the hexagons decreases at the rate of 10:11 instead of 1:2.

Specifics of the algorithm are mentioned in the appendix portion [Appendix. F].

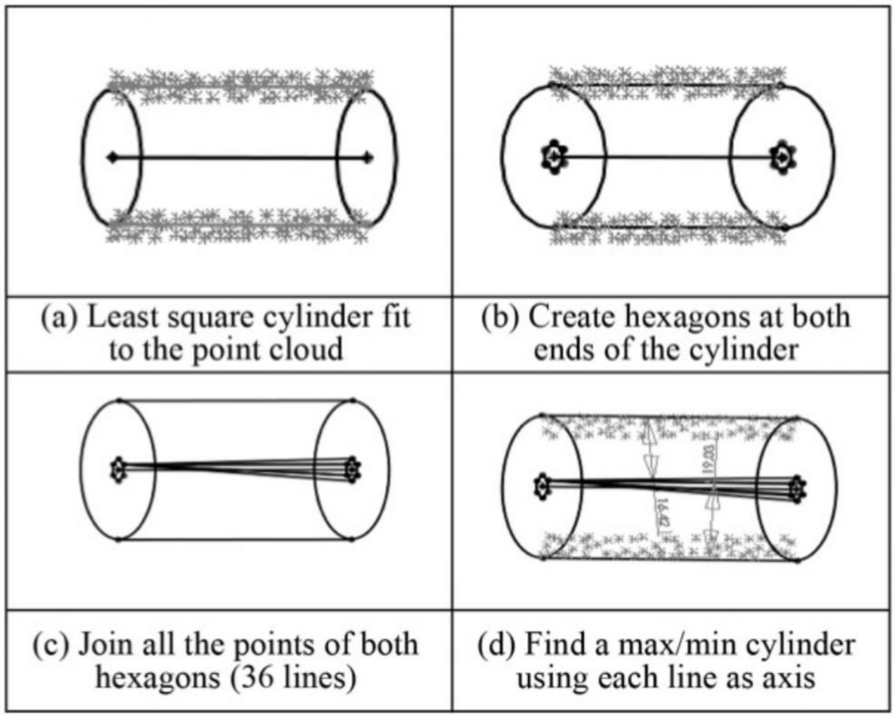

(a) Least square cylinder fit to the point cloud
(b) Create hexagons at both ends of the cylinder
(c) Join all the points of both hexagons (36 lines)
(d) Find a max/min cylinder using each line as axis
16.42
19.03

Fig 5.2. Minimum Cylindrical Fit for Points in Space. [6]

CHAPTER 6. Numi-sheet Parameters to Validate the FEA Stamping Model Created at The Ohio State University

As a part of NSF Goali project in collaboration with Ohio State university a lot of Stamping FEA simulations were done for hat-shaped components having different properties such as Shapes, Channel Width, Material, Thickness, Draw depth and Blank holding forces. To validate the OSU FEA model for forming components, software was created as a part of this thesis to compute the geometric parameters from the NUMISHEET 1993 benchmark [4] and consistent with more recent existing experimental and simulated results [20].

To gain confidence in the forming-stage simulations, the process was first validated against an existing set of experimental and simulated results [2] based on the NUMISHEET 1993 benchmark [4].

6.1 Extracting Numi-Sheet Parameters (Inverted Hat Section) for FEA and Experimental Parts

The computations to obtain the Numi-sheet benchmark parameters for stamped sheet-metal components were undertaken using, as closely as practicable, procedures outlined in the *Numi-sheet 93 research paper* [4]. The Numi-sheet parameters are the angles θ_1, θ_2, and radius ρ, and these, along with the original Numi procedures, are shown in Fig. 6.1. Two modifications were made to the original procedures. First, since three points on a straight line do not define an arc, we modified somewhat the identification of point C to be consistent with the desired parameter ρ. Second, the FEA points in the region of point D did not always exhibit an inflection, so D was identified as the highest point (maximum *y*-value) in the list of points. With these modifications, our procedures are as follows.

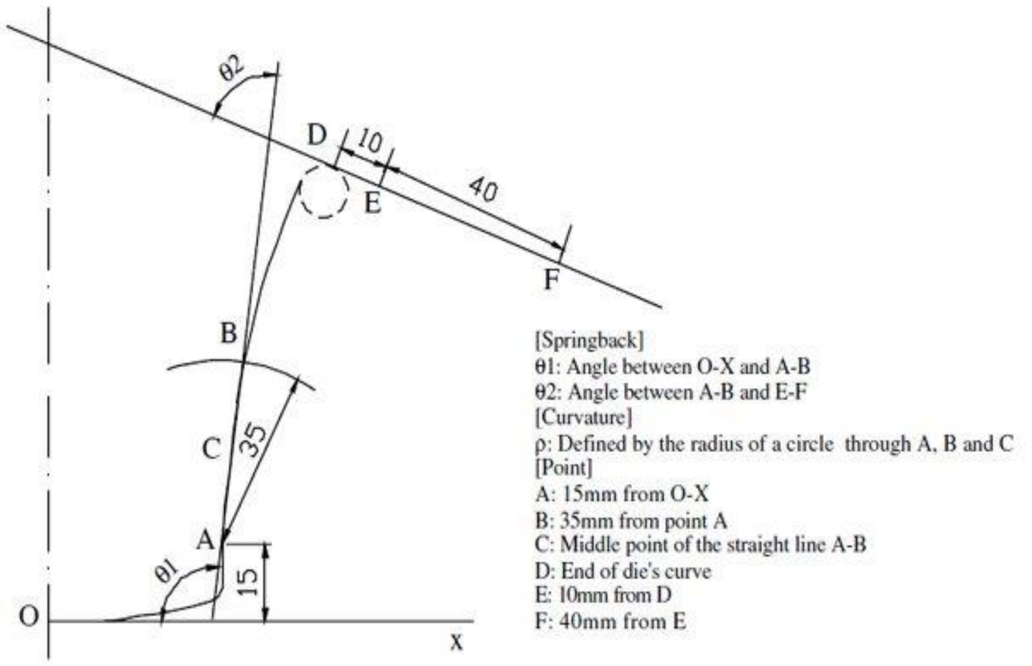

θ2
D
10
E
40
F
B
C
35
A
θ1
15
O
X
[Springback]
θ1: Angle between O-X and A-B
θ2: Angle between A-B and E-F
[Curvature]
ρ: Defined by the radius of a circle through A, B and C
[Point]
A: 15mm from O-X
B: 35mm from point A
C: Middle point of the straight line A-B
D: End of die's curve
E: 10mm from D
F: 40mm from E

Fig 6.1. Parameters to be Extracted from the Stamped Components

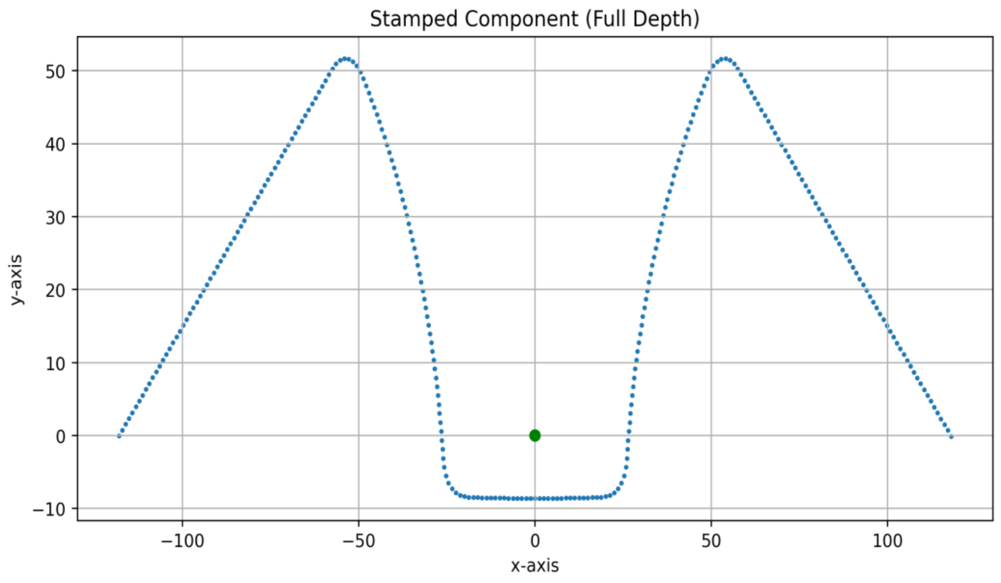


Fig 6.2. The Origin of the Stamped Component Obtained from FEA Simulation.

Figures 6.2 and 6.3 show a typical point-set in its original coordinates and a set of computational coordinates that were used in computing the Numi-parameters. The computational coordinates were obtained by shifting the origin so the point with least *y*-coordinate in Fig. 6.2 becomes the origin.

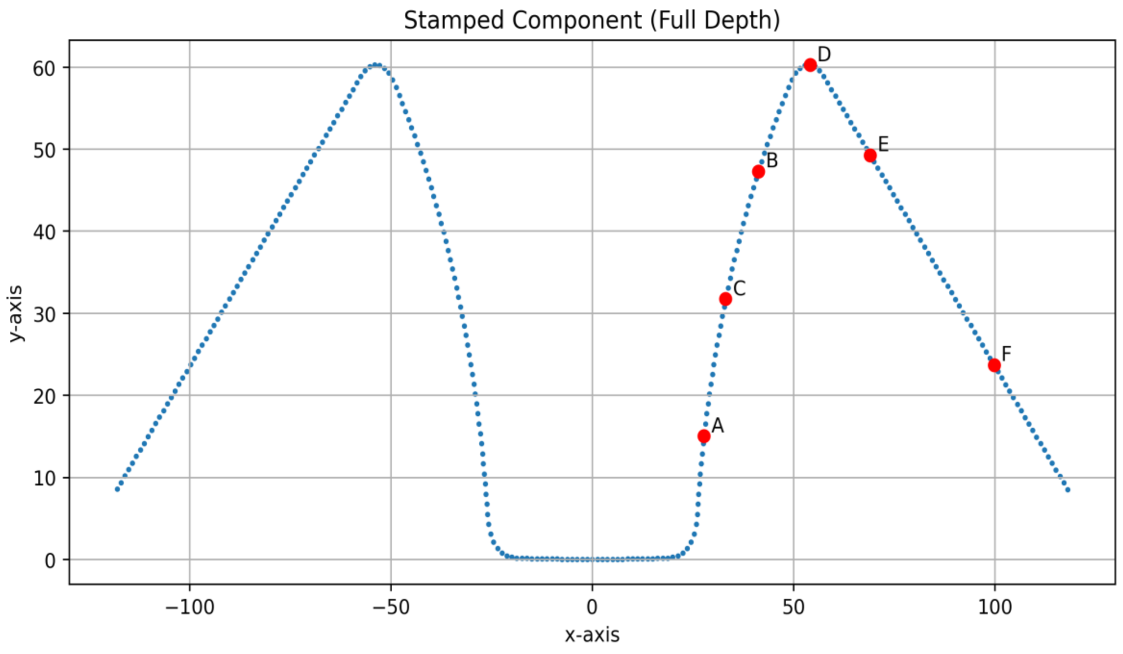


Fig 6.3. The Points Found Using Numi-sheet 93 [4] to Extract the Parameters.

Now the first point *A* is chosen at a distance of *y* = 15 mm from the base (Fig. 6.3), a location typically between two data points A_1 and A_2 that are the closest points above and below the 15mm height. The coordinates of *A* are then obtained by interpolating linearly between these. The second point *B* is chosen where an arc of 35mm radius centered at point *A* intersects the data between points B_1 and B_2. Once again, linear interpolation leads to the desired location for the point *B*. Next, find the perpendicular bisector to line *AB* and, where it passes through the sprung back points, identify the two points closest to it. These are C_1 and C_2. Use linear interpolation again to obtain the point *C* as shown in the figure 6.4.

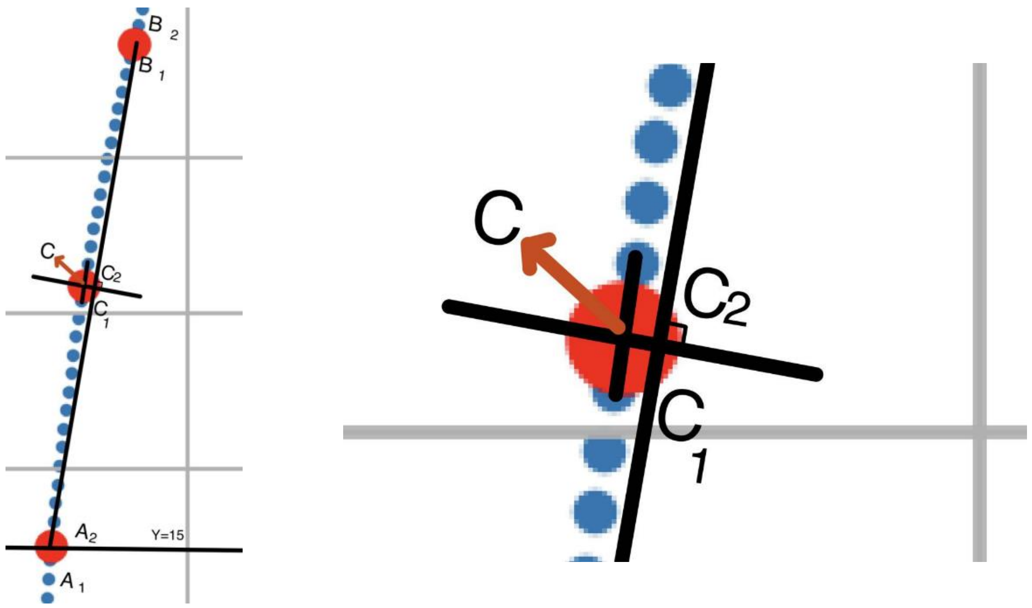


Fig 6.4. Finding Point C Using Numi-sheet [4]

The point *D* is, the point where the sheet leaves the die since we can’t exactly find the point from the profile given the idea is to find the inflection for which choose a point about 10mm along the profile from the peak point in the given data set and choose another point which is about 50mm along the profile and form a line with these two points and we can find the distance of each point from the line we have generated now if we observe the change in distances as shown if the Figure Fig 6.5 below the points tend to go down and starts to move up at a particular point and the point until where the distance decreases without any increases is taken to be the point where the stamping leaves the die. It works best in the case of dense data structure where this phenomenon can be clearly observed. But this is not the case if the simulation data doesn’t contain the die end in the mesh. So, to prevent all this mess and to standardize we assumed point ‘D’ to be at a maximum position and chose point ‘E’ to be at 15mm from point ‘D’.

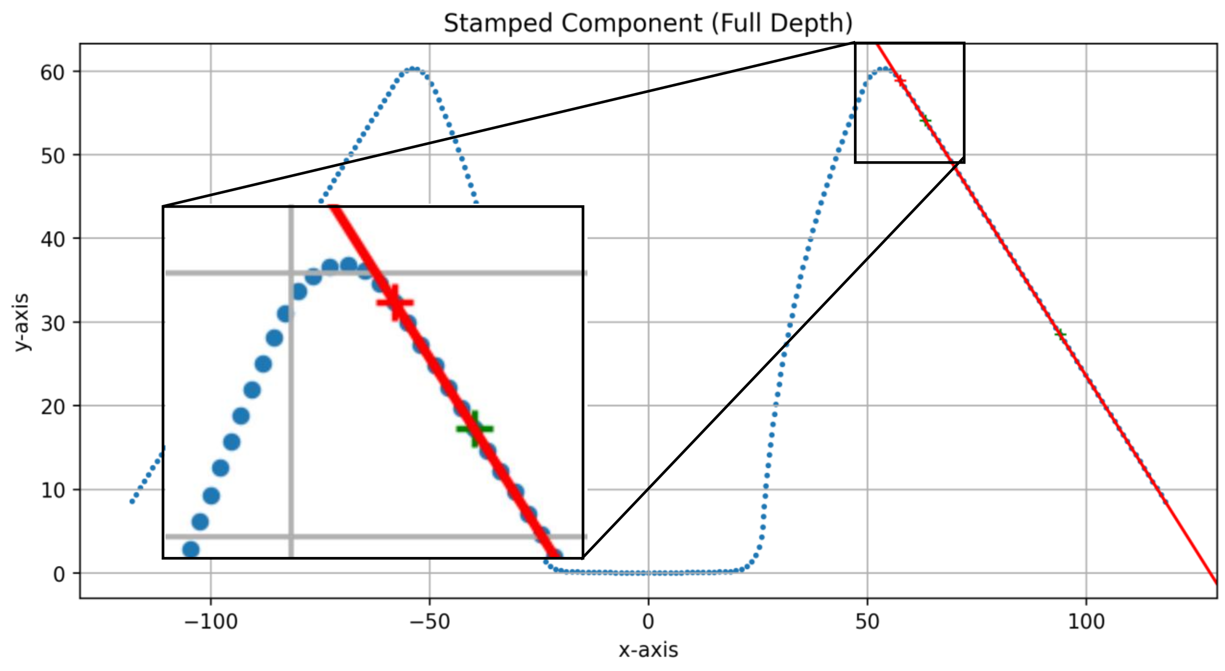


Fig 6.5. Point Where Structure Leaves the Die.

Now for point F find the point at 40mm from E by interpolation.

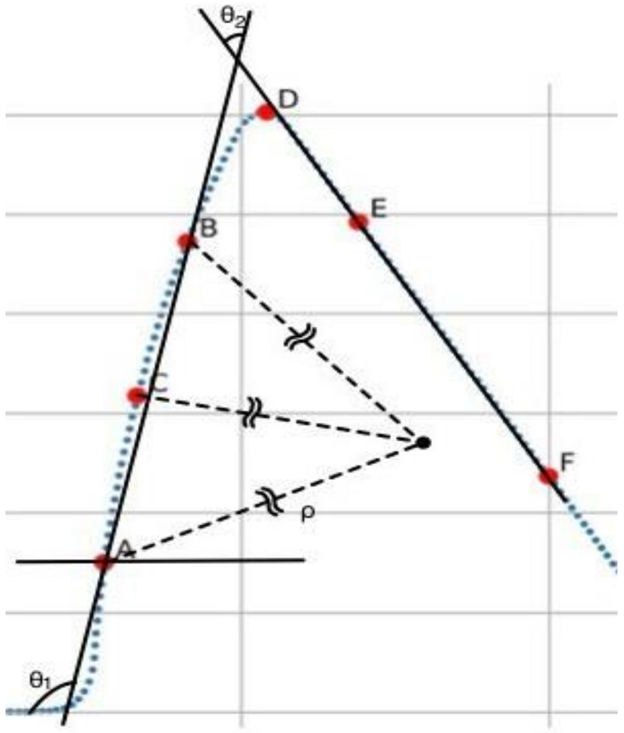


Fig 6.6. Parameters Extracted from the Stamped Profile.

Use the points obtained to find the curvature ρ and the slopes of both the line θ_1 and θ_2. The ρ can be found by applying the algorithm from He, *et al*. [18] *Appendix*. If i-1, i and i+1 are the points then the radius of curvature is given by,

$$A = (x_{i+1}, y_{i+1}), B = (x_{i-1}, y_{i-1}), C = (x_i, y_i)$$

$$\rho = \sqrt{(x_\mathrm{i} - x_\mathrm{c})^2 + (y_\mathrm{i} - y_\mathrm{c})} \qquad (6.1)$$

$$\text{where } x_c = \{1/d\}\{R_A(y_{i+1} - y_i) - R_B(y_i - y_{i-1})\} ,$$

$$y_c = \{-1/d\}\{R_A(x_{i+1} - x_i) - R_B(x_i - x_{i-1})\}$$

are the coordinates of the center of curvature and the values of R_a, R_b and d are,

$$R_A = (1/2)(x_i^2 - x_{i-1}^2 + y_i^2 - y_{i-1}^2),$$

$$R_B = (1/2)(x_{i+1}^2 - x_i^2 + y_{i+1}^2 - y_i^2) \text{ and}$$

$$d = (x_i - x_{i-1})(y_{i+1} - y_i) - (y_i - y_{i-1})(x_{i+1} - x_i)$$

This algorithm is robust even for ill conditioned points that all lie nearly on a line.

When dealing with the half depth stampings we changed the distance to A from ground as 10mm and the point B to be at 15mm from point A. and changed these parameters for different depths accordingly and extracted the values of ρ, θ_1 and θ_2. We developed a few programs to make this finding of the parameters automatically so we can test all the stamped parts instead of choosing few components and testing them.

When practical experiments of the half-depth stampings were done to compare with the simulations, then we noticed inconsistencies for the center of curvature: for some specimens, the center of curvature was to the left of the array of points between points A and B, yet for other specimens it was to the right of the array. By introducing a little perturbation along the x-axis at point B in the computations, it was found that the center of curvature would flip from side to side. So, for points scanned along the half-depth experimental specimens, the radius parameter was not useful (see 6.3).

The program is written to extract parameters from all the stamping simulations by taking their naming format to determine the depth and choose the proper distances for depths and save the resulting parameters into an excel sheet. Those values were then used to analyze the fittings of the stamped components.

6.2 3d Line Fitting for the Edge Profiles of Stamped Components

This is to fit the edges of the stamped profiles to a line using Least Squares. This is done in the same fashion mentioned in chapter 4.

6.3 Effect of Perturbation

While extracting the parameters from the stamped components using the Numi-Sheet methods there were a lot of inconsistencies for the value of radius ρ even for same stampings at different time points when done in practical test. So as hypothesis have been made to test the sensitivity of the ρ which is to implement a small perturbation of 0.1mm along the x-axis in either direction and calculated the value of ρ. These perturbations seem to affect the parameters of the stamped components by a considerable amount. The x-coordinate value of point B is varied by +0.1mm and -0.1mm and found the changes in the radius of curvatures for 55mm (full depth stamping) and found an average change of about 4% along both directions and for 35mm (half depth stamping) the average change is around 17% along the positive direction and about 26% along the negative direction.

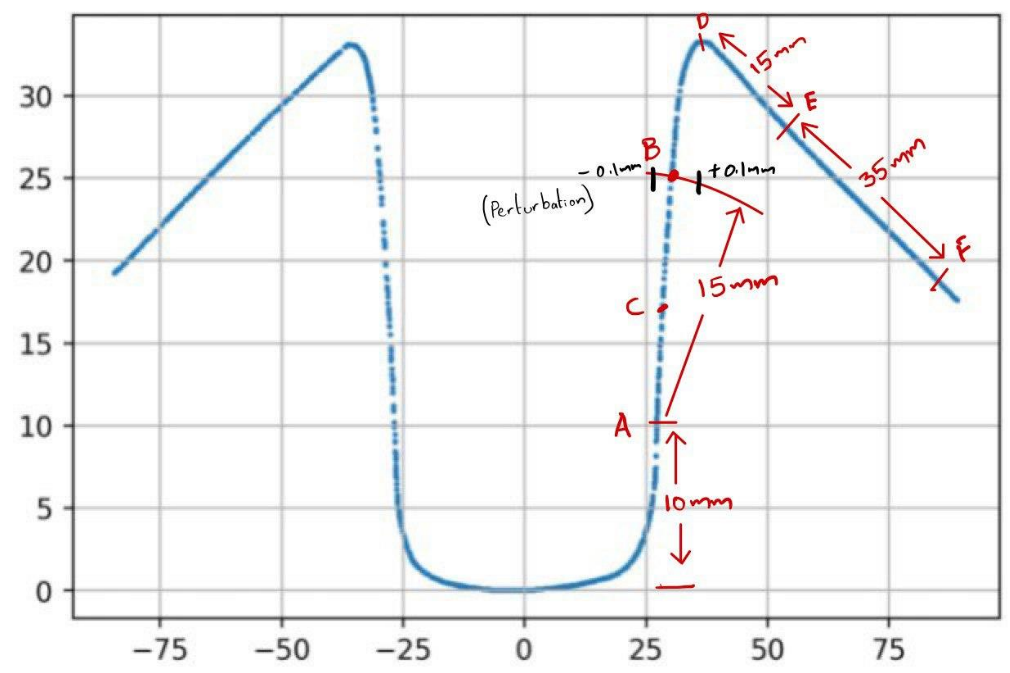


Fig 6.7. Showing the Perturbation Points at Point B for Half Depth Stamping

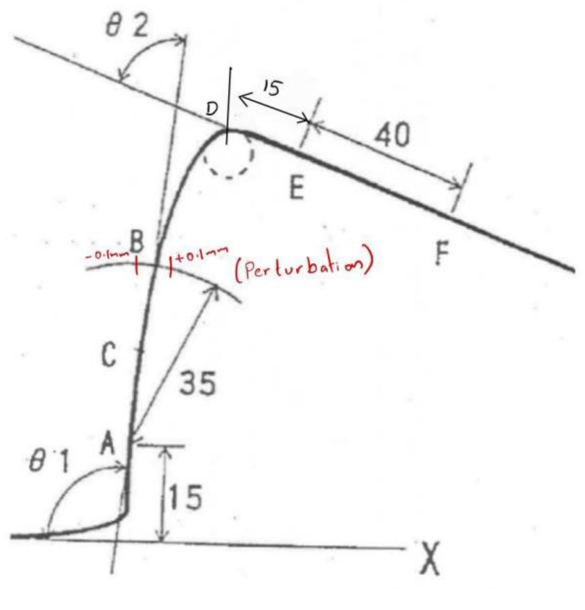


Fig 6.8. Showing the Perturbation Points at Point B for Full Depth Stamping

CHAPTER 7. Results and Conclusion

This chapter contains cross-sectional views of data points, axes, and zone boundaries for the least-squares fit zones and True Minimum zones for different profiles: the straight (Fig. 4.2), the arc-segment (Fig. 4.4), and the joined J-shaped profiles in Fig.4.5, all for both the shorter and longer edges of the assembly shown in Fig. 4.1. The coordinates for sample points of inner profile are in Table 7.1. The first 11 points represent the straight-line profile, the last 9 points are used for the arc profile, and all 19 are used to evaluate the J-shaped profile. Although points #11 was used to evaluate both the straight and arc-profiles, it was used only once for the J-shaped profile. The coordinates for sample points of outer profile are in Table 7.2. The first 11 points represent the straight-line profile, the last 12 points are used for the arc profile, and all 23 are used to evaluate the J-shaped profile. Although points #11 was used to evaluate both the straight and arc-profiles, it was used only once for the J-shaped profile.

The results of fitting for the inner J-section profile are given in the table 7.2 and the same operations are performed for the outer profiles and the results of those are given in the table 7.3.

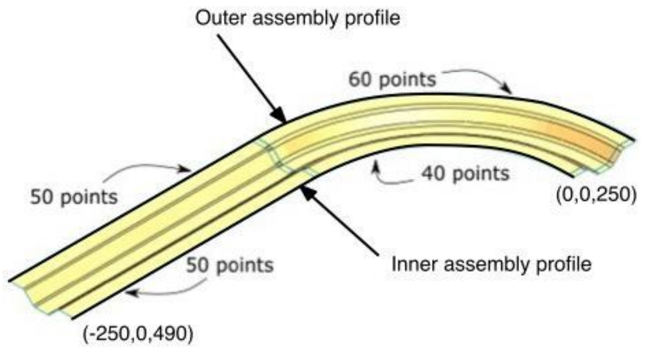


Fig 7.1 Figure Showing the Inner Assembly Profile and Outer Assembly Profile [1]

Table 7.1 Sample Points for Inner Assembly Profile

X coordinates	Y coordinates	Z coordinates
-241.899	0.743	490.025
-241.842	0.535	451.025
-241.789	0.303	401.025
-241.745	0.099	351.026
-241.709	-0.067	301.028
-241.669	-0.196	251.03
-241.633	-0.305	201.023
-241.599	-0.394	151.027
-241.564	-0.444	101.031
-241.53	-0.204	51.037
-241.501	0.982	1.042
-239.119	1.754	-45.512
-226.11	1.52	-95.784
-207.026	1.314	-135.018
-175.072	1.224	-176.908
-141.82	1.304	-205.99
-95.55	1.605	-232.311
-53.224	2.048	-246.172
-0.024	2.748	-252.122

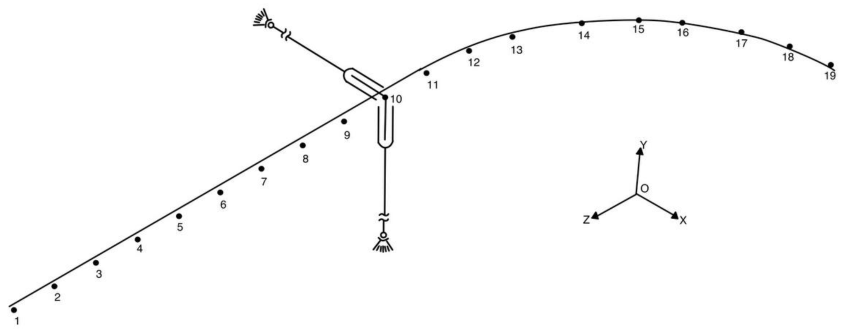


Fig 7.2 Figure Showing the Sampling Points along the Inner Assembly Profile.

Table 7.2 Sample Point for Outer Assembly Profile

X coordinates	Y coordinates	Z coordinates
-388.068	0.886	490.027
-388.122	0.658	451.028
-388.183	0.398	401.028
-388.235	0.161	351.029
-388.289	-0.037	301.031
-388.327	-0.2	251.033
-388.371	-0.343	201.036
-388.413	-0.458	151.03
-388.456	-0.512	101.034
-388.5	-0.279	51.046
-388.549	1.258	1.045
-385.449	3.099	-50.236
-376.946	3.298	-101.361
-361.442	3.038	-151.013
-339.32	2.718	-198.293
-310.977	2.339	-242.296
-276.835	1.934	-282.171
-237.514	1.546	-317.146
-193.715	1.189	-346.51
-146.271	0.876	-369.694
-96.042	0.626	-386.218
-43.959	0.446	-395.582
-1.683	0.385	-398.098

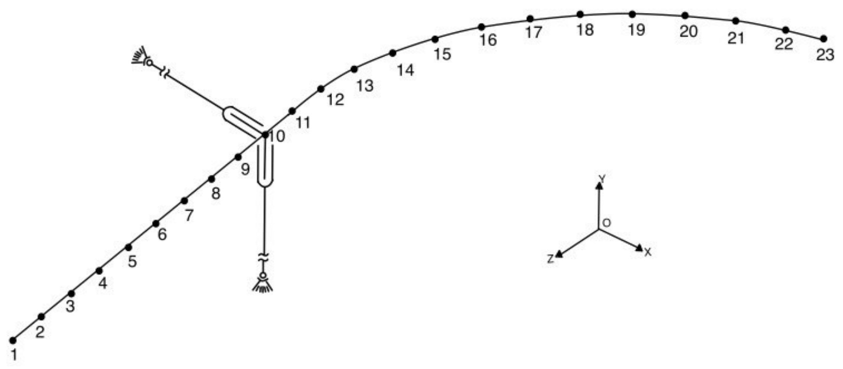


Fig 7.3 Figure Showing the Sampling Points along the Outer Assembly Profile.

Table 7.3 Results of Least Squares Fit of Sampling Points for Different Profiles along the Inner Assembly Profile.

$_{L-S}$	Straight	Arc	Arc	Assembly	Assembly
		$\Delta r = 0$	$\Delta r \neq 0$	$\Delta r = 0$	$\Delta r \neq 0$
$\Delta\theta_x, rad$	-0.054	-0.238	-0.238	0.125	0.125
$\Delta\theta_y, rad$	-0.045	0	0	0.12	-0.042
$\Delta\theta_z, rad$	——	0.478	0.478	-0.119	-0.119
Δx, mm	8.516	6.825	0.443	7.514	0.353
Δy, mm	-0.14	3.549	3.549	0.797	0.797
Δz, mm	0	-3.812	-10.132	-4.199	-10.271
Δr, mm	——	0	7.993	0	8.146
ϕ, deg	0.07	0.534	0.534	0.21	0.177
h, mm/deg	-91.809	-12.1	-17.367	34.78	39.112
s, mm	-6.454	-6.456	-9.266	7.298	6.931
LS zone radius, mm	1.121	1.715	0.609	2.413	1.51
Min zone rad, mm	0.689	1.35	0.444	1.621	1.214

Table 7.4 Line Coordinates of the Least Squares Fit of Sampling Points along Inner Profile.

Line Coordinates	Straight	Arc, $\Delta r = 0$	Arc, $\Delta r \neq 0$	Assy, $\Delta r = 0$	Assy, $\Delta r \neq 0$
L, radians	-0.768	-0.446	-0.446	0.595	0.704
M, radians	-0.64	0	0	0.572	-0.238
N, radians	0	0.895	0.895	-0.565	-0.669
P_1, mm	2899.958	423.715	-396.226	866.659	-1464.046
Q_1, mm	-3481.04	381.122	381.122	-921.858	790.287
R_1, mm	0	211.137	-197.439	-20.753	-1821.911
Z-intercept, mm	4531.2	——	——	——	——

Table 7.5 Results of Least Squares Fit of Sampling Points for Different Profiles along the outer Assembly Profile.

$_{L-S}$	Straight	Arc	Arc	Assembly	Assembly
		$\Delta r = 0$	$\Delta r \neq 0$	$\Delta r = 0$	$\Delta r \neq 0$
$\Delta\theta_x, rad$	-0.064	0.165	0.165	-0.028	-0.028
$\Delta\theta_y, rad$	0.054	0	0	-0.206	0.003
$\Delta\theta_z, rad$	——	-0.546	-0.546	-0.471	-0.471
Δx, mm	-8.556	-6.214	0.171	-6.977	0.644
Δy, mm	-0.139	-1.178	-1.178	0.052	0.052
Δz, mm	0	-16.565	-10.134	-16.13	-9.425
Δr, mm	——	0	-8.099	0	-8.887
ϕ, deg	0.084	0.57	0.57	0.515	0.472
h, mm/deg	76.636	24.652	17.102	29.361	19.853
s, mm	6.419	14.053	9.749	15.116	9.371
LS zone radius, mm	1.396	2.995	1.472	3.928	3.817
Min zone rad, mm	0.847	1.50	0.981	1.743	1.643

Table 7.6 Line Coordinates of the Least Square Fit of Sampling Points along Outer Profile.

Line Coordinates	Straight	Arc, $\Delta r = 0$	Arc, $\Delta r \neq 0$	Assy, $\Delta r = 0$	Assy, $\Delta r \neq 0$
L, radians	-0.761	0.29	0.29	-0.054	-0.059
M, radians	0.649	0	0	-0.399	0.006
N, radians	0	-0.957	-0.957	-0.915	-0.998
P_1, mm	-2512.56	-1033.81	-266.719	-685.93	144.862
Q_1, mm	-2944.88	-118.359	-118.359	677.717	-0.527
R_1, mm	0	-313	-80.753	-255.492	-8.517
Z-intercept, mm	3871.4	——	——	——	——

Table 7.7 Results of Least Squares Fit of Total Points for Different Profiles along the Inner Assembly Profile.

$_{L-S}$	Straight	Arc	Arc	Assembly	Assembly
		$\Delta r = 0$	$\Delta r \neq 0$	$\Delta r = 0$	$\Delta r \neq 0$
$\Delta\theta_x, rad$	-0.056	-0.463	-0.476	0.121	0.121
$\Delta\theta_y, rad$	-0.044	0	0	0.124	-0.04
$\Delta\theta_z, rad$	——	0.609	0.618	-0.159	-0.159
Δx, mm	8.514	6.517	0.484	7.543	0.312
Δy, mm	-0.203	4.588	4.649	0.675	0.675
Δz, mm	0	-3.772	-10.046	-4.422	-10.29
Δr, mm	——	0	7.905	0	8.182
ϕ, deg	0.072	0.766	0.78	0.235	0.204
h, mm/deg	-91.976	-9.074	-10.579	30.691	39.6
s, mm	-6.578	-6.948	-8.253	7.224	8.077
LS zone radius, mm	1.547	2.191	1.12	2.673	1.578
Min zone rad, mm	0.825	1.35	0.505	1.65	1.267

Table 7.8 Line Coordinates of the Least Square Fit of Total Points along Inner Profile.

Line Coordinates	Straight	Arc, $\Delta r = 0$	Arc, $\Delta r \neq 0$	Assy, $\Delta r = 0$	Assy, $\Delta r \neq 0$
L, radians	-0.787	-0.605	-0.61	0.514	0.593
M, radians	-0.616	0	0	0.528	-0.198
N, radians	0	0.796	0.792	-0.676	-0.78
P_1, mm	2671.309	172.939	-334.461	931.736	-1259.054
Q_1, mm	-3411.71	343.319	341.464	-764.156	639.913
R_1, mm	0	131.53	-257.747	112.024	-1120.866
Z-intercept, mm	2860.3	——	——	——	——

Table 7.9 Results of Least Squares Fit of Total Points for Different Profiles along the Outer Assembly Profile.

$\$_{L-S}$	Straight	Arc	Arc	Assembly	Assembly
		$\Delta r = 0$	$\Delta r \neq 0$	$\Delta r = 0$	$\Delta r \neq 0$
$\Delta\theta_x, rad$	-0.069	0.094	0.094	-0.021	-0.021
$\Delta\theta_y, rad$	0.054	0	0	-0.209	-0.021
$\Delta\theta_z, rad$	——	-0.513	-0.513	-0.457	-0.457
Δx, mm	-8.557	-6.008	-0.491	-7.042	0.582
Δy, mm	-0.224	-0.721	-0.721	-0.004	-0.004
Δz, mm	0	-16.596	-11.119	-15.897	-9.636
Δr, mm	——	0	-7.007	0	-8.706
ϕ, deg	0.087	0.521	0.521	0.502	0.457
h, mm/deg	75.692	29.255	20.818	29.326	20.963
s, mm	6.6	15.247	10.85	14.734	9.59
LS zone radius, mm	2.034	3.286	1.935	3.752	3.655
Min zone rad, mm	1.059	1.501	1.029	1.77	1.686

Table 7.10 Line Coordinates of the Least Square Fit of Total Points along Outer Profile.

Line Coordinates	Straight	Arc, $\Delta r = 0$	Arc, $\Delta r \neq 0$	Assy, $\Delta r = 0$	Assy, $\Delta r \neq 0$
L, radians	-0.787	0.18	0.18	-0.041	-0.045
M, radians	0.616	0	0	-0.416	-0.046
N, radians	0	-0.984	-0.984	-0.909	-0.998
P_1, mm	-2207.68	-961.558	-268.157	-734.369	126.734
Q_1, mm	-2820.54	-79.255	-79.255	697.796	54.878
R_1, mm	0	-175.558	-48.959	-286.143	-8.235
Z-intercept, mm	2311.5	——	——	——	——

Table 7.11 Least Square Zone Size of Different Profiles.

Least squares zone (Radius, mm)	Inner Sampling	Outer Sampling	Inner Total	Outer Total
Straight Profile	1.121	1.396	1.547	2.034
Arc Profile	1.715	2.995	2.191	3.286
Arc with size change	0.609	1.472	1.12	1.935
Assembly Profile	2.413	3.928	2.673	3.752

Assembly with size change	1.51	3.817	1.578	3.655

Table 7.12 True Minimum Zone Size of Different Profiles.

True minimum zone (Radius, mm)	Inner Sampling	Outer Sampling	Inner Total	Outer Total
Straight Profile	0.689	0.847	0.825	1.059
Arc Profile	1.350	1.500	1.351	1.501
Arc with size change	0.444	0.981	0.505	1.029
Assembly Profile	1.621	1.743	1.65	1.77
Assembly with size change	1.214	1.643	1.267	1.686

11 Point Straight Line Profile

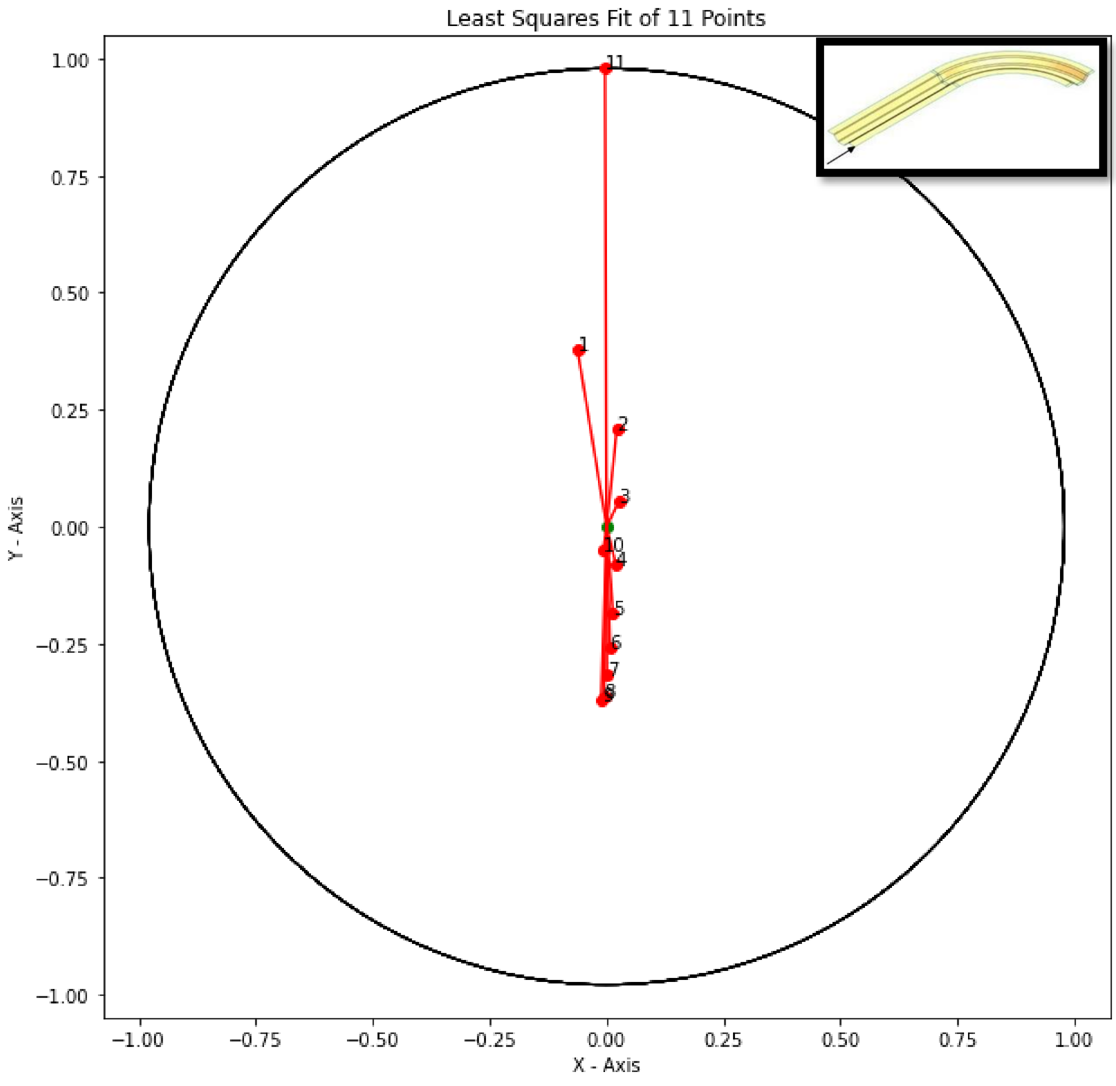


Fig 7.4 Least Squares Zone Boundary for 11 Points on the Straight Portion of the Inner Profile.

The screw coordinates of the SDT for the Least Squares Fit are L = -0.8676816753875916, M = -0.4971202170467245, N = 0, P = 1036.699263691128, Q = -1809.4716793786192, and R = 0. The zone magnitude (radius in mm) is 0.9781767537409084

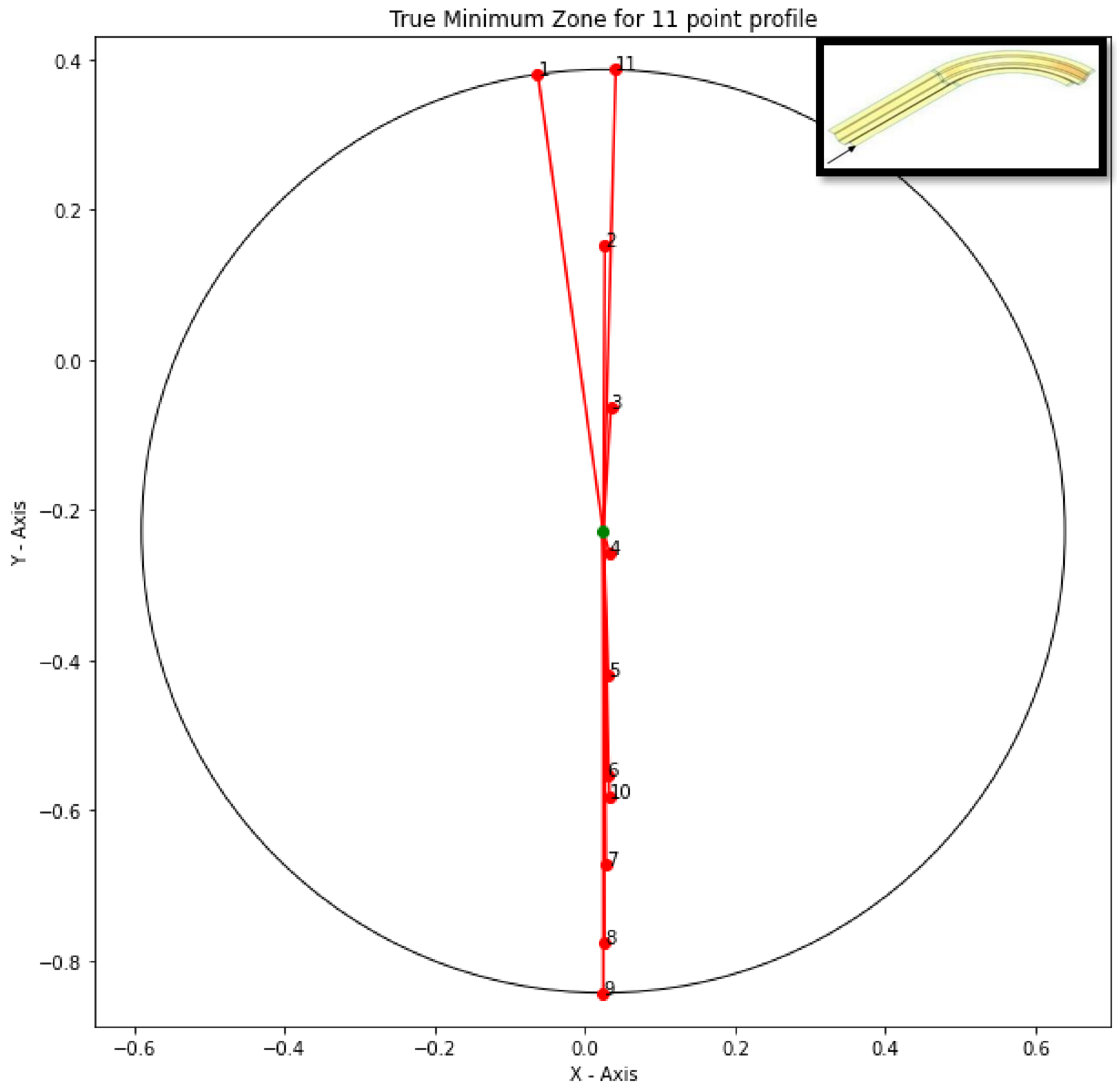


Fig 7.5 True Minimum Zone Boundary for 11 Points on the Straight Portion of the Inner Profile.

For True Minimum Zone fit the zone magnitude (radius in mm) is 0.6143731306047372 and the line coordinates of the true minimum zone axis is

[0.0986587098, 0.23447333435999998, -499.15010329853556, 114.08398211599791, 12.278273535654483, 0.028316745057440418]

9 Point Arc Profile

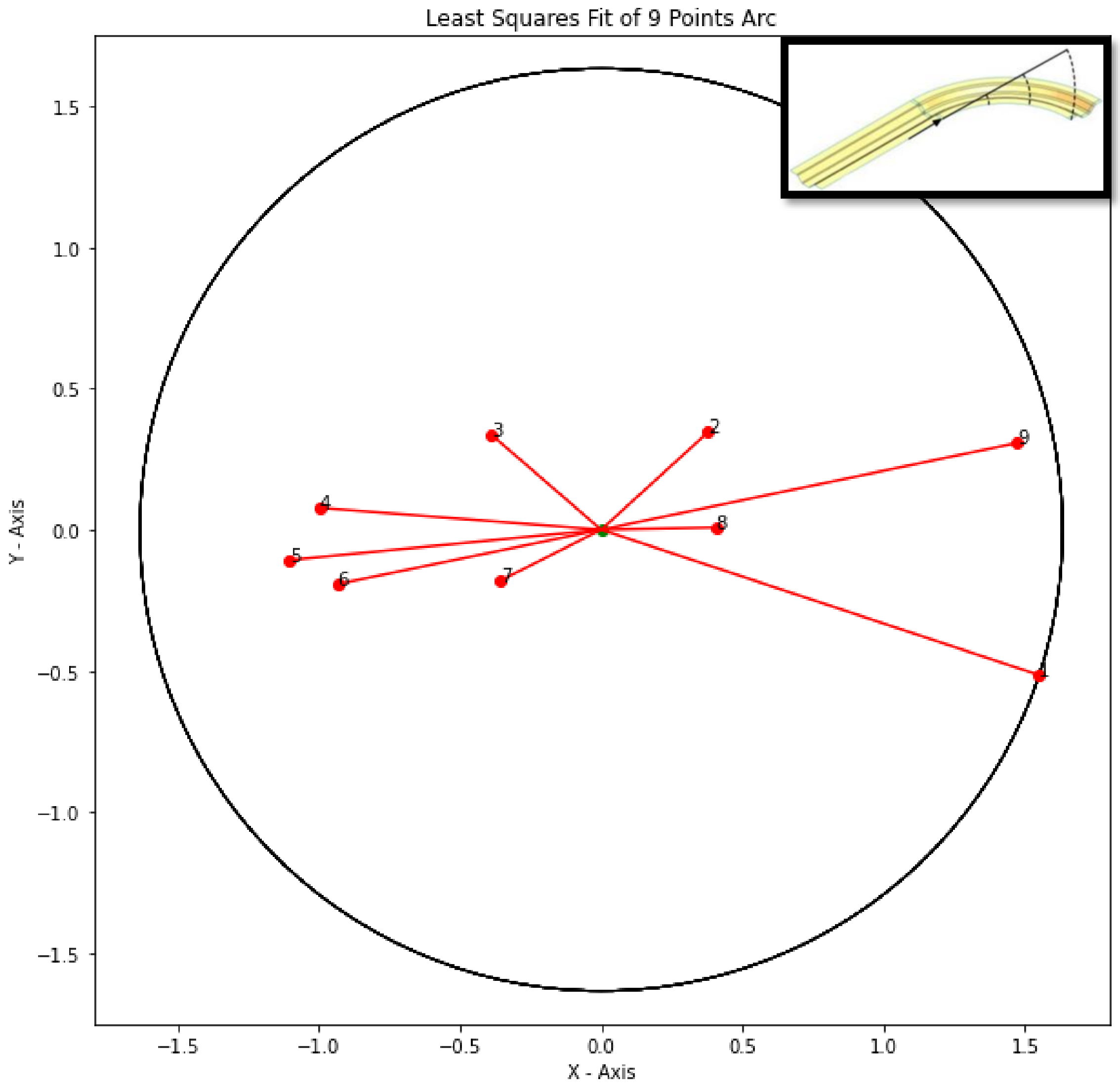


Fig 7.6 Projection of Toroidal Least Squares Zone Boundary for 9 Points on the Arc Portion of the Inner Profile.

The screw coordinates of the SDT for the Least Squares Fit are L = -0.6810376565080728, M = 0, N = 0.7322483939333648, P = 168.2171450051991, Q = 271.51507039120753, and R = 156.45266164864367. The zone magnitude (radius in mm) is 1.6342358007856808

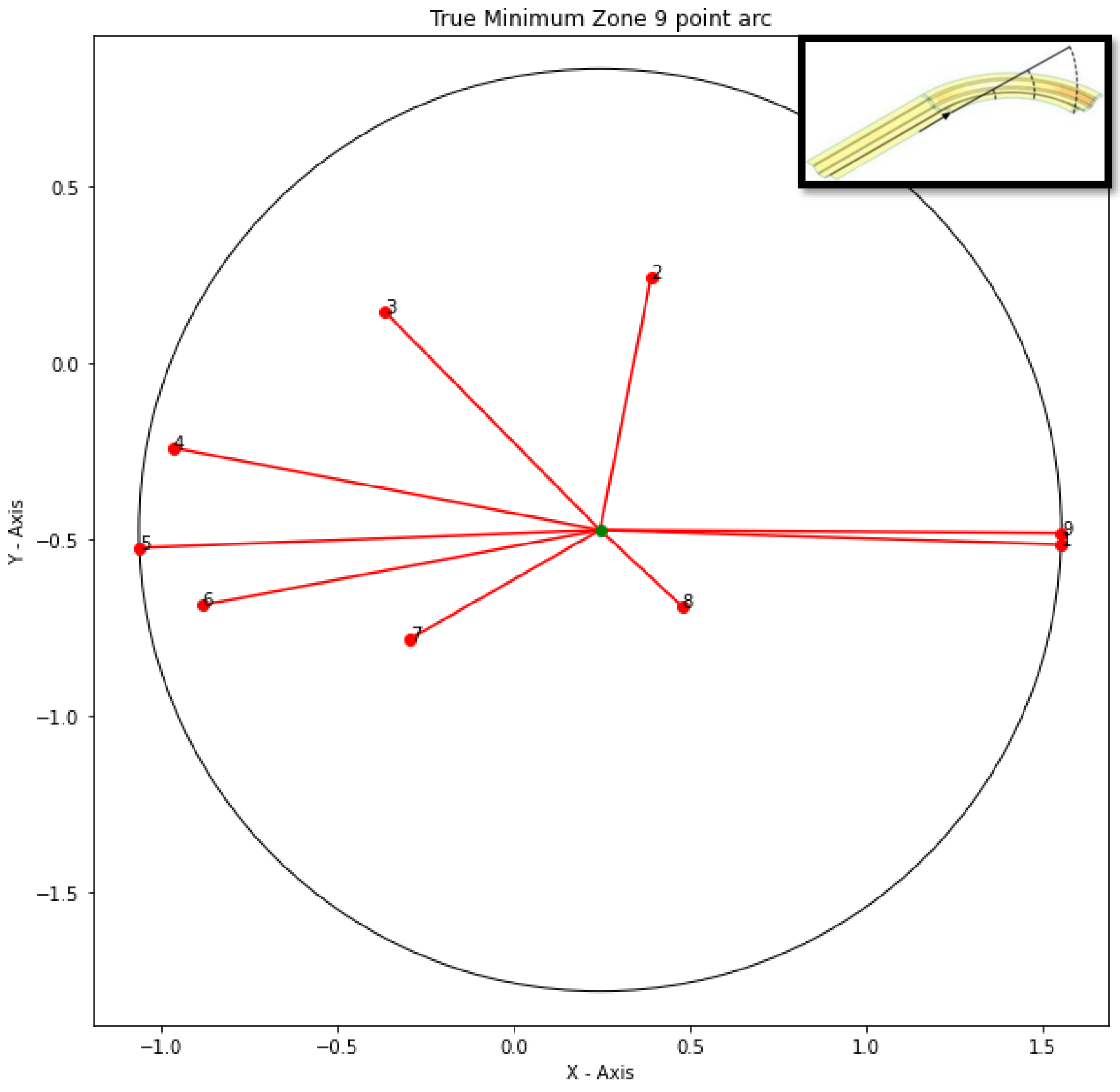


Fig 7.7 Projection of Toroidal True Minimum Zone Boundary for 9 Points on the Arc Portion of the Inner Profile.

For True Minimum Zone fit the zone magnitude (radius in mm) is 1.3081461525525635 and the line coordiantes of the true minimum zone axis is [-0.322208176, 1.263710878, -386.19293186126146, 182.99174177142638, 94.00693285842485, 0.15493848640106325]

9 Point Arc Profile with Size Change

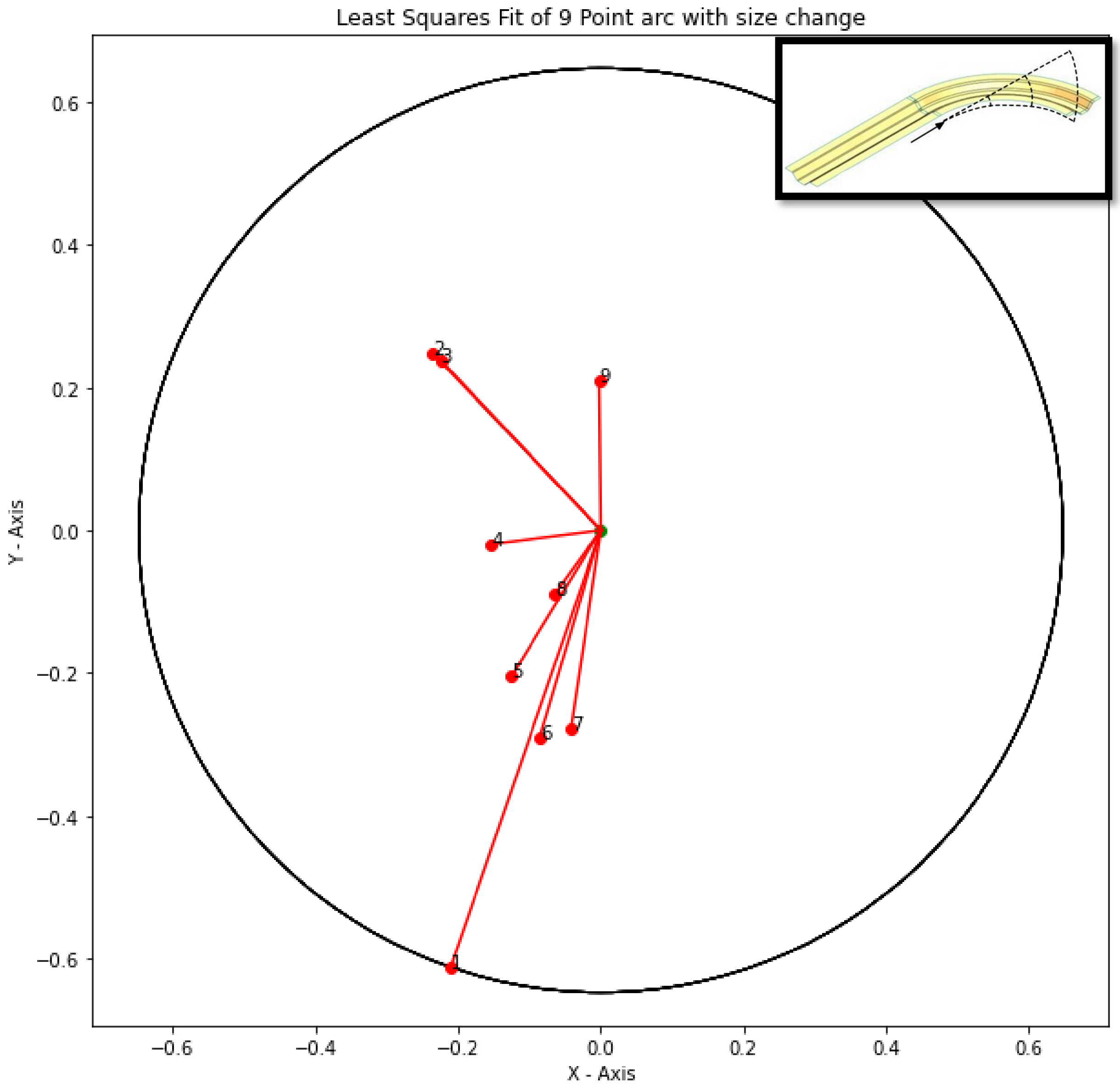


Fig 7.8 Projection of Toroidal Least Squares Zone Boundary for 9 Points on the Arc Portion of the Inner Profile with Size Change.

The screw coordinates of the SDT for the Least Squares Fit are L = -0.6810376565080728, M = 0, N = 0.7322483939333648, P = -425.28231911758354, Q = 271.51507039120753, and R = -395.5396507056241. The zone magnitude (radius in mm) is 0.6087075040987826

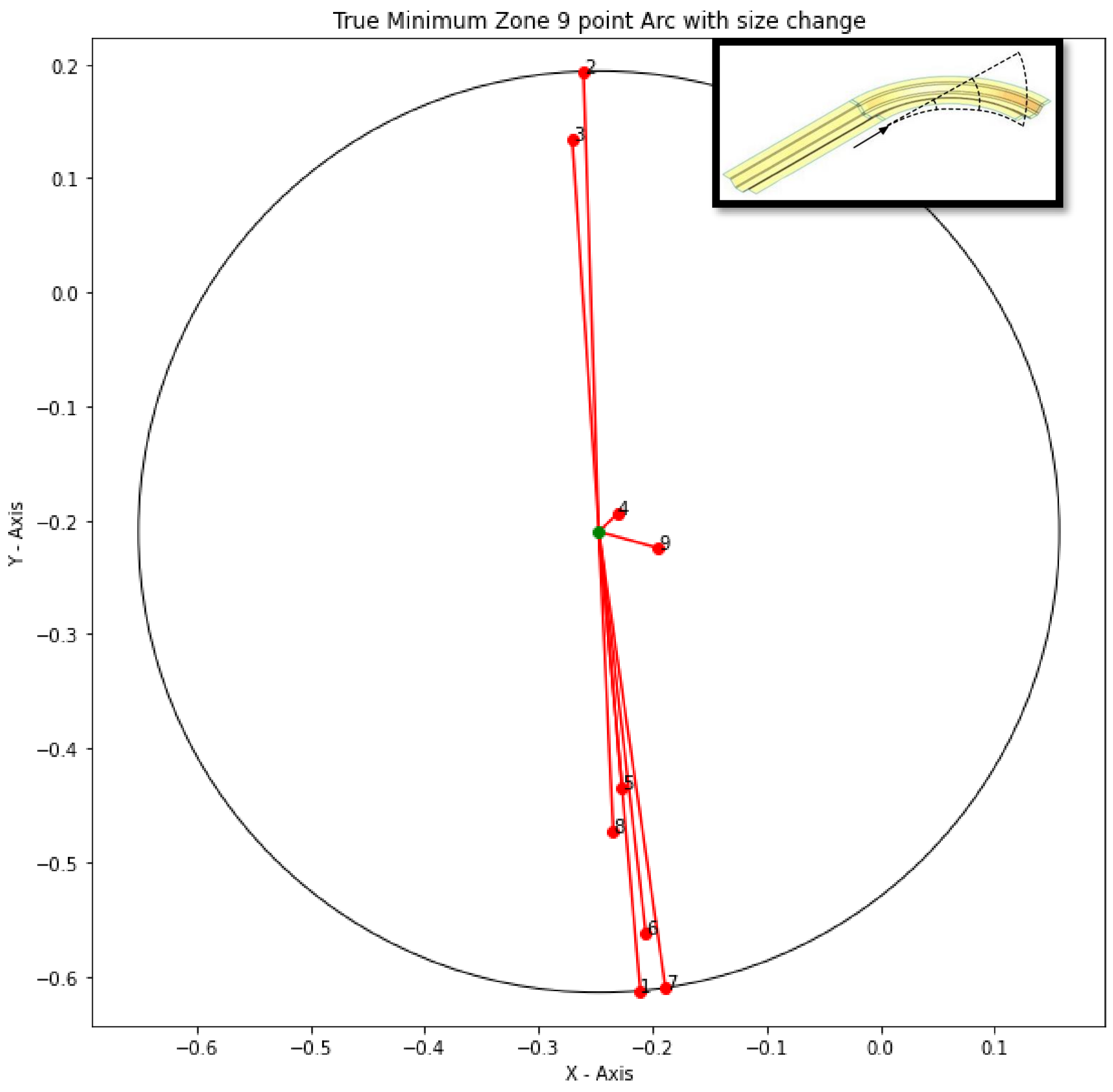


Fig 7.9 Projection of Toroidal True Minimum Zone Boundary for 9 Points on the Arc Portion of the Inner Profile with Size Change.

For True Minimum Zone fit the zone magnitude (radius in mm) is 0.4038519137982327 and the line coordinates of the true minimum zone axis is [0.409833541, 0.609878762, -398.99300613250654, 69.86577447600716, -88.65369719531978, -0.06374715584750915]

19 Point J-shape Profile

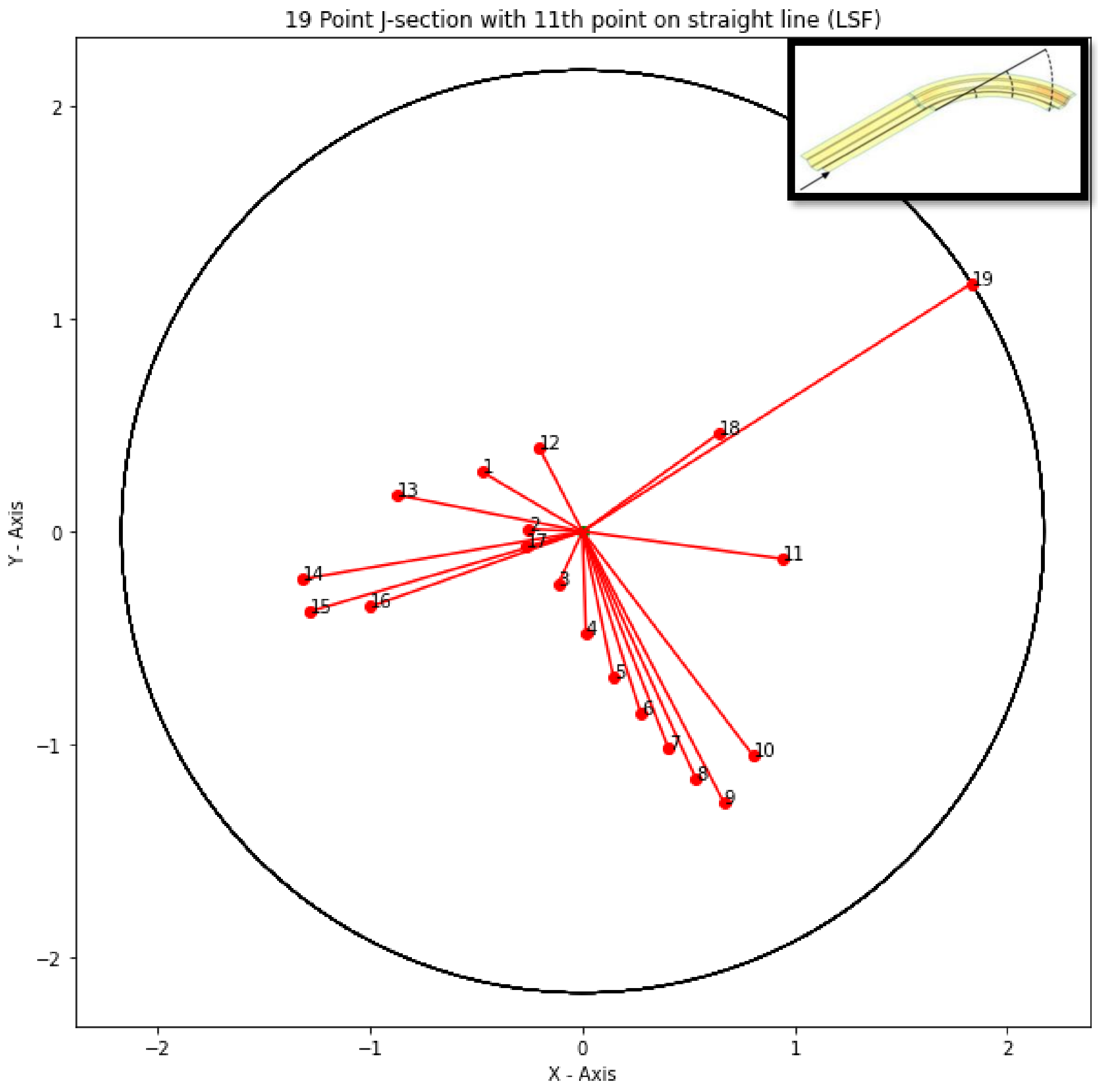


Fig 7.10 Least Squares Zone Boundary for 19 Points on the J-shaped Inner Profile with Size Change.

The screw coordinates of the SDT for the Least Squares Fit are L = 0.0236007377696814, M = 0.62019014063756, N = -0.784096419219396, P = 2971.90601742116, Q = -814.554362251714, and R = -554.82897157168. The zone magnitude (radius in mm) is 2.1721925461560327

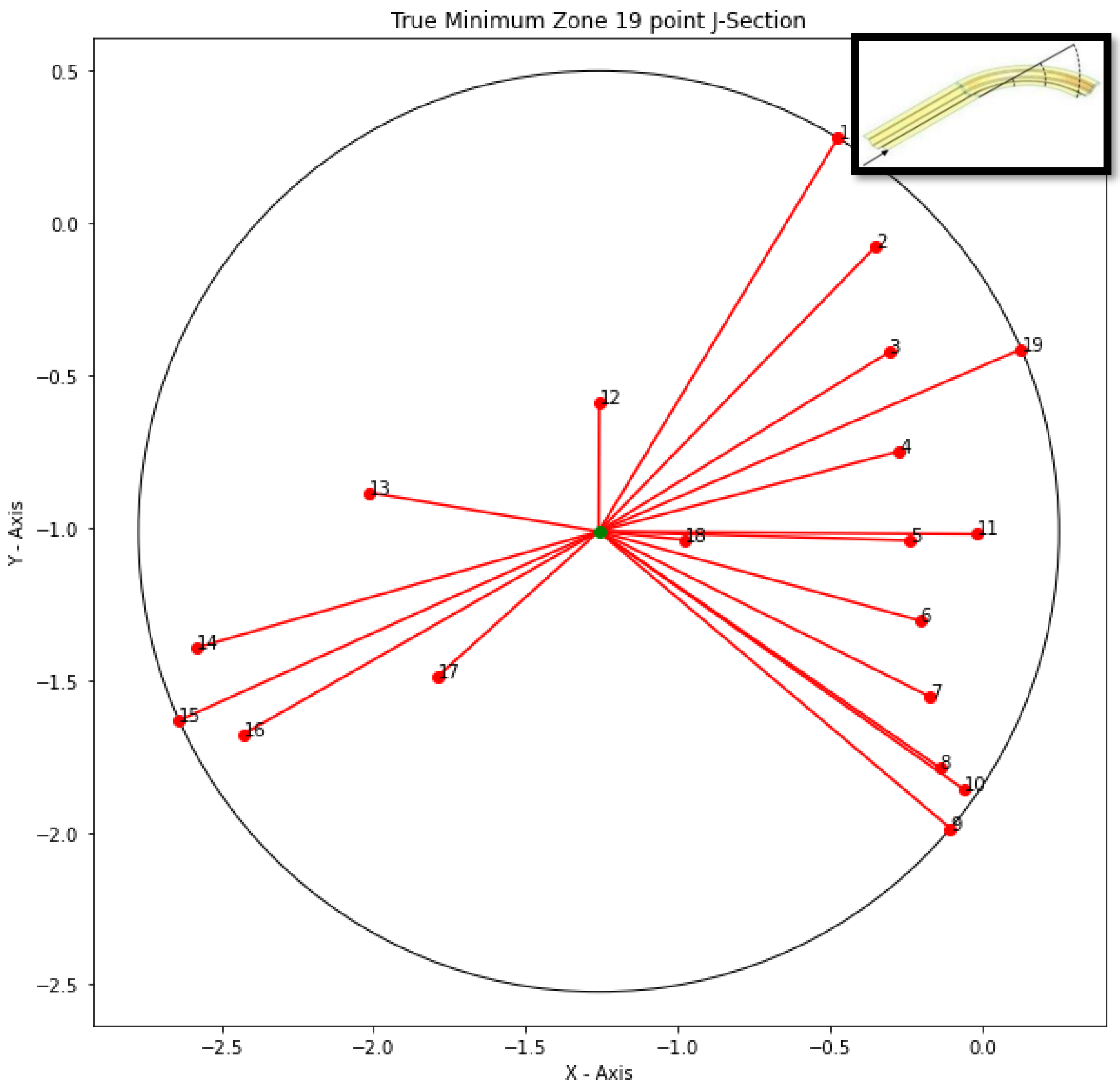


Fig 7.11 True Minimum Zone Boundary for 19 Points on the J-shaped Inner Profile When Unwrapped onto a Straight Line.

The true minimum zone fit magnitude (radius in mm) is 1.5127055727360417 and the line coordinates of the true minimum zone axis is [3.05001632, 2.49843188, -884.988161426439, 876.0182223181955, -1142.7886289042638, -0.20713233900502792]

19 Point J-shape Profile with Size Change

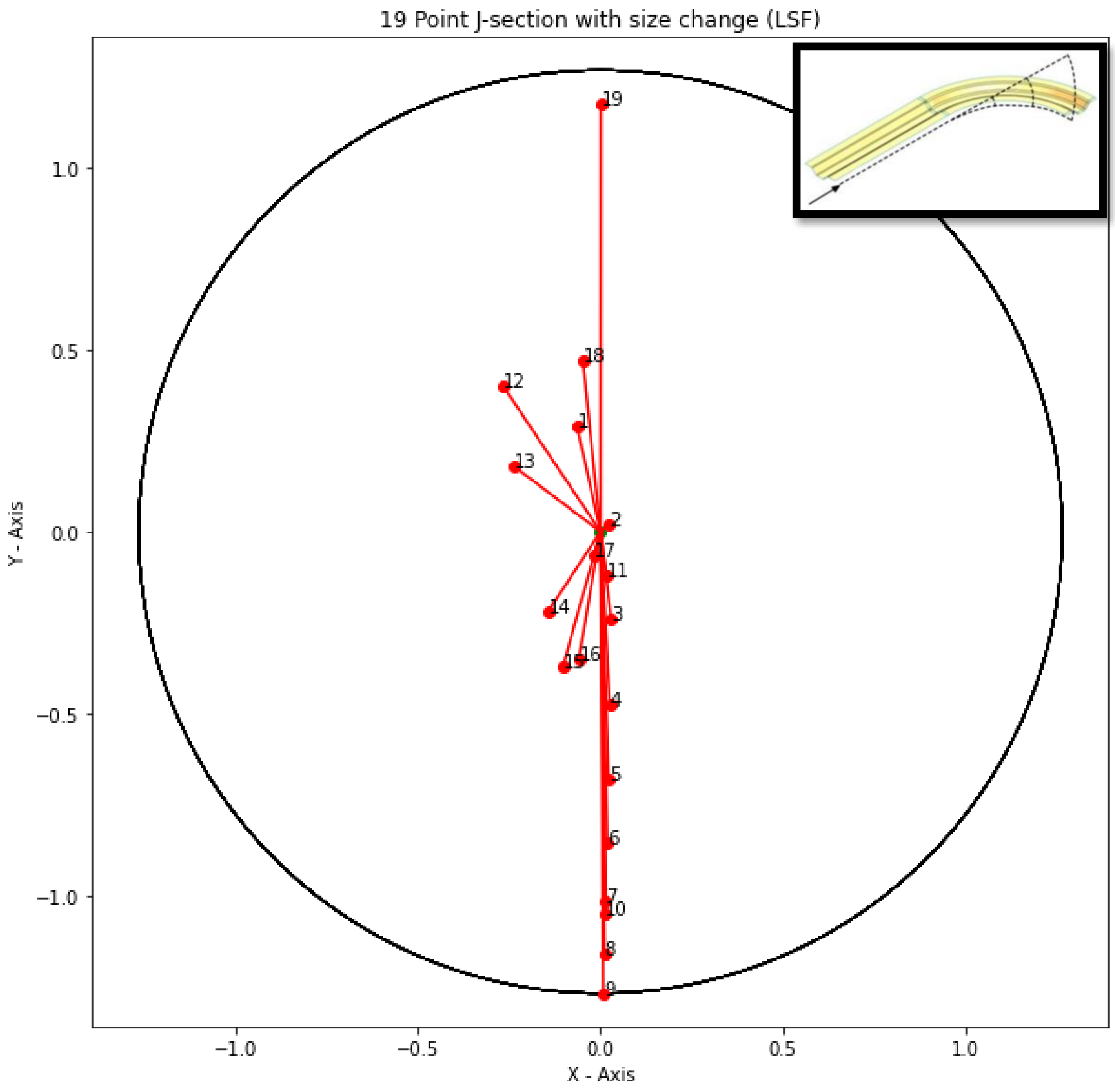


Fig 7.12 Least Squares Zone Boundary for 19 Points on the J-shaped Inner Profile with Size Change When Unwrapped onto a Straight Line.

The screw coordinates of the SDT for the Least Squares Fit are L = 0.026503637579415, M = -0.47322916679475, N = -0.88054057992224, P = 163.864006522574, Q = 1912.6525655547, and R = -1022.98520734124. The zone magnitude (radius in mm) is 1.2666840840111688.

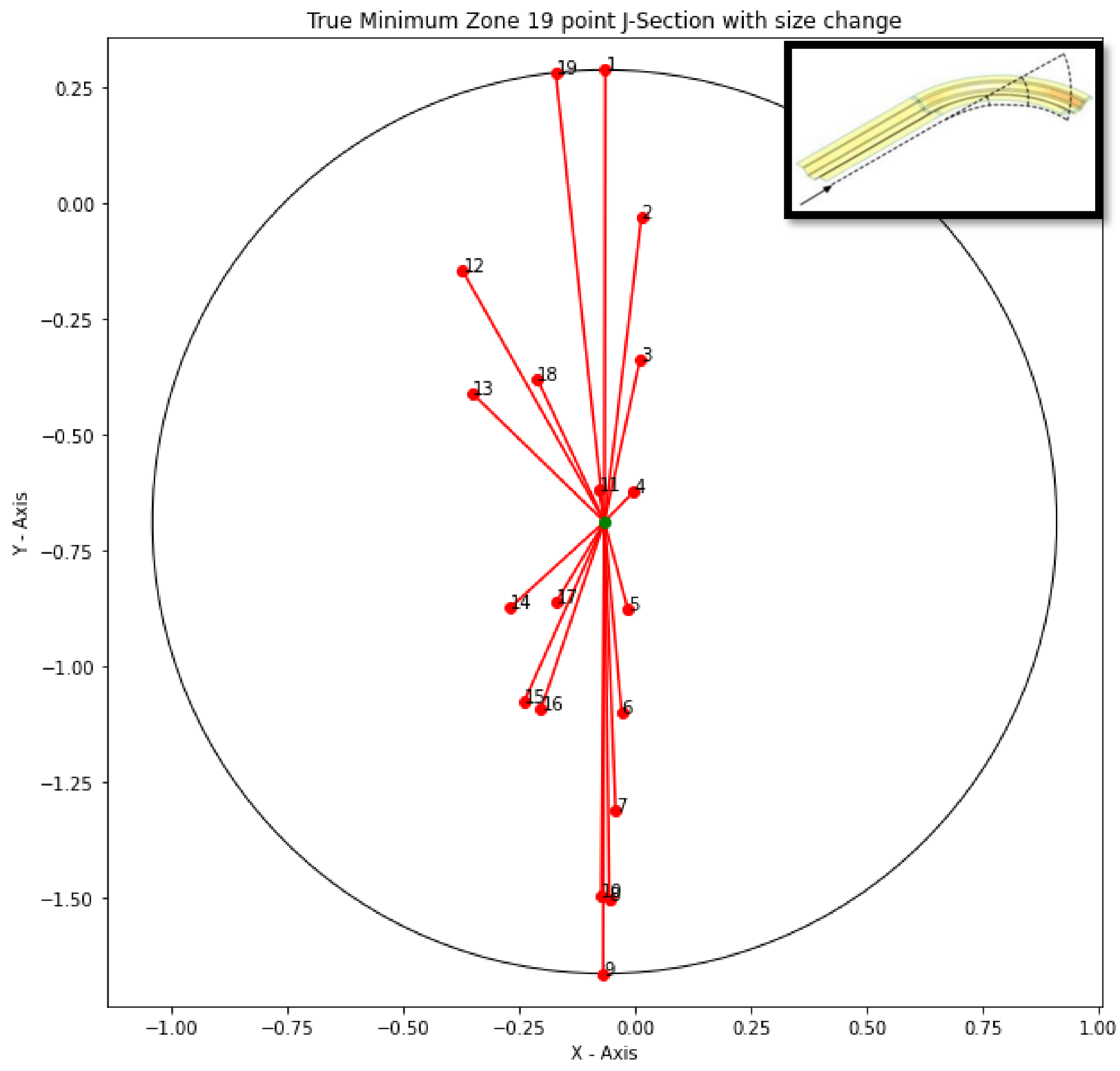


Fig 7.13 True Minimum Zone Boundary for 19 Points on J-shaped Inner Profile with Size Change When Unwrapped onto a Straight Line.

For True Minimum Zone fit the zone magnitude (radius in mm) is 0.9758224692007229 and the line coordiantes of the true minimum zone axis is [0.23889283760000002, 1.57789168, -897.9557096779063, 616.80679705544, -59.71919153901355, 0.05915693832255892]

REFERENCES

[1]. Joshi, A. (2020). Quantifying Deformations in Flexible Assemblies Using Least Square Fit and Capture Zone Techniques (MS Thesis, Arizona State University).

[2]. Adrian, A., Ramnath, S., Sunkara, S. C., Korkolis, Y., Davidson, J. K., & Shah, J. J. (2022). Curating datasets of flexible assemblies to predict spring-back behavior for machine learning purposes. *Volume 2: Manufacturing Processes; Manufacturing Systems*. https://doi.org/10.1115/msec2022-85718

[3]. Shah, J.J., Detwiler, D.T., Korkolis, Y., and Davidson, J. K., 2001, "Collaborative Research, GOALI: Mapping Design Space with Distributed Machine Learning Networks for Precision Engineering of Flexible Assemblies", NSF GOALI Award No. 2029905 to Ohio State University and Arizona State University.

[4]. A. Makinouchi, E. Nakamchi, E. Onate, and R. Wagoner, "NUMISHEET '93," 1993.

[5]. Davidson, J. K., & Hunt, K. H. (2004). *Robots and screw theory: Applications of kinematics and statics to robotics*. Oxford University Press.

[6]. Mohan, P., Haghighi, P., Shah, J. J., and Davidson, J. K. (2015). Development of a library of feature fitting algorithms for CMMs. *International Journal of Precision Engineering and Manufacturing*, *16*(10), 2101–2113. https://doi.org/10.1007/s12541-015-0272-1

[7]. Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1986). *Numerical Recipes*. Cambridge University Press.

[8]. Hansen, P. C., Pereyra, V. and Scherer, G. (2014). *Least squares data fitting with applications*. Johns Hopkins University Press.

[9]. Srinivasan, V., Shakarji, C.M. and Morse, E.P. (2013). On the Enduring Appeal of Least-squares Fitting in Computational Coordinate Metrology. *Journal of Computing and Information Science in Engineering*, https://doi.org/10.1115/1.3647877

[10]. Eberly, D. (2017). Least squares fitting of data. https://www.cse.iitb.ac.in/~cs749/spr2017/handouts/eberly_least_square_fitting.pdf Last accessed, April, 2023.

[11]. Gohari, H. and Barari, A. (2016). A quick deviation zone fitting in coordinate metrology of NURBS surfaces using principle component analysis. *Measurement, 92, 352–364*. https://doi.org/10.1016/j.measurement.2016.05.050

[12]. Strang, G. (2021). *Introduction to linear algebra*. Cambridge University Press.

[13]. Shakarji, C. M. and Srinivasan, V. (2013). Theory and algorithms for weighted total least-squares fitting of lines, planes, and parallel planes to support tolerancing standards. *Journal of Computing and Information Science in Engineering, 13(3)*. https://doi.org/10.1115/1.4024854.

[14]. Nassef, A. O. and ElMaraghy, H. A. (1999). Determination of best objective function for evaluating geometric deviations. *The International Journal of Advanced Manufacturing Technology, 15(2)*, 90–95. https://doi.org/10.1007/s001700050044.

[15]. Davidson, J. K., Savaliya, S. B., He, Y. and Shah, J. J. (2012). Methods of robotics and the pseudoinverse to obtain the least-squares fit of measured points on line-profiles. Volume 5: 6th International Conference on Micro- and Nanosystems; 17th Design for Manufacturing and the Life Cycle Conference. https://doi.org/10.1115/detc2012-70203

[16]. Davidson, J., Savaliya, S. and Shah, J. J. (2013). Least-squares fit of measured points for square line-profiles. Procedia CIRP, 10, 203–210. https://doi.org/10.1016/j.procir.2013.08.032

[17]. Davidson, J.K., Savaliya, S., and Shah, J.J. (2013). Screws and Robotics for Metrology. In Advances in Mechanisms, Robotics and Design Education and Research (eds V. Kumar, J. Schmiedeler, S.V. Sreenivasan, and H-J Su).

[18]. He, Y., Kalish, N.J., Davidson, J.K., and Shah, J.J. (2016). Tolerance-Maps for Line-Profiles Formed by Intersecting Kinematically Transformed Primitive T-Map Elements. J. of Computing and Information Science in Engrg., Vol. 16(2), https://doi.org/10.1115/1.4033236.

[19]. Liu, S. C., Hu, S. J., and Woo, T. C. (1996). Tolerance analysis for Sheet Metal Assemblies. Journal of Mechanical Design, 118(1), 62–67. https://doi.org/10.1115/1.2826857

[20]. J. Y. Lee, M.-G. Lee, and F. Barlat, “Evaluation of Constitutive Models for Springback Prediction in U-draw⁄bending of DP and TRIP Steel Sheets,” in AIP Conference Proceedings, 2011, pp. 571–578.

[21]. Lei, X., Song, H., Xue, Y., Li, J., Zhou, J., and Duan, M. (2011). Method for cylindricity error evaluation using geometry optimization searching algorithm. Measurement, 44(9), 1556–1563. https://doi.org/10.1016/j.measurement.2011.06.010

APPENDIX

A. 11 – POINT LINE PROFILE FITTING

A program has been developed to find the least squares fit of points along a 3D line profile. The way of implementations is first we need to take the points along the nominal profile in our case the points are along the nominal profile which is the Z-axis we first change the Origin by -250mm along the axis which changes the points and now we find the line coordinates for each point which is [0, Z, 0; 1, 0, 0] when taken along the XZ plane and [-Z, 0, 0; 0, 1, 0] along the YZ plane and these line coordinates of each point we can find a set of equations which doesn't have a definite solution so we go for least squares method. which gives the SDT of the LSF line with the nominal profile.

The equations involve are.

$$[\$_{L-S}] = [k']^{\#}[d] = (([k']^{T}[k'])^{-1}[k']^{T})[d]$$

Where the $\$_{L-S}$ is the angular and positional components which are used to find the small displacement torsor (SDT) which gives the location of the Least Squares line with respect to the nominal profile. The K is the matrix developed with the line coordinates of each actuator to node along the profile and the d is the deflections of the points along the axis in this based on the view the deflection values are given by the X and Y coordinates of the points. Using the values of $\$_{L-S}$ we can find the *phi* which is the angular orientation and *h* which is pitch of the line and using these we can find the [L, M, N; P, Q, R] of the SDT using these values we can generate a transformation matrix as mentioned in [4] using which we can find the transformed points which lies on the least squares line. Using which we can find the zone magnitude. The below program deals with these.

```
import numpy as np
import matplotlib.pyplot as plt
### Initialization of the parameters
OrignalPoints = np.array([[-2.4190e+02, 7.4250e-01, 4.9002e+02],
      [-2.4184e+02, 5.3490e-01, 4.5102e+02],
      [-2.4179e+02, 3.0300e-01, 4.0103e+02],
      [-2.4175e+02, 9.9200e-02, 3.5103e+02],
      [-2.4171e+02, -6.6700e-02, 3.0103e+02],
      [-2.4167e+02, -1.9560e-01, 2.5103e+02],
      [-2.4163e+02, -3.0480e-01, 2.0102e+02],
      [-2.4160e+02, -3.9400e-01, 1.5103e+02],
      [-2.4156e+02, -4.4430e-01, 1.0103e+02],
      [-2.4153e+02, -2.0430e-01, 5.1040e+01],
      [-2.4150e+02, 9.8220e-01, 1.0420e+00]])
dist_from_origin_to_pts = 250 # To transform the origin to (-250,0,0)
itrsize = len(OrignalPoints)
di = []
kprime = []
### Creating a Deflection 'd' Matrix
for i in range (itrsize):
  di.append(dist_from_origin_to_pts+OrignalPoints[i][0])
  di.append(OrignalPoints[i][1])
di = np.transpose(np.reshape(di,(2*itrsize)))
### Creating a K matrix
for i in range (itrsize):
  kprime.append([0,OrignalPoints[i][2],1,0])
  kprime.append([-OrignalPoints[i][2],0,0,1])
kprime = np.reshape(kprime,(2*itrsize,4))
### Applying Least squares fit for the equations
kprTranspose=np.transpose(kprime)
intermediate=np.linalg.inv(np.matmul(kprTranspose,kprime))
kmoorePenrose=np.matmul(intermediate,kprTranspose)
pd=np.matmul(kmoorePenrose,di)
deltatheta_x=pd[0]*(180/np.pi)
deltatheta_y=pd[1]*(180/np.pi)
deltatheta_z=0
delta_x=pd[2]
delta_y=pd[3]
delta_z=0
delta_r=0
print("deltatheta_x={},   deltatheta_y={},   delta_x={},
delta_y={}".format(deltatheta_x,deltatheta_y,delta_x,delta_y))
### Finding the values of PHI, h(pitch) and s
phi=np.sqrt((np.power(pd[0],2))+(np.power(pd[1],2)))
h=((pd[0]*pd[2])+(pd[1]*pd[3]))/((np.power(pd[0],2))+(np.power(pd[1],2)))
s=phi*h
print("Phi={}, Pitch(h)={}, s={}".format(phi*(180/np.pi), h*(np.pi)/180, s))
```

```
    ### Finding the Plucker Coordinates [L, M, N; P, Q, R]
    l=pd[0]/phi
    m=pd[1]/phi
    n=0
    p=(delta_x-(h*pd[0]))/phi
    q=(delta_y-(h*pd[1]))/phi
    r=0
    print("L={},   M={},   N={}".format(l,m,n))
    print("P={},   Q={},   R={}".format(p,q,r))
    ### Distance of the SDT line coordinates on the nominal profile axis
    zt1=-p/m
    zt2=q/l
    print("Distance of the least squares line in the coordinate axis is {}".format(zt1))
    ### Finding the Nominal X,Y and Z of the Least Squares fit Line
    vphi=1-np.cos(phi)
    x_nom=vphi*(m*r-n*q)+p*np.sin(phi)+l*s
    y_nom=vphi*(n*p-l*r)+q*np.sin(phi)+m*s
    z_nom=vphi*(l*q-m*p)+r*np.sin(phi)+n*s
    # print("vphi={},   X_nom={},   Y_nom={},   Z_nom={}".format(vphi,x_nom,y_nom,z_nom))
    ### Creating a Matrix to find the trasformed points
    trnomls1=np.array([(vphi*np.power(l,2)+np.cos(phi)), (vphi*l*m-n*np.sin(phi)),
(vphi*l*n+m*np.sin(phi)), x_nom])
    trnomls2=np.array([(vphi*l*m+n*np.sin(phi)), (vphi*np.power(m,2)+np.cos(phi)), (vphi*m*n-
l*np.sin(phi)), y_nom])
    trnomls3=np.array([(vphi*l*n-m*np.sin(phi)), (vphi*n*m+l*np.sin(phi)),
(vphi*np.power(n,2)+np.cos(phi)), z_nom])
    trnomls4=np.array([0,0,0,1])
    trnomls=np.vstack([trnomls1,trnomls2,trnomls3,trnomls4])
    invtrnomls=np.linalg.inv(trnomls)
    # print("The trasformation matrix is \n{}".format(invtrnomls))
    ### Original points transformed along the Leasts squares line
    origptset=[]
    for i in range (itrsize):
    origptset.append([dist_from_origin_to_pts+OrignalPoints[i][0],OrignalPoints[i][1],OrignalPoints[
i][2],1])
    origptset=np.reshape(origptset,(itrsize,4))  # The origin is translated to (-250,0,0)
    origptsetCol=np.transpose(origptset)
    ptset = np.matmul(invtrnomls,origptsetCol)
    print("The transformed points of the original points along the Least squares line is
\n{}".format(repr(np.transpose(ptset[0:3])))) # Transformed Points
    ### Finding the least Squares Sum
    lssum=0
    for i in range (itrsize):
      lssum=lssum+np.power((ptset[0,i]),2)+np.power((ptset[1,i]),2)
    print("The least squares sum value is {}".format(lssum))
```

```
### Finding the Zone magnitude
ptlsdist=[]
for i in range (itrsize):
    ptls=np.sqrt((np.power(ptset[0,i],2))+(np.power(ptset[1,i],2)))
    ptlsdist.append(ptls)
Ptlstdist=np.hstack(ptlsdist)
zonemag=np.max(ptlsdist)
print("The zone magnitude (radius) is {}".format(zonemag))
### Plotting of points along the z-axis
center=(0,0)
circledist=[]
XCords=ptset[0]
YCords=ptset[1]
ZCords=ptset[2]
for i in range (itrsize):
  circledist.append(np.sqrt(np.square(XCords[i])+np.square(YCords[i])))
plt.figure(figsize=(10, 10))
plt.axis('equal')
plt.plot(center[0],center[1],'go')
plt.title('Least Squares Fit of 11 Points')
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.axis('equal')
plt.gca().add_patch(plt.Circle((center[0],center[1]),max(circledist),fill=False))
for i in range (itrsize):
  plt.plot([center[0],XCords[i]],[center[1],YCords[i]],'r')
  plt.plot(XCords[i],YCords[i],'ro')
  plt.text(XCords[i],YCords[i],i+1)
plt.show()
```

The output of the program is as follows:

```
deltatheta_x=-0.0540075905907143, deltatheta_y=-0.045283898104306164,

delta_x=8.517611415805774,   delta_y=-0.14003470744737345


Phi=0.07048014804847813, Pitch(h)=-91.32940177382942, s=-6.436909758198438


L=-0.76628089023829, M=-0.6425057176832076, N=0

P=2914.478061710077, Q=-3475.936139152482, R=0


The zone magnitude (radius) is 1.1213798503766341.
```

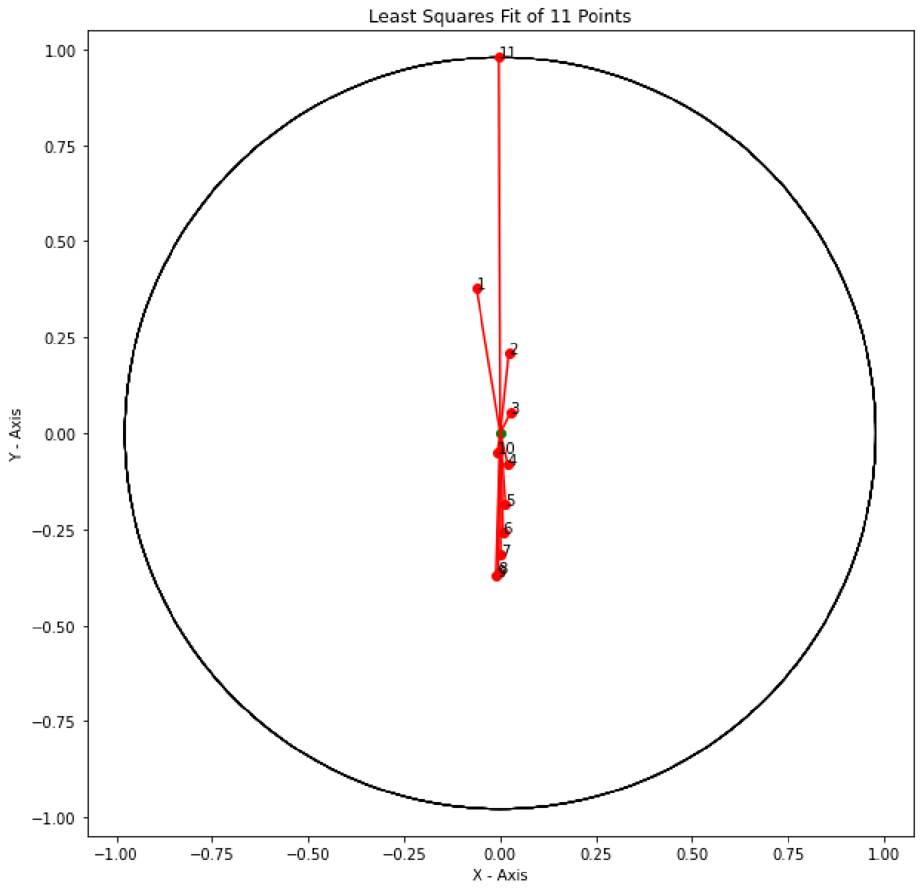


Fig A.1. Least Squares Zone Boundary for 11 Points on the Straight Portion of the Inner Profile.

B. 9 POINT ARC FITTING

For dealing with the Least squares fit for an arc which is spanned over the XZ plane with the arc of 250mm radius as the nominal profile. We first found the angles the points have been making along the XZ plane and the magnitude of length in the XZ plane from the origin now using these values. We need to find the forces and torques applied by the actuators of assumed platform robot and find the line coordinates and the deflections caused by them and using these we can find the matrices K and d and using these by applying the least squares we can find the angular movement and displacement of the nominal profile we can also find the line coordinates of the Small Displacement Torsor

(SDT). We can find the transformation matrix which gives the transformed coordinates along the Least squares fit line we could determine the zone size using these coordinates. The Program to find this least square profile is give below.

```
import numpy as np
import matplotlib.pyplot as plt
### Initialization of the parameters
OriginalPoints=np.array([[-2.4150e+02, 9.8220e-01, 1.0420e+00],
    [-2.3920e+02, 1.7540e+00, -4.5510e+01],
    [-2.2611e+02, 1.5195e+00, -9.5780e+01],
    [-2.0703e+02, 1.3136e+00, -1.3502e+02],
    [-1.7507e+02, 1.2240e+00, -1.7691e+02],
    [-1.4182e+02, 1.3043e+00, -2.0599e+02],
    [-9.5550e+01, 1.6049e+00, -2.3231e+02],
    [-5.3220e+01, 2.0476e+00, -2.4617e+02],
    [-2.4100e-02, 2.7479e+00, -2.5212e+02]])
itrsize = len(OriginalPoints)
### Finding the Angles made by the points along the xz plane
alpha=[]
for i in range (itrsize):
  alphas=np.arctan2(OriginalPoints[i,0],OriginalPoints[i,2])
  alpha.append(alphas)
alp=np.hstack(alpha)
print("The alpha values are {}".format(alp))
### Finding the magnitudes of the coordinates in the xz plane
magnitude=[]
for i in range (itrsize):
  magnitudes=np.sqrt(np.power(OriginalPoints[i,0],2)+np.power(OriginalPoints[i,2],2))
  magnitude.append(magnitudes)
mag=np.hstack(magnitude)
### Initializing the Deflection Values (d matrix)
dlvalue=[]
for i in range (itrsize):
  dlvalue.append((250-mag[i]))
  dlvalue.append(OriginalPoints[i,1])
di=np.hstack(dlvalue)
### Initializing of the Kprime matrix
kprime=[]
for i in range (itrsize):
  kprime.append([0,0,-OriginalPoints[i,0]/mag[i],0,-OriginalPoints[i,2]/mag[i]])
  kprime.append([-250*np.cos(alp[i]),250*np.sin(alp[i]),0,1,0])
kprime=np.reshape(kprime,((itrsize*2),5))
### Applying least squares fit to the equations
```

```
kprTranspose=np.transpose(kprime)
intermediate=np.linalg.inv(np.matmul(kprTranspose,kprime))
kmoorePenrose=np.matmul(intermediate,kprTranspose)
pd=np.matmul(kmoorePenrose,di)
deltatheta_x=((180*pd[0])/np.pi)
deltatheta_y=0
deltatheta_z=((180*pd[1])/np.pi)
delta_x=pd[2]
delta_y=pd[3]
delta_z=pd[4]
delta_r=0
print ("deltatheta_x = {}, deltatheta_z = {}".format(deltatheta_x,deltatheta_z))
print("delta_x = {}, delta_y = {}, delta_z = {}".format(delta_x,delta_y,delta_z))
### Finding the values of PHI, h(pitch) and S
phi=np.sqrt(np.power(pd[0],2)+np.power(pd[1],2))
h=((pd[0]*pd[2])+(pd[1]*pd[4]))/(np.power(pd[0],2)+np.power(pd[1],2))
s=phi*h
print("Phi={}, Pitch(h)={}, s={}".format(phi*(180/np.pi), h*(np.pi)/180, s))
### Finding the Plucker Coordinates
l=pd[0]/phi
m=0
n=pd[1]/phi
p=(delta_x - h*pd[0])/phi
q=delta_y/phi
r=(delta_z - h*pd[1])/phi
print("L={},   M={},   N={},   P={},   Q={},   R={}".format(l,m,n,p,q,r))
### Finding the Nominal X,Y and Z of the least squares fit line
x_nomls=p*np.sin(phi)+l*s
y_nomls=q*np.sin(phi)+m*s
z_nomls=r*np.sin(phi)+n*s
# print("The Nominal X, Y and Z of the least squares line are
{},{},{}".format(x_nomls,y_nomls,z_nomls))
### Creating a Matrix to find the trasformed points
tr1=np.array([np.cos(phi), -n*np.sin(phi), m*np.sin(phi), x_nomls])
tr2=np.array([n*np.sin(phi), np.cos(phi), -l*np.sin(phi), y_nomls])
tr3=np.array([-m*np.sin(phi), l*np.sin(phi), np.cos(phi), z_nomls])
tr4=np.array([0,0,0,1])
tr_nomls=np.vstack([tr1,tr2,tr3,tr4])
invtrnomls=np.linalg.inv(tr_nomls)
# print("The trasformation matrix is \n{}".format(invtrnomls))
### Original points transformed along hte least squares line
origptset=np.append(OriginalPoints,np.ones((itrsize,1)),axis=1)
origptsetCol=np.transpose(origptset)
ptset = np.matmul(invtrnomls,origptsetCol)
TransformedPoints=np.transpose(ptset[0:3])
```

```
# print("The transformed points of the original points along the Least squares line is
\n{}".format(TransformedPoints)) # Transformed Points
### Finding the zone magnitude
pltsdist=[]
for i in range (itrsize):
    magnitudels=np.sqrt(np.power(ptset[0,i],2)+np.power(ptset[2,i],2))
    pltsdist.append(np.sqrt(np.power((250-magnitudels),2)+np.power(ptset[1,i],2)))
zonemag=np.max(pltsdist)
print("The zone magnitude (radius) is {}".format(zonemag))
### Unwrapping of points on the least squares fit arc
newcords=[]
for i in range (itrsize):
    alp=np.arctan2(TransformedPoints[i,0],TransformedPoints[i,2])
    mag=np.sqrt(np.power(TransformedPoints[i,0],2)+np.power(TransformedPoints[i,2],2))
    newcords.append([250-mag, TransformedPoints[i,1], 250*((np.pi/2)+alp)])
newcords=np.reshape(newcords,(itrsize,3))
print("The unwrapped coordinates are {}".format(repr(newcords)))
### Plotting of points along the z-axis
center=(0,0)
circledist=[]
XCords=newcords[:,0]
YCords=newcords[:,1]
ZCords=newcords[:,2]
for i in range (len(XCords)):
  circledist.append(np.sqrt(np.square(XCords[i])+np.square(YCords[i])))
plt.figure(figsize=(10, 10))
plt.plot(center[0],center[1],'go')
plt.axis('equal')
plt.title('Least Squares Fit of 9 Points Arc')
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.gca().add_patch(plt.Circle((center[0],center[1]),max(circledist),fill=False))

for i in range (len(newcords)):
  plt.plot([center[0],XCords[i]],[center[1],YCords[i]],'r')
  plt.plot(XCords[i],YCords[i],'ro')
  plt.text(XCords[i],YCords[i],i+1)
plt.show()
```

The Output of the above program is:

```
deltatheta_x = -0.2379620481478811, deltatheta_z = 0.47753821566048144 delta_x =

6.823709569304667, delta_y = 3.549024554723155, delta_z = -3.8105229466022563
```

Phi=0.533543516289844, Pitch(h)=-12.096357730920312, s=-6.453933238055062

L=-0.446003073568623, M=0, N=0.8950314287036748, P=423.668055036129, Q=381.12004394308434, R=211.1180106743644

The zone magnitude (radius) is 1.6947389152410268.

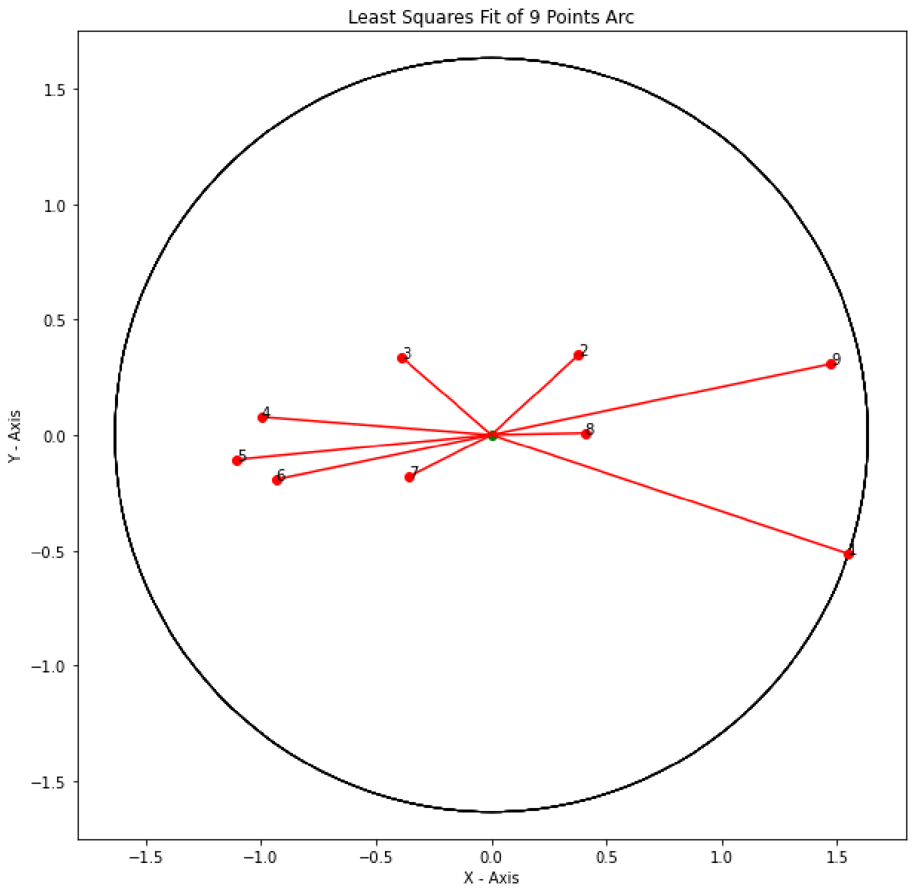


Fig B.1. Projection of Toroidal Least Squares Zone Boundary for 9 Points on the Arc Portion of the Inner Profile.

C. 9 POINT ARC WITH SIZE CHANGE

This program is like the previous 9-point arc program, but this includes the size change. The nominal shape in the previous case is fixed which is an arc of 250mm radius in the xz plane. This code allows the flexibility of nominal shape to be of suitable appropriate radius for better fitting options. The code for this is as follows.

```
import numpy as np
import matplotlib.pyplot as plt

### Initialization of the parameters
OriginalPoints=np.array([[-2.4150e+02, 9.8220e-01, 1.0420e+00],
      [-2.3920e+02, 1.7540e+00, -4.5510e+01],
      [-2.2611e+02, 1.5195e+00, -9.5780e+01],
      [-2.0703e+02, 1.3136e+00, -1.3502e+02],
      [-1.7507e+02, 1.2240e+00, -1.7691e+02],
      [-1.4182e+02, 1.3043e+00, -2.0599e+02],
      [-9.5550e+01, 1.6049e+00, -2.3231e+02],
      [-5.3220e+01, 2.0476e+00, -2.4617e+02],
      [-2.4100e-02, 2.7479e+00, -2.5212e+02]])
itrsize = len(OriginalPoints) ## for looping through the points and creation of matrices
### Finding the Angles made by the points along the xz plane
alpha=[]
for i in range (itrsize):
   alphas=np.arctan2(OriginalPoints[i,0],OriginalPoints[i,2])
   alpha.append(alphas)
alp=np.hstack(alpha)
### Finding the magnitudes of the coordinates in the xz plane
magnitude=[]
for i in range (itrsize):
   magnitudes=np.sqrt(np.power(OriginalPoints[i,0],2)+np.power(OriginalPoints[i,2],2))
   magnitude.append(magnitudes)
mag=np.hstack(magnitude)
### Initializing the Deflection Values (d matrix)
dlvalue=[]
for i in range (itrsize):
   dlvalue.append((250-mag[i]))
   dlvalue.append(OriginalPoints[i,1])
di=np.hstack(dlvalue)
### Initializing of the Kprime matrix
kprime=[]
for i in range (itrsize):
   kprime.append([0,0,-OriginalPoints[i,0]/mag[i],0,-OriginalPoints[i,2]/mag[i], -1])
   kprime.append([-250*np.cos(alp[i]),250*np.sin(alp[i]),0,1,0,0])
kprime=np.reshape(kprime,((itrsize*2),6))
### Applying least squares fit to the equations to get the angular displacement and location of the
Nominal shape with respect to Least squares line
kprTranspose=np.transpose(kprime)
intermediate=np.linalg.inv(np.matmul(kprTranspose,kprime))
kmoorePenrose=np.matmul(intermediate,kprTranspose)
pd=np.matmul(kmoorePenrose,di)
```

```
deltatheta_x=((180*pd[0])/np.pi)
deltatheta_y=0
deltatheta_z=((180*pd[1])/np.pi)
delta_x=pd[2]
delta_y=pd[3]
delta_z=pd[4]
delta_r=pd[5]
print ("deltatheta_x = {}, deltatheta_z = {}".format(deltatheta_x,deltatheta_z))
print("delta_x = {}, delta_y = {}, delta_z = {}, delta_r = {}".format(delta_x,delta_y,delta_z,delta_r))
### Finding the values of PHI, h(pitch) and S
phi=np.sqrt(np.power(pd[0],2)+np.power(pd[1],2))
h=((pd[0]*pd[2])+(pd[1]*pd[4]))/(np.power(pd[0],2)+np.power(pd[1],2))
s=phi*h
print("Phi={}, Pitch(h)={}, s={}".format(phi*(180/np.pi), h*(np.pi)/180, s))
### Finding the Plucker Coordinates [L, M, N; P, Q, R]
l=pd[0]/phi
m=0
n=pd[1]/phi
p=(delta_x - h*pd[0])/phi
q=delta_y/phi
r=(delta_z - h*pd[1])/phi
print("L={},   M={},   N={},   P={},   Q={},   R={}".format(l,m,n,p,q,r))
### Finding the Nominal X,Y and Z of the least squares fit line
vp=1-np.cos(phi)
x_nomls=p*np.sin(phi)+l*s+vp*(m*r-n*q)
y_nomls=q*np.sin(phi)+m*s+vp*(n*p-l*r)
z_nomls=r*np.sin(phi)+n*s+vp*(l*q-m*p)
# print("The Nominal X, Y and Z of the least squares line are
{},{},{}".format(x_nomls,y_nomls,z_nomls))
### Creating a Matrix to find the trasformed points
tr1=np.array([vp*(np.square(l))+np.cos(phi), vp*m*l-n*np.sin(phi), vp*l*n+m*np.sin(phi), x_nomls])
tr2=np.array([vp*m*l+n*np.sin(phi), vp*(np.square(m))+np.cos(phi), vp*m*n-l*np.sin(phi),
y_nomls])
tr3=np.array([vp*n*l-m*np.sin(phi), vp*m*n+l*np.sin(phi), vp*(np.square(n))+np.cos(phi),
z_nomls])
tr4=np.array([0,0,0,1])
tr_nomls=np.vstack([tr1,tr2,tr3,tr4])
invtrnomls=np.linalg.inv(tr_nomls)
# print("The trasformation matrix is \n{}".format(invtrnomls))
### Original points transformed along hte least squares line
origptset=np.append(OriginalPoints,np.ones((itrsize,1)),axis=1)
origptsetCol=np.transpose(origptset)
ptset = np.matmul(invtrnomls,origptsetCol)
TransformedPoints=np.transpose(ptset[0:3])
```

```
# print("The transformed points of the original points along the Least squares line is
\n{}".format(TransformedPoints)) # Transformed Points
### Finding the zone magnitude
pltsdist=[]
for i in range (itrsize):
    magnitudels=np.sqrt(np.power(ptset[0,i],2)+np.power(ptset[2,i],2))
    pltsdist.append(np.sqrt(np.power((250+delta_r-magnitudels),2)+np.power(ptset[1,i],2)))
zonemag=np.max(pltsdist)
print("The zone magnitude (radius) is {}".format(zonemag))
### Unwrapping of points on the least squares fit arc
newcords=[]
for i in range (itrsize):
    alp=np.arctan2(TransformedPoints[i,0],TransformedPoints[i,2])
mag=np.sqrt(np.power(TransformedPoints[i,0],2)+np.power(TransformedPoints[i,2],2))
    newcords.append([250+delta_r-mag, TransformedPoints[i,1], (250+delta_r)*((np.pi/2)+alp)])
newcords=np.reshape(newcords,(itrsize,3))
print("The unwrapped points are {}".format(repr(newcords)))
### Plotting of points along the z-axis
center=(0,0)
circledist=[]
XCords=newcords[:,0]
YCords=newcords[:,1]
ZCords=newcords[:,2]
for i in range (len(XCords)):
  circledist.append(np.sqrt(np.square(XCords[i])+np.square(YCords[i])))
plt.figure(figsize=(25, 25))
plt.plot(center[0],center[1],'go')
plt.axis('equal')
plt.title('Least Squares Fit of 9 Points Arc with size change')
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.gca().add_patch(plt.Circle((center[0],center[1]),max(circledist),fill=False))
for i in range (len(newcords)):
  plt.plot([center[0],XCords[i]],[center[1],YCords[i]],'r')
  plt.plot(XCords[i],YCords[i],'ro')
  plt.text(XCords[i],YCords[i],i+1)
plt.show()
```

The output of the above code is:

```
deltatheta_x = -0.2379620481478811, deltatheta_z = 0.47753821566048144 delta_x =

0.43504684441878094, delta_y = 3.549024554723155, delta_z = -10.137169495160746,

delta_r = -8.001115119283417
```

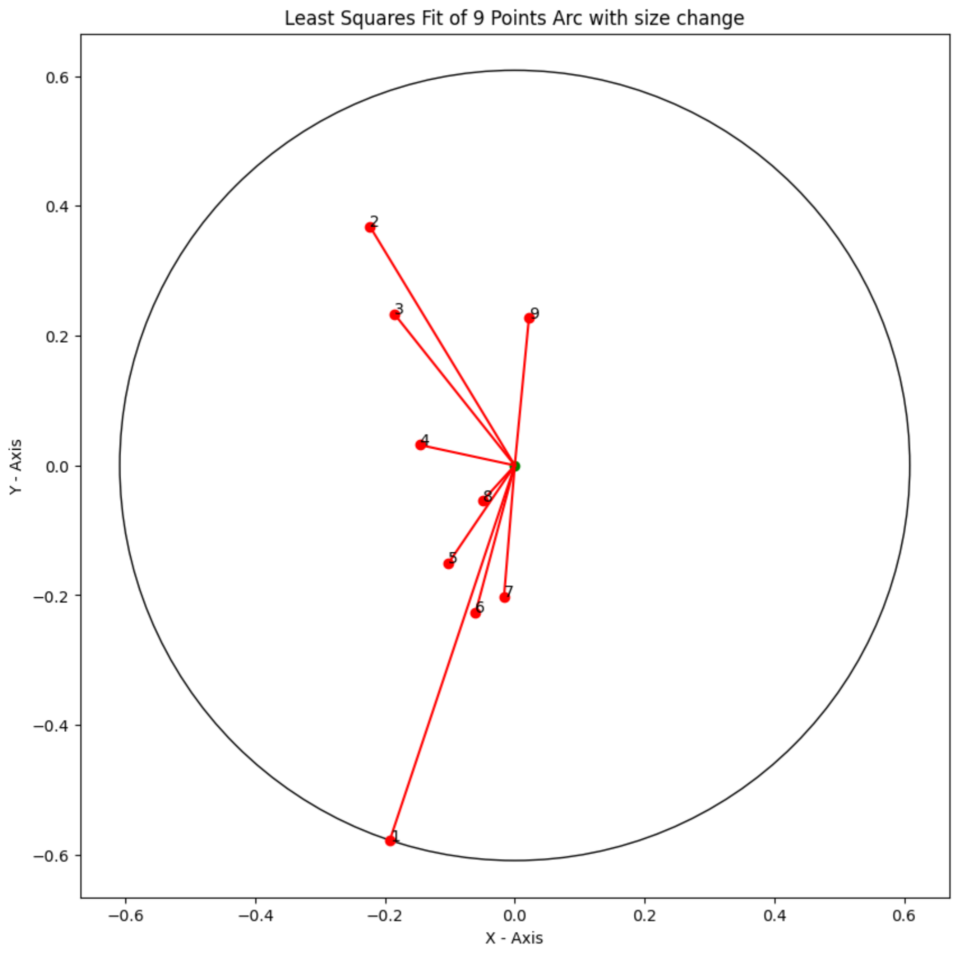


Fig C.1. Projection of Toroidal Least Squares Zone Boundary for 9 Points on the Arc Portion of the Inner Profile with Size Change.

Phi=0.533543516289844, Pitch(h)=-17.36900035907819, s=-9.26711752602214

L=-0.446003073568623, M=0, N=0.8950314287036748, P=-397.1305725766778, Q=381.12004394308434, R=-197.89411890686418

The zone magnitude (radius) is 0.6088761796412765.

D. 19 POINTS J SHAPED PROFILE FITTING

The J-Shaped profile is the combination of both the straight line and the arc profile we have delt in the previous cases. The program is to find the Least Squares fit of the given profile in the 3D space. Similar to the others we assume a nominal profile for the J-section and take the screw coordinates of the points as described in the chapter 2 using which we form a set of equations which doesn't have a definite solution and so we go for the least squares solution. Which gives the angular orientation and location of the least squares fit profile with respect to the Nominal profile. The program is given below.

```
import numpy as np
import matplotlib.pyplot as plt
### Initialization of the parameters
OriginalPoints=np.array([[-2.4190e+02, 7.4250e-01, 4.9002e+02],
     [-2.4184e+02, 5.3490e-01, 4.5102e+02],
     [-2.4179e+02, 3.0300e-01, 4.0103e+02],
     [-2.4175e+02, 9.9200e-02, 3.5103e+02],
     [-2.4171e+02, -6.6700e-02, 3.0103e+02],
     [-2.4167e+02, -1.9560e-01, 2.5103e+02],
     [-2.4163e+02, -3.0480e-01, 2.0102e+02],
     [-2.4160e+02, -3.9400e-01, 1.5103e+02],
     [-2.4156e+02, -4.4430e-01, 1.0103e+02],
     [-2.4153e+02, -2.0430e-01, 5.1040e+01],
     [-2.4150e+02, 9.8220e-01, 1.0420e+00],
     [-2.3920e+02, 1.7540e+00, -4.5510e+01],
     [-2.2611e+02, 1.5195e+00, -9.5780e+01],
     [-2.0703e+02, 1.3136e+00, -1.3502e+02],
     [-1.7507e+02, 1.2240e+00, -1.7691e+02],
     [-1.4182e+02, 1.3043e+00, -2.0599e+02],
     [-9.5550e+01, 1.6049e+00, -2.3231e+02],
     [-5.3220e+01, 2.0476e+00, -2.4617e+02],
     [-2.4100e-02, 2.7479e+00, -2.5212e+02]])
dist_from_origin_to_pts=250 # To transform the origin to (-250,0,0)
itrsize=len(OriginalPoints)
### Creation of the deflections (d) matrix
deflections1 = []
deflections2 = []
for i in range (itrsize):
   if (OriginalPoints[i,2]>0):
      deflections1.append(dist_from_origin_to_pts+OriginalPoints[i,0])
      deflections1.append(OriginalPoints[i,1])
   else:
      deflections2.append(dist_from_origin_to_pts-
np.sqrt(np.square(OriginalPoints[i,0])+np.square(OriginalPoints[i,2])))
      deflections2.append(OriginalPoints[i,1])
di=np.transpose(np.append(deflections1,deflections2))
### Creation of the K matrix
```

```
kprime1=[]
kprime2=[]
for i in range (itrsize):
    if (OriginalPoints[i,2]>0):
        kprime1.append(np.array([0,OriginalPoints[i,2],0,1,0,0]))
        kprime1.append(np.array([-OriginalPoints[i,2],0,0,0,1,0]))
    else: mag=np.sqrt(np.square(OriginalPoints[i,0])+np.square(OriginalPoints[i,2]))
        alp=np.arctan2(OriginalPoints[i,0],OriginalPoints[i,2])
        kprime2.append(np.array([0,0,0,-OriginalPoints[i,0]/mag,0,-OriginalPoints[i,2]/mag]))
        kprime2.append(np.array([-250*np.cos(alp),0,250*np.sin(alp),0,1,0]))
kprime=np.append(kprime1,kprime2).reshape(2*len(OriginalPoints),6)
### Application of Least Squares fit to the equations
kprTranspose=np.transpose(kprime)
intermediate=np.linalg.inv(np.matmul(kprTranspose,kprime))
kmoorePenrose=np.matmul(intermediate,kprTranspose)
pd=np.matmul(kmoorePenrose,di)
deltatheta_x=pd[0]*(180/np.pi)
deltatheta_y=pd[1]*(180/np.pi)
deltatheta_z=pd[2]*(180/np.pi)
delta_x=pd[3]
delta_y=pd[4]
delta_z=pd[5]
delta_r=0
print("deltatheta_x = {}, deltatheta_y = {}, deltatheta_z = {}, delta_x = {}, delta_y = {},
delta_z={}".format(deltatheta_x,deltatheta_y,deltatheta_z,delta_x,delta_y,delta_z))
### Finding the values of PHI, h(pitch) and s
phi=np.sqrt((np.power(pd[0],2))+(np.power(pd[1],2))+(np.power(pd[2],2)))
h=((pd[0]*pd[3])+(pd[1]*pd[4])+(pd[2]*pd[5]))/((np.power(pd[0],2))+(np.power(pd[1],2))+(np.powe
r(pd[2],2)))
s=phi*h
print("Phi = {}, Pitch(h) = {}, s = {}".format(phi*(180/np.pi), h*(np.pi)/180, s))
### Finding the Plucker coordinates [L, M, N; P, Q, R]
l=pd[0]/phi
m=pd[1]/phi
n=pd[2]/phi
p=(delta_x-(h*pd[0]))/phi
q=(delta_y-(h*pd[1]))/phi
r=(delta_z-(h*pd[2]))/phi
print("L = {}, M = {}, N = {}".format(l,m,n))
print("P = {}, Q = {}, R = {}".format(p,q,r))
### Finding the nominal x,y and z of the least squares fit line
vphi=1-np.cos(phi)
x_nom=vphi*(m*r-n*q)+p*np.sin(phi)+l*s
y_nom=vphi*(n*p-l*r)+q*np.sin(phi)+m*s
z_nom=vphi*(l*q-m*p)+r*np.sin(phi)+n*s
```

```
print("vphi = {}, X_nom = {}, Y_nom = {}, Z_nom = {}".format(vphi,x_nom,y_nom,z_nom))
### Creation of matrix to find the transformed points
trnomls1=np.array([(vphi*np.power(l,2)+np.cos(phi)), (vphi*l*m-n*np.sin(phi)),
(vphi*l*n+m*np.sin(phi)), x_nom])
trnomls2=np.array([(vphi*l*m+n*np.sin(phi)), (vphi*np.power(m,2)+np.cos(phi)), (vphi*m*n-
l*np.sin(phi)), y_nom])
trnomls3=np.array([(vphi*l*n-m*np.sin(phi)), (vphi*n*m+l*np.sin(phi)),
(vphi*np.power(n,2)+np.cos(phi)), z_nom])
trnomls4=np.array([0,0,0,1])
trnomls=np.vstack([trnomls1,trnomls2,trnomls3,trnomls4])
invtrnomls=np.linalg.inv(trnomls)
# print("The trasformation matrix is \n{}".format(invtrnomls))
### Finding the Original points transformed along the Least Squares Line
origptset=np.append(OriginalPoints,np.ones((itrsize,1)),axis=1)
origptsetCol=np.transpose(origptset)
ptset = np.matmul(invtrnomls,origptsetCol)
# print("The transformed points are \n{}".format(repr(np.transpose(ptset[0:3]))))
### Finding the Least Squares fit zone magnitude
ptlsdistofl=[]
pltsdistofa=[]
for i in range (itrsize):
    if(OriginalPoints[i,2]>0):
        ptls=np.sqrt((np.power((dist_from_origin_to_pts+ptset[0,i]),2))+(np.power(ptset[1,i],2)))
        ptlsdistofl.append(ptls)
    else:
        magnitudels=np.sqrt(np.power(ptset[0,i],2)+np.power(ptset[2,i],2))
        pltsdistofa.append(np.sqrt(np.power((dist_from_origin_to_pts-
magnitudels),2)+np.power(ptset[1,i],2)))
pltsdist=np.append(ptlsdistofl,pltsdistofa)
zonemag=np.max(pltsdist)
print("The zone magnitude (radius) is {}".format(zonemag))
### unwrapping of points along the straight line with the (-250,0,0) as the origin
plot=(np.transpose(ptset[0:3]))
transformedline=[]
transformedarc=[]
for i in range (itrsize):
    if (OriginalPoints[i,2]>0):
        transformedline.append([plot[i,0]+dist_from_origin_to_pts, plot[i,1], plot[i,2]])
    else:
        alpha=np.arctan2(plot[i,0],plot[i,2])
        if alpha>0:
            alpha=alpha-(2*np.pi)
        magnitude=np.sqrt((plot[i,0]**2)+(plot[i,2]**2))
        transformedarc.append([dist_from_origin_to_pts-magnitude, plot[i,1],
dist_from_origin_to_pts*((np.pi/2)+alpha)])
```

```
transformedpoints=np.concatenate((transformedline,np.array(transformedarc)),axis=0)
print("The unwrapped points are {}".format(repr(transformedpoints)))
### Plotting of the transformed points along the z-axis
center=(0,0)
circledist=[]
XCords=transformedpoints[:,0]
YCords=transformedpoints[:,1]
ZCords=transformedpoints[:,2]
for i in range (len(XCords)):
  circledist.append(np.sqrt(np.square(XCords[i])+np.square(YCords[i])))
plt.figure(figsize=(25, 25))
plt.plot(center[0],center[1],'go')
plt.axis('equal')
plt.title('J-section profile (LSF)')
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.gca().add_patch(plt.Circle((center[0],center[1]),max(circledist),fill=False))
for i in range (len(transformedpoints)):
  plt.plot([center[0],XCords[i]],[center[1],YCords[i]],'r')
  plt.plot(XCords[i],YCords[i],'ro')
  plt.text(XCords[i],YCords[i],i+1)
plt.show()
```

The Output of the above program is:

```
deltatheta_x = 0.12479789780800658, deltatheta_y = 0.11991271915090945,

deltatheta_z = -0.11854568354822141, delta_x = 7.514101176990379, delta_y =

0.7973663292438798, delta_z=-4.198184893128284

Phi = 0.20977763131320173, Pitch(h) = 34.79099030044069, s = 7.298371536267024


L = 0.5949056485516375, M = 0.5716182340331493, N = -0.5651016402756049

P = 866.4271558359096, Q = -921.66890666297, R = -20.173970330501866


vphi = 6.702589301244011e-06, X_nom = 7.510525842660261, Y_nom =

0.7941725926354621, Z_nom = -4.205179359940644

The zone magnitude (radius) is 2.4135391159007806.
```

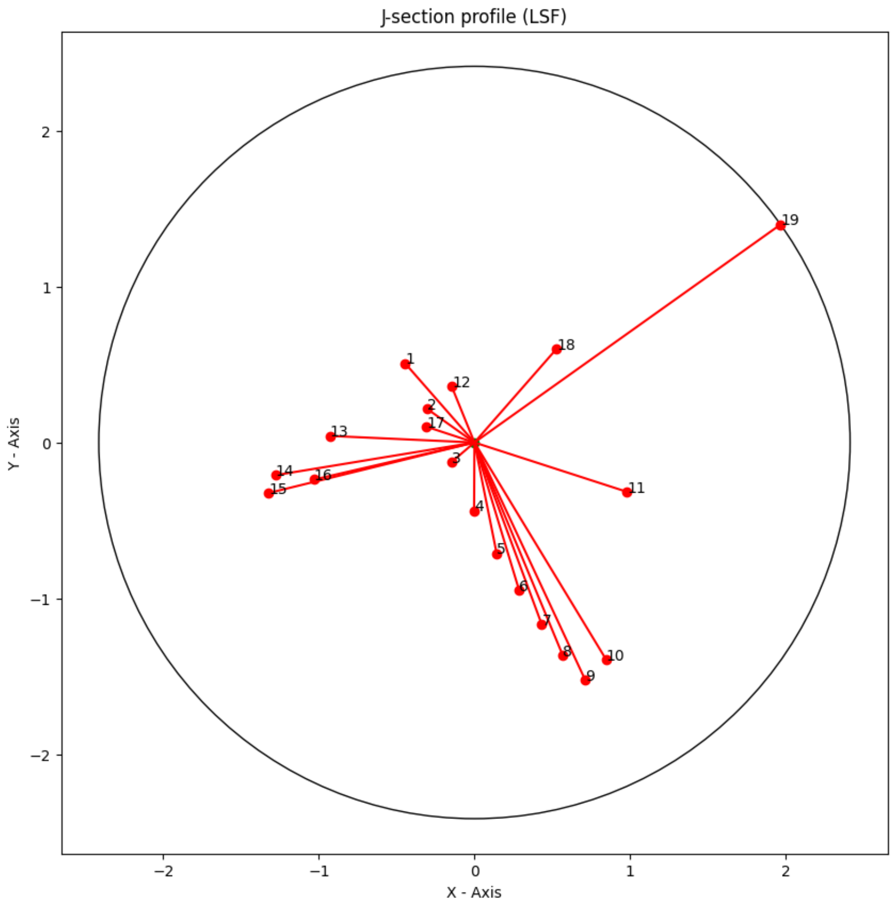


Fig D.1. Least Squares Zone Boundary for 19 Points on the J-shaped Inner Profile with Size Change.

E. J-SHAPED PROFILE WITH SIZE CHANGE

This program is like the previous least squares fit of the J-Shaped profile, but it incorporates size change which allows the total profile to move instead of fixing it at 250mm from the origin for both the straight-line profile and the arc section. The code for this is also pretty much like the previous program with a very small change.

```
import numpy as np
import matplotlib.pyplot as plt
### Initialization of the parameters
OriginalPoints=np.array([[-2.4190e+02, 7.4250e-01, 4.9002e+02],
    [-2.4184e+02, 5.3490e-01, 4.5102e+02],
    [-2.4179e+02, 3.0300e-01, 4.0103e+02],
    [-2.4175e+02, 9.9200e-02, 3.5103e+02],
    [-2.4171e+02, -6.6700e-02, 3.0103e+02],
    [-2.4167e+02, -1.9560e-01, 2.5103e+02],
    [-2.4163e+02, -3.0480e-01, 2.0102e+02],
    [-2.4160e+02, -3.9400e-01, 1.5103e+02],
    [-2.4156e+02, -4.4430e-01, 1.0103e+02],
    [-2.4153e+02, -2.0430e-01, 5.1040e+01],
    [-2.4150e+02, 9.8220e-01, 1.0420e+00],
    [-2.3920e+02, 1.7540e+00, -4.5510e+01],
    [-2.2611e+02, 1.5195e+00, -9.5780e+01],
    [-2.0703e+02, 1.3136e+00, -1.3502e+02],
    [-1.7507e+02, 1.2240e+00, -1.7691e+02],
    [-1.4182e+02, 1.3043e+00, -2.0599e+02],
    [-9.5550e+01, 1.6049e+00, -2.3231e+02],
    [-5.3220e+01, 2.0476e+00, -2.4617e+02],
    [-2.4100e-02, 2.7479e+00, -2.5212e+02]])
dist_from_origin_to_pts=250 # To transform the origin to (-250,0,0)
itrsize=len(OriginalPoints)
### Creation of the deflections (d) matrix
deflections1=[]
deflections2=[]
for i in range (itrsize):
  if (OriginalPoints[i,2]>0):
    deflections1.append(dist_from_origin_to_pts+OriginalPoints[i,0])
    deflections1.append(OriginalPoints[i,1])
  else:
    deflections2.append(dist_from_origin_to_pts-
np.sqrt(np.square(OriginalPoints[i,0])+np.square(OriginalPoints[i,2])))
    deflections2.append(OriginalPoints[i,1])
di=np.transpose(np.append(deflections1,deflections2))
### Creation of the K matrix
kprime1=[]
kprime2=[]
for i in range (itrsize):
  if (OriginalPoints[i,2]>0):
    kprime1.append(np.array([0,OriginalPoints[i,2],0,1,0,0,-1]))
    kprime1.append(np.array([-OriginalPoints[i,2],0,0,0,1,0,0]))
  else:
    mag=np.sqrt(np.square(OriginalPoints[i,0])+np.square(OriginalPoints[i,2]))
    alp=np.arctan2(OriginalPoints[i,0],OriginalPoints[i,2])
    kprime2.append(np.array([0,0,0,-OriginalPoints[i,0]/mag,0,-OriginalPoints[i,2]/mag,-1]))
    kprime2.append(np.array([-250*np.cos(alp),0,250*np.sin(alp),0,1,0,0]))
kprime=np.append(kprime1,kprime2).reshape(2*itrsize,7)
### Application of Least Squares fit to the equations
kprTranspose=np.transpose(kprime)
intermediate=np.linalg.inv(np.matmul(kprTranspose,kprime))
kmoorePenrose=np.matmul(intermediate,kprTranspose)
```

```
pd=np.matmul(kmoorePenrose,di)
deltatheta_x=pd[0]*(180/np.pi)
deltatheta_y=pd[1]*(180/np.pi)
deltatheta_z=pd[2]*(180/np.pi)
delta_x=pd[3]
delta_y=pd[4]
delta_z=pd[5]
delta_r=pd[6]
print("deltatheta_x = {}, deltatheta_y = {}, deltatheta_z = {}, delta_x = {}, delta_y = {}, delta_z = {},
delta_r = {}".format(deltatheta_x,deltatheta_y,deltatheta_z,delta_x,delta_y,delta_z,-delta_r))
### Finding the values of PHI, h(pitch) and s
phi=np.sqrt((np.power(pd[0],2))+(np.power(pd[1],2))+(np.power(pd[2],2)))
h=((pd[0]*pd[3])+(pd[1]*pd[4])+(pd[2]*pd[5]))/((np.power(pd[0],2))+(np.power(pd[1],2))+(np.powe
r(pd[2],2)))
s=phi*h
print("Phi = {}, Pitch(h) = {}, s = {}".format(phi*(180/np.pi), h*(np.pi)/180, s))
### Finding the Plucker coordinates [L, M, N; P, Q, R]
l=pd[0]/phi
m=pd[1]/phi
n=pd[2]/phi
p=(delta_x-(h*pd[0]))/phi
q=(delta_y-(h*pd[1]))/phi
r=(delta_z-(h*pd[2]))/phi
print("L = {}, M = {}, N = {}".format(l,m,n))
print("P = {}, Q = {}, R = {}".format(p,q,r))
### Finding the nominal x,y and z of the least squares fit line
vphi=1-np.cos(phi)
x_nom=vphi*(m*r-n*q)+p*np.sin(phi)+l*s
y_nom=vphi*(n*p-l*r)+q*np.sin(phi)+m*s
z_nom=vphi*(l*q-m*p)+r*np.sin(phi)+n*s
# print("vphi = {}, X_nom = {}, Y_nom = {}, Z_nom = {}".format(vphi,x_nom,y_nom,z_nom))
### Creation of matrix to find the transformed points
trnomls1=np.array([(vphi*np.power(l,2)+np.cos(phi)), (vphi*l*m-n*np.sin(phi)),
(vphi*l*n+m*np.sin(phi)), x_nom])
trnomls2=np.array([(vphi*l*m+n*np.sin(phi)), (vphi*np.power(m,2)+np.cos(phi)), (vphi*m*n-
l*np.sin(phi)), y_nom])
trnomls3=np.array([(vphi*l*n-m*np.sin(phi)), (vphi*n*m+l*np.sin(phi)),
(vphi*np.power(n,2)+np.cos(phi)), z_nom])
trnomls4=np.array([0,0,0,1])
trnomls=np.vstack([trnomls1,trnomls2,trnomls3,trnomls4])
invtrnomls=np.linalg.inv(trnomls)
# print("The trasformation matrix is \n{}".format(invtrnomls))
### Finding the Original points transformed along the Least Squares Line
origptset=np.append(OriginalPoints,np.ones((itrsize,1)),axis=1)
origptsetCol=np.transpose(origptset)
```

```
ptset = np.matmul(invtrnomls,origptsetCol)
# print("The transformed points are \n{}".format(np.transpose(ptset[0:3])))
### Finding the Least Squares fit zone magnitude
ptlsdistofl=[]
pltsdistofa=[]
for i in range (itrsize):
    if (OriginalPoints[i,2]>0):
        ptls=np.sqrt((np.power((dist_from_origin_to_pts+ptset[0,i]+delta_r),2))+(np.power(ptset[1,i],2)))
        ptlsdistofl.append(ptls)
    else:
        magnitudels=np.sqrt(np.power(ptset[0,i],2)+np.power(ptset[2,i],2))
        pltsdistofa.append(np.sqrt(np.power((dist_from_origin_to_pts+delta_r-
magnitudels),2)+np.power(ptset[1,i],2)))
pltsdist=np.append(ptlsdistofl,pltsdistofa)
zonemag=np.max(pltsdist)
print("The zone magnitude (radius) is {}".format(zonemag))
### unwrapping of points along the straight line with the (-250,0,0) as the origin
plot=(np.transpose(ptset[0:3]))
transformedline=[]
transformedarc=[]
for i in range (itrsize):
    if (OriginalPoints[i,2]>0):
        transformedline.append([plot[i,0]+dist_from_origin_to_pts+delta_r, plot[i,1], plot[i,2]])
    else:
        alpha=np.arctan2(plot[i,0],plot[i,2])
        magnitude=np.sqrt((plot[i,0]**2)+(plot[i,2]**2))
        transformedarc.append([dist_from_origin_to_pts+delta_r-magnitude, plot[i,1],
(dist_from_origin_to_pts+delta_r)*((np.pi/2)+alpha)])
transformedpoints=np.concatenate((transformedline,np.array(transformedarc)),axis=0)
print("The unwrapped points are {}".format(repr(transformedpoints)))
### Plotting of the transformed points along the z-axis
center=(0,0)
circledist=[]
XCords=transformedpoints[:,0]
YCords=transformedpoints[:,1]
ZCords=transformedpoints[:,2]
for i in range (len(XCords)):
  circledist.append(np.sqrt(np.square(XCords[i])+np.square(YCords[i])))
plt.figure(figsize=(25, 25))
plt.plot(center[0],center[1],'go')
plt.axis('equal')
plt.title('J-section Profile with size change (LSF)')
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.gca().add_patch(plt.Circle((center[0],center[1]),max(circledist),fill=False))
```

```
for i in range (len(transformedpoints)):
  plt.plot([center[0],XCords[i]],[center[1],YCords[i]],'r')
  plt.plot(XCords[i],YCords[i],'ro')
  plt.text(XCords[i],YCords[i],i+1)
plt.show()
```

The output for the above program is as follows:

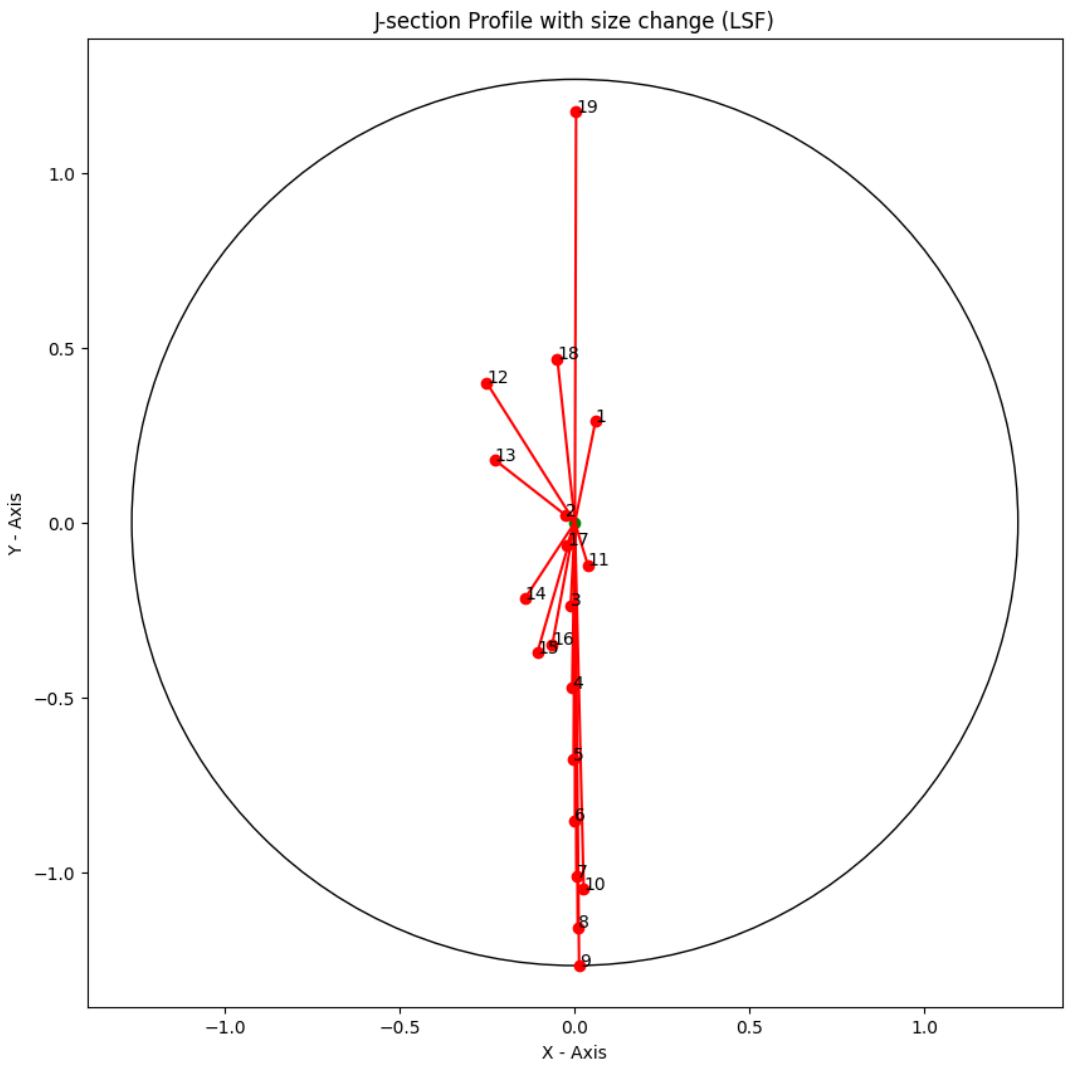


Fig E.1. Least Squares Zone Boundary for 19 Points on the J-shaped Inner Profile with Size Change When Unwrapped onto a Straight Line.

deltatheta_x = 0.0034782233100175055, deltatheta_y = -0.053106244593978745, deltatheta_z = -0.115176810248895, delta_x = 0.6412404097483266, delta_y = 0.13925630663394217, delta_z = -10.152524761857135, delta_r = 7.977072615073124

Phi = 0.1268781654634863, Pitch(h) = 72.31739565809166, s = 9.175498492195759

L = 0.027413883998965036, M = -0.4185609430904166, N = -0.9077748707049301

P = 175.98311668131896, Q = 1797.1850619249476, R = -823.3397043701374

The zone magnitude (radius) is 1.2668170345224004.

F. TRUE MINIMUM ZONE (Prashant Mohan's algorithm [6])

To find the True Minimum Zone of the Least square points there is an algorithm developed by Mohan, *et al*. [6] in the way this algorithm works is it creates two hexagons at each ends of the LSF line and gives the combinations of the hexagons which forms 6x6 which is 36 lines and the hexagon radius is 10% of the original True minimum zone then find distances from points to each of the lines formed during combination and find the maximum distance among all the distances to points from lines which gives us 36 distances of these maximum distances the minimum distance is chosen and the True minimum zone axis is transformed to this line which is transformed to be parallel to z-axis and the iteration continues with a change of 10/11 in radius size and until the radius is greater than the 0.001% of the original zone size we have started with. The way it is implemented in the program is that a lot of function are created which deals with each part of the iterations. The functions of the created functions are to find the distance between two points, to find the distance from a point to line in 3D space, to find the rotation matrix with transforms the true minimum zone axis parallel to the z-axis, to rotate the points according to the rotation of the True minimum zone axis with the rotation matrix generated in the previous case, to generate hexagons with distance and to find the distance from lines generated by the combinations of the hexagons to the LSF points. The program that implements this finding of True minimum zone is as follows it is done for the results from J-section profile with size change (it can also be done for any of the LSF points obtained).

```
import numpy as np
import matplotlib.pyplot as plt

PointsFromLeastSquares=np.array([[ 5.89415668e-02,  2.89197559e-01,  5.10576681e+02],
       [-2.60952053e-02,  2.00869488e-02,  4.61406755e+02],
       [-1.19235933e-02, -2.37873825e-01,  4.11406736e+02],
       [-7.79618197e-03, -4.73854745e-01,  3.61406725e+02],
       [-3.73309840e-03, -6.77835730e-01,  3.11406712e+02],
       [ 2.78946114e-04, -8.51816159e-01,  2.61416698e+02],
       [ 4.24632115e-03, -1.01279785e+00,  2.11406683e+02],
       [ 8.20006867e-03, -1.15777834e+00,  1.61416666e+02],
       [ 1.20721793e-02, -1.26675951e+00,  1.11416648e+02],
       [ 2.52961849e-02, -1.04872063e+00,  6.14266005e+01],
       [ 3.70893686e-02, -1.21683556e-01,  1.14305109e+01],
       [-2.52306885e-01,  3.99620881e-01, -3.73755187e+01],
       [-2.29489170e-01,  1.79179095e-01, -8.06975259e+01],
       [-1.43104565e-01, -2.18149905e-01, -1.44034761e+02],
       [-1.06954765e-01, -3.71020007e-01, -1.92469624e+02],
       [-6.44077268e-02, -3.48491481e-01, -2.30093506e+02],
       [-2.16764430e-02, -6.46147015e-02, -2.83417904e+02],
       [-5.00976501e-02,  4.67259818e-01, -3.30249003e+02],
       [ 3.45346728e-03,  1.17412341e+00, -3.74945502e+02]])
lengthtotal=(PointsFromLeastSquares[-1,2]-PointsFromLeastSquares[0,2])
itrsize=len(PointsFromLeastSquares)
### Function to find the distance between two points a,b
def dbtp(a,b):
      return np.sqrt(np.square(a[0]-b[0])+np.square(a[1]-b[1])+np.square(a[2]-b[2]))
### Function to find the distance to point x from line formed by a,b
def distpttl(a,b,x):
  l=b[0]-a[0]
  m=b[1]-a[1]
  n=b[2]-a[2]
  t=((l*x[0]+m*x[1]+n*x[2])-(l*a[0]+m*a[1]+n*a[2]))/(np.square(l)+np.square(m)+np.square(n))
  nc=[(t*l+a[0]),(t*m+a[1]),(t*n+a[2])]
  return dbtp(nc,x)
### Function to generate matrix that rotate the least squares line to be parallel to the z-axis
def rotmat(initialpoint,finalpoint):
  number=finalpoint-initialpoint
  x=number[0]
  y=number[1]
  z=number[2]
  a=np.arctan(y/z)
  b=np.arctan(-x/((y*np.sin(a)+(z*np.cos(a)))))
  trmat=np.array([[np.cos(b),      np.sin(a)*np.sin(b),      np.sin(b)*np.cos(a)],[0,np.cos(a),-np.sin(a)],[-
np.sin(b),np.sin(a)*np.cos(b),np.cos(a)*np.cos(b)]])
  return trmat
### Function to rotate points with the least squares line
def rotpts(temcords,PointsFromLeastSquares,Q):
  return (np.transpose(np.matmul(Q,np.transpose(PointsFromLeastSquares))))
### Function to create a hexagon with a distance d from the origin
def hexagon(d):
  return                          np.array([[0,d,0],[d*((np.sqrt(3))/2),d/2,0],[d*((np.sqrt(3))/2),-d/2,0],[0,-d,0],[-
d*((np.sqrt(3))/2),-d/2,0],[-d*((np.sqrt(3))/2),d/2,0]])
### Function to create two different hexagons of distance d at ends of the given points 'temcords' and
find the maximum distances to points from each line that joins the coordinates of the hexagons
def disthex(temcords,d,rotatedpoints):
  hex1pts=hexagon(d)+temcords[0]
  hex2pts=hexagon(d)+temcords[1]
  distvals=[]
  for i in range (6):
    for j in range (6):
      for k in range (itrsize):
        distvals.append(distpttl(hex1pts[i],hex2pts[j],rotatedpoints[k]))
  return distvals
### Main code
PointsFromLeastSquares[:,2]=PointsFromLeastSquares[:,2]-PointsFromLeastSquares[0,2]                #
Translate coordinates along the z-axis such that all the rotations can be done with respect to the origin
temcords=np.array([[0,0,0],[0,0,PointsFromLeastSquares[itrsize-1,2]]])  # Starting position of the true
minimum zone axis
### To find the zone magnitude to determine number of iterations to do
distance=[]
for i in range (itrsize):
   distance.append(distpttl(temcords[0],temcords[1],PointsFromLeastSquares[i]))
d=max(distance)/10
print(max(distance))
origtransform=[0,0,0]
### Iterations to find the True minimum zone
while d>0.00001*(max(distance)):
  Q=rotmat(temcords[0],temcords[1]) #   Finding the rotation matrix to make the line parallel to z-axis
  latestpoint=np.transpose(np.matmul(Q,np.transpose(temcords[1]))) #     Finding the new point of the
current true minimum zone line which is parallel to the z-axis
  print("The initial points that the least squares line passes through are {}".format(temcords))      #
Prints the starting line taken to find the true minimum zone axis
  temcords[1]=latestpoint   # Changing the true minimum zone line coordinates parallel to Z-axis
  print("The least squares line points after rotation are {}".format(temcords))      # Transformed true
minimum zone line coordinates parallel to z-axis
  rotatedpoints=rotpts(temcords,PointsFromLeastSquares,Q)     # Rotation of Points with the current
true minimum zone axis
```

```
calc=np.reshape(np.array(disthex(temcords,d,rotatedpoints)),(36,itrsize))       # List of distances from lines formed with hexagon to LSF points
  print("The maximum radius with each line of all 36 lines are {}".format(np.amax(calc,axis=1)))    # Prints the maximum distances form each line to Points
  print("The least most value is located at {} and the distance is {}".format(np.argmin(np.amax(calc,axis=1)),min(np.amax(calc,axis=1))))   # Prints the least distance among the maximum distances
newstart=(hexagon(d)+temcords[0])[int(np.argmin(np.amax(calc,axis=1))/6)]          # Changing the Current true minimum zone axis coordinates Starting point
  newend=(hexagon(d)+temcords[1])[np.argmin(np.amax(calc,axis=1))%6]    # Changing the Current true minimum zone axis coordinates Ending point
  temcords=np.array([newstart,newend])  # New true minimum zone axis coordinates
  origtransform += (newend-newstart)
  print("The new coordinates are {}".format(temcords)) # Printing the new true minimum zone axis coordinates
  d=d/1.1   # changed to enclosure radius of the minimum zone circle
  PointsFromLeastSquares=rotatedpoints      # Replacing the Transformed points for the PointsFromLeastSquares to continue the iterations
print(origtransform)
### Plotting of the True minimum zone of the least squares Points in different orientations
center=[temcords[0][0],temcords[0][1]]
plt.figure(figsize=(10, 10))
plt.axis('equal')
for i in range (itrsize):
plt.plot([center[0],rotatedpoints[i,0]],[center[1],rotatedpoints[i,1]],'r')
    plt.plot(rotatedpoints[i,0],rotatedpoints[i,1],'ro')
    plt.text(rotatedpoints[i,0],rotatedpoints[i,1],i+1)
plt.plot(temcords[0][0],temcords[0][1],'go')
plt.title("zone size={}".format(min(np.amax(calc,axis=1))))
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.gca().add_patch(plt.Circle((center[0],center[1]),min(np.amax(calc,axis=1)),fill=False))
plt.show()
plt.figure(figsize=(15,5))
for i in range (itrsize):
    plt.plot(rotatedpoints[i,2],rotatedpoints[i,1],'r')
    # plt.gca().add_patch(plt.Circle((center[0],center[1]),min(np.amax(calc,axis=1)),fill=False))
    plt.plot(rotatedpoints[i,2],rotatedpoints[i,1],'ro')
    plt.text(rotatedpoints[i,2],rotatedpoints[i,1],i+1)
plt.title("zone size={}".format(min(np.amax(calc,axis=1))))
plt.xlabel('z - Axis')
plt.ylabel('Y - Axis')
plt.show()

plt.figure(figsize=(15,5))
for i in range (itrsize):
    plt.plot(rotatedpoints[i,2],rotatedpoints[i,1],'r')
    # plt.gca().add_patch(plt.Circle((center[0],center[1]),min(np.amax(calc,axis=1)),fill=False))
    plt.plot(rotatedpoints[i,2],rotatedpoints[i,0],'ro')
    plt.text(rotatedpoints[i,2],rotatedpoints[i,0],i+1)
plt.title("zone size={}".format(min(np.amax(calc,axis=1))))
plt.xlabel('z - Axis')
plt.ylabel('x - Axis')
plt.show()
```

The output of the above program is as follows (only for the last iteration):

```
The new coordinates are [[-5.31982027e-02 -6.84447028e-01  0.00000000e+00]

 [-5.31982027e-02 -6.84461833e-01 -8.85521901e+02]]

The initial points that the least squares line passes through are [[-5.31982027e-02 -
6.84447028e-01  0.00000000e+00]

 [-5.31982027e-02 -6.84461833e-01 -8.85521901e+02]]

The least squares line points after rotation are [[-5.31982027e-02  -6.84447028e-01
0.00000000e+00]

 [-5.31982027e-02 -6.84447028e-01 -8.85521901e+02]]

The maximum radius with each line of all 36 lines are [0.98010516 0.98010213
0.98009738 0.98010303 0.98009606 0.98010213

 0.98010147 0.98009843 0.98009738 0.98010303 0.98009528 0.98009843

 0.98009407 0.98009104 0.98009738 0.98010303 0.98009528 0.98009104

 0.9800944  0.9800944  0.98009738 0.98010303 0.98009528 0.9800944

 0.98009407 0.98009104 0.98009738 0.98010303 0.98009528 0.98009104

 0.98010147 0.98009843 0.98009738 0.98010303 0.98009528 0.98009843]

The least most value is located at 29 and the distance is 0.9800910392132668

The new coordinates are [[-5.32098587e-02, -6.84453758e-01,  0.00000000e+00]

 [-5.32098587e-02, -6.84440299e-01, -8.85521901e+02]]
```

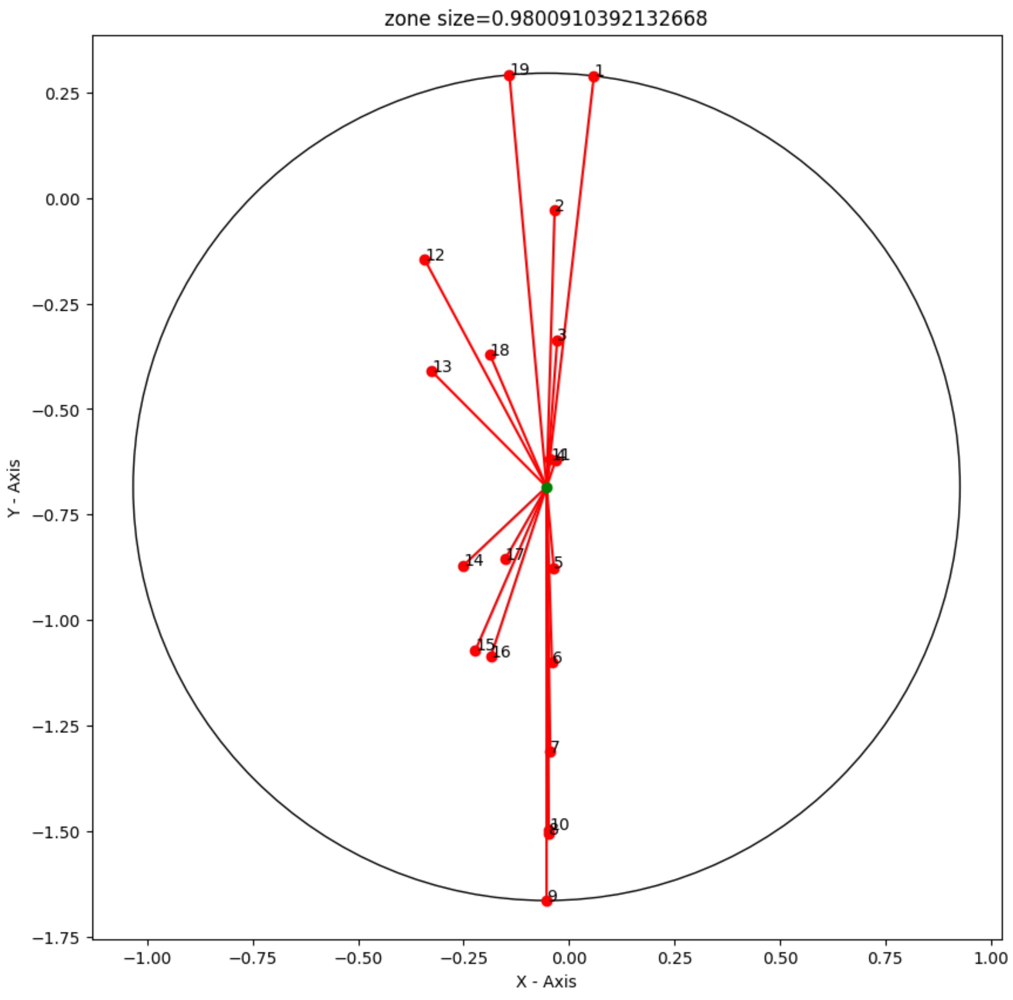


Fig F.1. XY-view of the True Minimum Zone of 19-point Profile with Size Change.

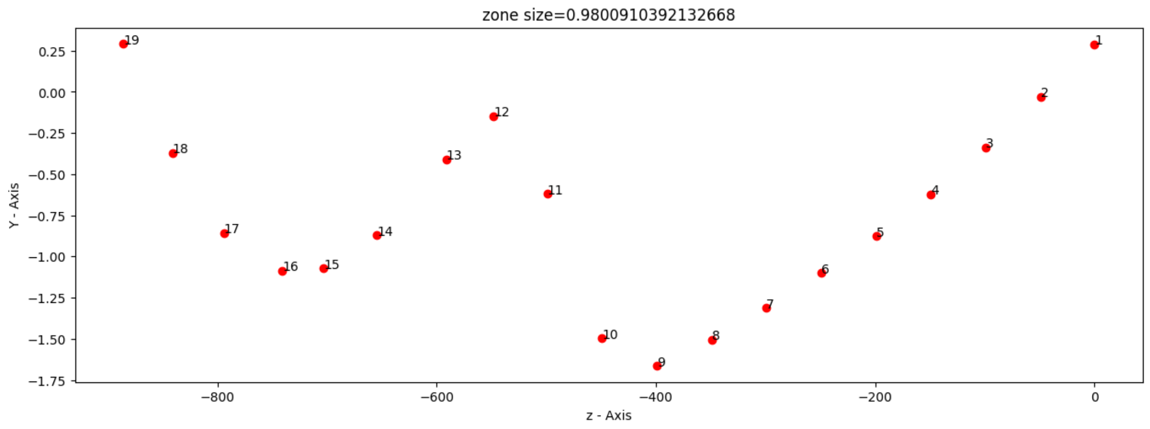


Fig F.2. YZ-view of the True Minimum Zone of 19-point Profile with Size Change.

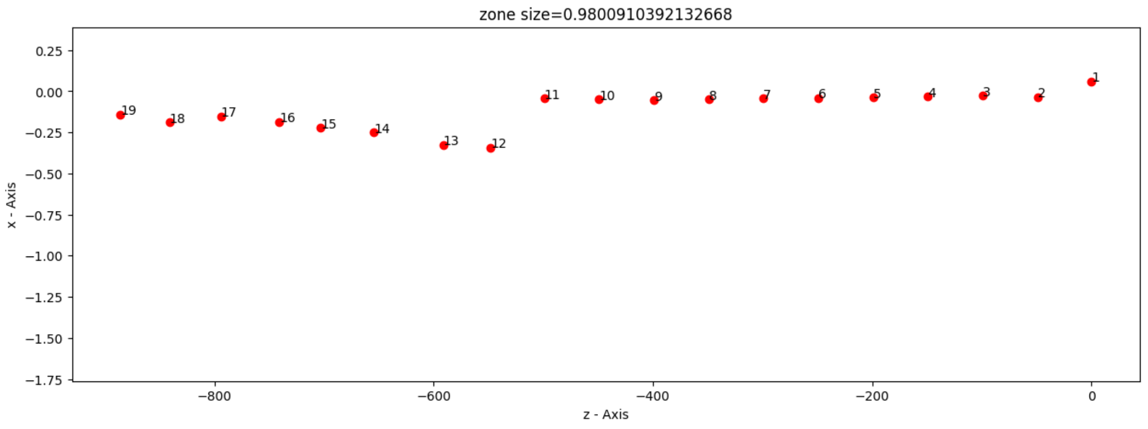


Fig F.3. XZ-view of the True Minimum Zone of 19-point Profile with Size Change.

G. PARAMETERS FROM STAMPED COMPONENTS.

The code developed can extract the Numi-Sheet parameters [4] from the list of stamped components available of different depths and takes the parameters from each sheet that contains the front, center and back profile coordinates of the stamped component and could save all the parameters to an excel file for easy reference. The code is as follows

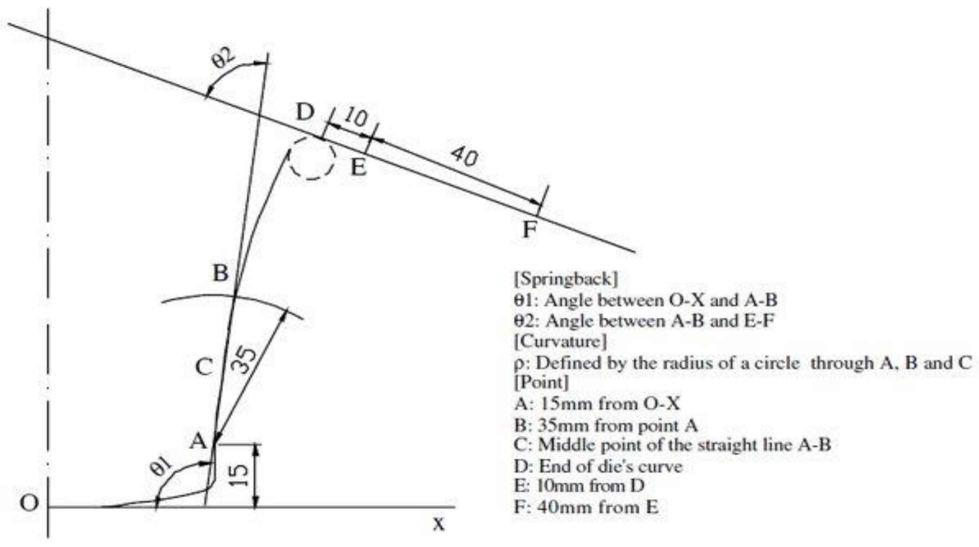

θ2
D
10
40
E
F
B
C
35
A
θ1
15
O
X
[Springback]
θ1: Angle between O-X and A-B
θ2: Angle between A-B and E-F
[Curvature]
ρ: Defined by the radius of a circle through A, B and C
[Point]
A: 15mm from O-X
B: 35mm from point A
C: Middle point of the straight line A-B
D: End of die's curve
E: 10mm from D
F: 40mm from E

Fig G.1. Parameter to Be Extracted from the Cross-sectional Profile of Stamped Component.

```
# NSF GOALI Project File Naming Convention
# Last Updated 3/8/2022

# -----------------------------------------------------------------------------------------
# Component-Level Results: [Shape]_[Channel Width]_[Material]_[Thickness]_[Draw
Depth]_[Blank Holding Force]
#   - Ex: S_50_80DP590_120_55_25700.xlsx

# Assembly-Level Results: [T]_[Top Component Code]-[B]_[Bottom Component Code]-[Weld
Count]_[Weld Pattern]
#   - Ex: T_S_50_80DP590_120_55_25700-B_S_50_100DP590_80_70_25700-3_L.xlsx
#   (Edge # within assembly does not go into filename - denotes Excel sheet; see below)
# -----------------------------------------------------------------------------------------
# Shapes:
#   S - Straight hat section
#   C - Curved hat section
#   T - Tapered hat section

# Channel Width:
#   val[mm]
#       --> 50mm => 50

# Material:
#   [100*MISO scale factor][Nominal]
#       --> 0.8 scale factor for nominal material DP590 Steel => 80DP590
#       --> 1.2 scale factor for nominal material DP590 Steel => 120DP590

# Thicknesses:
#   t[mm]*100
#        --> 1.2mm => 120
#            0.80mm => 80

# Draw Depth:
#   val[mm]
#       --> 55mm => 55
#       --> 0mm => 00 [Flat sheet]

# Blank Holding Force:
#   val[N]
#       --> 25700N => 25700
#       (Flat sheet will always be 00)

# Weld Count:
#   Integer number of welds per flange
#       --> 3 welds per flange => 3
# Weld Pattern:
#   L - Linear; welds in straight line along each flange

# Edge (Excel Sheet #):
#   Sheet1 - Upr Front Profile (rename "Upr Front")
#   Sheet2 - Lwr Front Profile (rename "Lwr Front")
#   Sheet3 - Upr Back Profile (rename "Upr Back")
#   Sheet4 - Lwr Back Profile (rename "Lwr Back")
#   Sheet5 - Upr Left Flange (rename "Upr L")
#   Sheet6 - Lwr Left Flange (rename "Lwr L")
#   Sheet7 - Upr Right Flange (rename "Upr R")
#   Sheet8 - Lwr Right Flange (rename "Lwr R")

import os
import pandas as pd
import numpy as np
from openpyxl import Workbook
import matplotlib.pyplot as plt
os.chdir("e:/Downloads/OSU research/Components/Profiles")   # Changes the directiory to the current
folder
arr=os.listdir()    # Gets list of all the files in the directory
### Function to find the distance between two points
def dist(a,b):
    return np.sqrt((a[0]-b[0])**2+(a[1]-b[1])**2)
### Function to find the roots of a quadratic equation
def quadeq(a,b,c):
    ans=[]
    dis = (b**2)-(4*a*c)
    ans.append((-b-np.sqrt(dis))/(2 * a))
    ans.append((-b+np.sqrt(dis))/(2 * a))
    return ans
### Function to find the required parameters from the stamped component
def distancesfordepths(Adist,Bdist,Edist,Fdist,source,sheetname):
  excel=pd.read_excel(source,sheet_name=sheetname)  # To read the xlsx file in the folder
  reqcol=excel.columns.get_indexer(['Xf (mm)','Yf (mm)'])     # To find the required columns in the
excel sheet
d=pd.read_excel(source,sheet_name=sheetname,usecols=reqcol,names=[0,1],header=None,skiprows=
[0])  # Read only the required columns from the excel sheet
  d=d.dropna()  # To remove any missing values from the columns
  change=d[d[1]==d[1].min()].head(1).index.tolist()[0]    # Finding the lowest point in the stamped
profile
  changeval=d.loc[change][1]    # Finding the location of the lowest point
  d[1]=d[1]-changeval   # Transform the points so that the minimum value will be on the x-axis
### Assumption if excel contains only right side values or the values are in reversed
  if(d.loc[0][0]>50):
    old = pd.DataFrame()
    for i in range (len(d)-1,-1,-1):
      old.at[len(d)-1-i,0]=d.loc[i][0]
      old.at[len(d)-1-i,1]=d.loc[i][1]
    d=old
  par=d[(d[1]>=Adist) & (d[0]>0.0)].head(1).index.tolist()[0]     # To find the point above the line
y=15mm
  a1=d.loc[par-1]   # Location of point below y=15mm line
  a2=d.loc[par]   # Location of point above y=15mm line
  Ax=a1[0]+(Adist-a1[1])*((a2[0]-a1[0])/(a2[1]-a1[1]))   # Interpolating of points to find the optimal
value of point A
  A=[Ax,Adist]  # Assignment of the required point
  tempb=d[(d[1]>=A[1]+(Bdist/2)) & (d[0]>0.0)].head(1).index.tolist()[0]     # Finding the point at a
distance between A and B
  # Iterating through each point from the point choosen above to find the points for B to interpolate
  while (dist(A,d.loc[tempb])<Bdist):
    tempb=tempb+1
  b1=d.loc[tempb-1]    # Location of point around the distance AB towards A
  b2=d.loc[tempb]   # Location of point around the distance AB away from A
  # Interpolation
  n=((b2[0]-b1[0])/(b2[1]-b1[1]))   # Creating quadratic equation for interpolation
  q=(b1[0])-(n*(b1[1]))
  a=n**2+1
  b=2*((n*(q-Ax))-(A[1]))
  c=(q-Ax)**2+(A[1]**2)+-(Bdist**2)
  By=max(quadeq(a,b,c))
  Bx=(b1[0]+((b2[0]-b1[0])/(b2[1]-b1[1]))*(By-b1[1]))   # To check for the sensitivity of the rho value
we can add +0.1 implement perturbation
  B=[Bx,By]    # Assignment of required point B
  tempc=d[(d[1]>=A[1]) & (d[0]>0.0)].head(1).index.tolist()[0]  # Starting to find the points around C
to interpolate and find the point c
  # Finding the points around the equidistant point form A and B
  while (dist(B,d.loc[tempc])-dist(A,d.loc[tempc])>0):
    tempc=tempc+1
  c1=d.loc[tempc]   # Point equidistant from A and B close to B
  c2=d.loc[tempc-1]    # Point equidistant from A and B close to A
  # Finding the optimal point (perpendicular bisector cutting through between the points c1 and c2)
  a1=B[0]-A[0]
  b1=B[1]-A[1]
  ct=(B[0]**2+B[1]**2-A[0]**2-A[1]**2)/2
  d1=A[0]*B[1]-A[1]*B[0]
  cy=(b1*ct-a1*d1)/(a1**2+b1**2)
  cx=(a1*ct+b1*d1)/(a1**2+b1**2)
  cm=-(a1/b1)
  mc=((c2[1]-c1[1])/(c2[0]-c1[0]))
  tempcx=(c1[1]+cm*cx-cy-mc*c1[0])/(cm-mc)
```

```
 tempcy=cy+cm*(tempcx-cx)
 C=[tempcx,tempcy]     # Assignment of required point C
 ### Finding the rho value from the points A, B and C
 tempd=(C[0]-B[0])*(A[1]-C[1])+(B[1]-C[1])*(A[0]-C[0])
 ra=(C[0]**2+C[1]**2-B[0]**2-B[1]**2)/2
 rb=(A[0]**2+A[1]**2-C[0]**2-C[1]**2)/2
 rx=(ra*(A[1]-C[1])-rb*(C[1]-B[1]))/tempd
 ry=(rb*(C[0]-B[0])-ra*(A[0]-C[0]))/tempd
 coc=[rx,ry]
 ### Determine the direction of center of curvature
 if rx>0:
   direction="Right"
 else:
   direction="Left"
 rho=dist(coc,C)
 ### Selecting the top point along the positive x axis as the point D
 posd=d[d[0]>0.0]
 maxpt=posd[posd[1]==posd[1].max()].head(1).index.tolist()[0]
 mxpt=d.loc[maxpt]
 D=[d.loc[maxpt][0],d.loc[maxpt][1]]   # Assignment of required point D
 dieend=D
 ### Finding the point E starting from the point D
 tempe=maxpt
 while(dist(dieend,d.loc[tempe])<Edist):    # Works till the distance DE is the required distance from
D
   tempe+=1
 e1=d.loc[tempe-1]     # Location of point just before the required distance DE form D
 e2=d.loc[tempe]   # Location of point just after the required distance DE form D
 # Interpolation
 en=((e2[0]-e1[0])/(e2[1]-e1[1]))
 eq=(e1[0])-(en*(e1[1]))
 ea=en**2+1
 eb=2*((en*(eq-dieend[0]))-(dieend[1]))
 ec=(eq-dieend[0])**2+(dieend[1]**2)+-(Edist**2)      # for change in distance DE
 Ey=min(quadeq(ea,eb,ec))
 Ex=e1[0]+((e2[0]-e1[0])/(e2[1]-e1[1]))*(Ey-e1[1])
 E=[Ex,Ey]     # Assignment of required point E
 ### Finding the point F starting from the point E
 tempf=d[d[0]>E[0]+(Fdist/2)].head(1).index.tolist()[0]
 while(dist(E,d.loc[tempf])<Fdist):    # Works till the distance EF is the required distance form E
   tempf+=1
 f1=d.loc[tempf-1]     # Location of point just before the required distance EF form E
 f2=d.loc[tempf]   # Location of point just after the required distance EF form E
 # Interpolation
 fn=((f2[0]-f1[0])/(f2[1]-f1[1]))
 fq=(f1[0])-(fn*(f1[1]))
```

```
  fa=fn**2+1
  fb=2*((fn*(fq-E[0]))-(E[1]))
  fc=(fq-E[0])**2+(E[1]**2)+-(Fdist**2)      # for change in distance EF
  Fy=min(quadeq(fa,fb,fc))
  Fx=f1[0]+((f2[0]-f1[0])/(f2[1]-f1[1]))*(Fy-f1[1])
  F=[Fx,Fy]     # Assignment of required point F
  ### Finding of the angle theta1 using the point A and B
  angleAB=np.arctan2((B[1]-A[1]),(B[0]-A[0]))
  theta1=np.arctan2((B[1]-A[1]),(A[0]-B[0]))
  t1=(theta1*(180/np.pi))
  ### Finding of the angle theta2 using the point E and F
  angleEF=np.arctan2((E[1]-F[1]),(E[0]-F[0]))
  theta2=angleEF-angleAB
  t2=(theta2*(180/np.pi))
  [A[1],B[1],C[1],D[1],E[1],F[1]]=[A[1],B[1],C[1],D[1],E[1],F[1]]+changeval
  ### Printing the results for reference
  print("\n\nThe points are (original points not with x-Axis as datum) \nA={} \nB={} \nC={} \nD={}
\nE={} \nF={}".format(A,B,C,D,E,F))
  print("The values of \u03F41, \u03F42 and \u03C1 are {}\u00b0, {}\u00b0 and
{}mm".format(t1,t2,rho))
  ### Plotting graph of the profile with point A, B, C, D, E and F
  mx=[A[0],B[0],C[0],D[0],E[0],F[0]]
  my=[A[1],B[1],C[1],D[1],E[1],F[1]]-changeval
  plt.rcParams['figure.dpi'] = 150
  plt.scatter(d[0],d[1],3)
  plt.plot(mx,my,'ro')
  plt.grid()
  plt.show()
  return [A,B,C,D,E,F,t1,t2,rho,direction]  # Returns the required values accordingly
### Main program
output = pd.DataFrame(columns=["FileName","sheetname","Point A","Point B","Point C","Point
D","Point E","Point F","Theta 1 (deg)","Theta 2 (deg)","Radius of Curvature (mm)","Direction of
curvature"])     # Creates a new pandas dataframe to store the results
for i in arr:   # Loop through all the files
    source=i    # Assigning the file name to source
    sheets=["front","center","back"]    # Sheets in each excel file
    string = source     # To take data from the file name to detrmine conditions to take
    string=string.replace(".xlsx","")   # Removes the .xlsx extenision from the string
    req=string.split("_")   # Splits the string to read each parameter and take appropriate conditions
    distdict = {
    ### "format": [A from base, B from A, E from D, F from E]
    "35": [10,15,15,40],
    "45": [10,25,15,40],
    "55": [15,35,15,40],
    "70": [10,50,15,40]
    }   # This is a dictionary of suitable conditions to choose based on the Draw Depth of the stamping
    rds=(distdict[req[4]])
    for sheetname in sheets:    # Looping through each sheet of the file
        print("{0}, {1}, {2}mm".format(source,sheetname,req[4]))      # Prints the filename, sheetname and Drawdepth before print out the results.
        [A,B,C,D,E,F,t1,t2,rho,direction]=distancesfordepths(rds[0],rds[1],rds[2],rds[3],source,sheetname)      # Reading outputs form the function to append to the pandas dataframe
        output.loc[output.shape[0]]=[source,sheetname,A,B,C,D,E,F,t1,t2,rho,direction]    # Appending the data to pandas dataframe
output.to_excel('../Outputswithpandas.xlsx', header=True, index=False)  # Writing the outputs to an excel file
```

The output for the above code is as follows (reduced to one part of the result):

```
S_50_100DP590_100_35_15500.xlsx, front, 35mm

The points are (original points not with x-Axis as datum)

A = [26.0244333976834, -16.288]

B = [27.234399772474156, -1.336880263609828]

C = [26.615856122161862, -8.811342708705812]
```

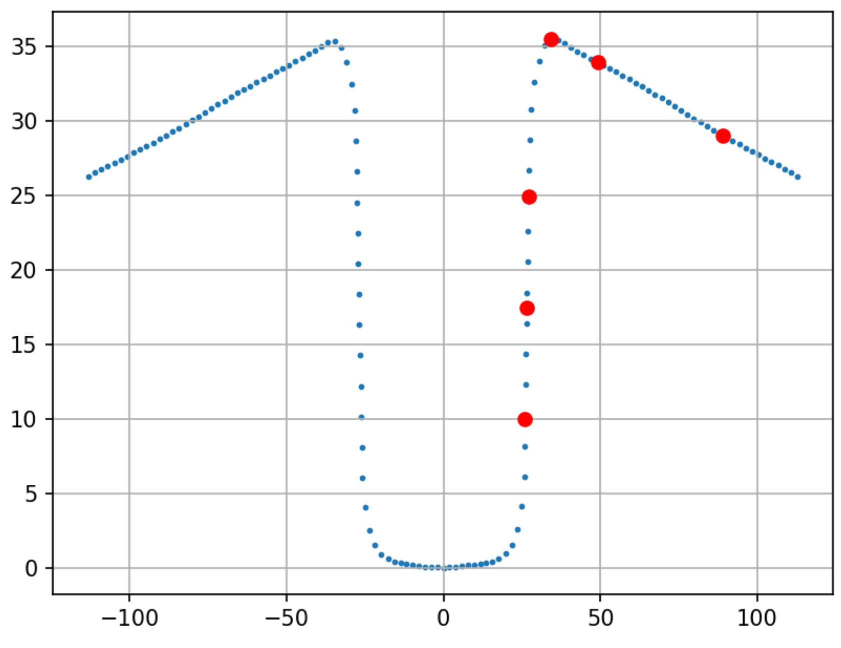


Fig G.2 The Points Obtained from the Program.

D = [34.379, 9.249300000000002]

E = [49.29227253039767, 7.638616159491729]

F = [88.9923534229796, 2.7494876970301156]

The values of ϴ1, ϴ2 and ρ are 94.62675791558097°, 87.60604141025006° and 2067.2923867836153mm

The output sheet has been modified a little for further ease of reference.

Table G.1(a). Sample list of the parameters generated in an excel document for stamped components.

File Name	Shape	Channel Width	Material	Thickness	Draw Depth	Blank Holding Force	Sheet Name	Point A	Point B
S_50_100DP590_100_35_15500.xlsx	S	50	100DP590	100	35	15500	front	[26.0244333976834, -16.288]	[27.234399772474156, -1.336880263609828]
S_50_100DP590_100_35_15500.xlsx	S	50	100DP590	100	35	15500	back	[26.055245603864734, -16.292]	[27.265645962747058, -1.3409153914768623]
S_50_100DP590_100_35_15500.xlsx	S	50	100DP590	100	35	15500	center	[25.84421334622824, -16.18]	[27.146128559538937, -1.2366062496717234]
S_50_100DP590_100_35_25700.xlsx	S	50	100DP590	100	35	25700	front	[25.951078716216216, -18.704]	[26.858764322903117, -3.731488292226537]
S_50_100DP590_100_35_25700.xlsx	S	50	100DP590	100	35	25700	back	[25.903623444283646, -18.787]	[26.869474130785886, -3.818127883124152]
S_50_100DP590_100_35_25700.xlsx	S	50	100DP590	100	35	25700	center	[25.6455595549105, -18.62]	[26.657955312819734, -3.6542038357671274]

Table G.1(b). Sample list of the parameters generated in an excel document for stamped components.

Point C	Point D	Point E	Point F	Theta1 (deg)	Theta2 (deg)	Radius of Curvature (mm)	Direction of center
[26.615856122161862, -8.811342708705812]	[34.379, 9.249300000000002]	[49.29227253039767, 7.638616159491729]	[88.9923534229796, 2.7494876970301156]	94.62675792	87.606041	2067.292387	Right
[26.65332596032008, -8.815881293656386]	[34.379, 9.263300000000001]	[49.28536231363366, 7.589871610528853]	[89.00089548910036, 2.8278946483411325]	94.62842103	87.791175 71	3937.360399	Right
[26.371364020604567, -8.697516677570903]	[34.398, 9.175599999999996]	[49.31484270889775, 7.598322726335098]	[89.03957677884222, 2.9137199606545856]	94.97921481	88.25358073	226.3730784	Right
[26.447660424158276, -11.220335126792186]	[33.69199999999999, 7.116]	[48.621749750281516, 5.665975036764642]	[88.44138775089519, 1.8717123283916202]	93.46922306	88.0261586	656.881001	Left
[26.41857176658813, -11.30463019051264]	[33.69, 7.160600000000002]	[48.61963104765967, 5.7093533701151244]	[88.43264658936776, 1.8462207713464274]	93.69183197	88.14966361	876.4690185	Left
[26.051810961355883, -11.130340808506162]	[33.71, 7.066599999999999 76]	[48.655766965303116, 5.792217867795554]	[88.47836856421847, 2.02918692265861]	93.87000893	88.47188162	280.8090488	Right

H. FITTING OF EDGE PROFILES OF STAMPED COMPONENTS

This Edge profile is like the 11 points straight line profile least squares fit. This program also finds the Least squares fit line along the edges of the stamped components by taking the inputs from the excel sheets that contain the data about the stamped profile and plot the least square points for all the components available. The code for this is as follows

```
import os
import pandas as pd
import numpy as np
from openpyxl import Workbook
import matplotlib.pyplot as plt
os.chdir("e:/Downloads/OSU research/Components/Profiles")     # To change the current working
directory to the path give
arr = os. listdir()     # List all the files in the folder specified
### Function to print the edge profile of the stamped component
def edges(source,sheetname):
  point= lambda pt : [d.loc[pt][0],d.loc[pt][1],d.loc[pt][2]]    # This returns the point coordiantes at the
point location
  excel=pd.read_excel(source,sheet_name=sheetname)  # Reads the excel sheet in the folder
  reqcol=excel.columns.get_indexer(['Xf (mm)','Yf (mm)','Zf (mm)']) # Finds the columns from which
the data needs to be taken
d=pd.read_excel(source,sheet_name=sheetname,usecols=reqcol,names=[0,1,2],header=None,skiprows
=[0])    # Read the required columns into a pandas dataframe
  d=d.dropna()  # Remove any of the empty rows in the data
  temporary=(np.average(d[0]))   # Finding the average of x-axis to translate the origin to that point
  d[0]=d[0]-(temporary)     # Translate points to the new origin
  print("The offset for the x-coordiantes are {}".format(temporary))
  ## Choosing the points equidistant in the set of points
  req_points=[]
  for i in range (len(d)):
      if i%6==0:
          req_points.append(point(i))
  ### Creating the d and K matrix to find the SDT of the least square matrix
  tempa=np.array([])
  tempb=np.array([])
  origptset=[]
  for i in range (len(req_points)):
    tempa=np.append(tempa,req_points[i][0])
    tempa=np.append(tempa,req_points[i][1])
    tempb=np.append(tempb,np.array([0,req_points[i][2],1,0]),axis=0)
    tempb=np.append(tempb,np.array([-req_points[i][2],0,0,1]),axis=0)
    origptset.append([req_points[i][0],req_points[i][1],req_points[i][2],1])
  di=np.array(tempa)
  kprime=np.hstack(tempb).reshape(int(np.size(tempb)/4),4)
  ### Least squares for the equations and finding the angular orientaion and position
  kprTranspose=np.transpose(kprime)
  intermediate=np.linalg.inv(np.matmul(kprTranspose,kprime))
  kmoorePenrose=np.matmul(intermediate,kprTranspose)
  pdi=np.matmul(kmoorePenrose,di)
  deltatheta_x=pdi[0]*(180/np.pi)
  deltatheta_y=pdi[1]*(180/np.pi)
  delta_x=pdi[2]
  delta_y=pdi[3]
  print("deltatheta_x = {}, deltatheta_y = {}, delta_x = {}, delta_y = 
{}".format(deltatheta_x,deltatheta_y,delta_x,delta_y))
  ### Finding the values of PHI, h(pitch) and s
  phi=np.sqrt((np.power(pdi[0],2))+(np.power(pdi[1],2)))
  h=((pdi[0]*pdi[2])+(pdi[1]*pdi[3]))/((np.power(pdi[0],2))+(np.power(pdi[1],2)))
  s=phi*h
  print("Phi = {}, Pitch(h) = {}, s = {}".format(phi*(180/np.pi), h*(np.pi)/180, s))
  ### Finding the values of PHI, h(pitch) and s
  l=pdi[0]/phi
  m=pdi[1]/phi
  n=0
  p=(delta_x-(h*pdi[0]))/phi
  q=(delta_y-(h*pdi[1]))/phi
  r=0
  print("L = {}, M = {}, N = {}".format(l,m,n))
  print("P = {}, Q = {}, R = {}".format(p,q,r))
  Lt,Mt,Nt,Pt,Qt,Rt=l,m,n,p,q,r
  ### Finding the Nominal X,Y and Z of the Least Squares fit Line
  vphi=1-np.cos(phi)
  x_nom=vphi*(m*r-n*q)+p*np.sin(phi)+l*s
  y_nom=vphi*(n*p-l*r)+q*np.sin(phi)+m*s
  z_nom=vphi*(l*q-m*p)+r*np.sin(phi)+n*s
  print("X_nom = {}, Y_nom = {}, Z_nom = {}".format(x_nom,y_nom,z_nom))
  ### Creating a Matrix to find the transformed points
  trnomls1=np.array([(vphi*np.power(l,2)+np.cos(phi)),                    (vphi*l*m-n*np.sin(phi)), 
(vphi*l*n+m*np.sin(phi)), x_nom])
  trnomls2=np.array([(vphi*l*m+n*np.sin(phi)),    (vphi*np.power(m,2)+np.cos(phi)),    (vphi*m*n-
l*np.sin(phi)), y_nom])
  trnomls3=np.array([(vphi*l*n-m*np.sin(phi)),                          (vphi*n*m+l*np.sin(phi)), 
(vphi*np.power(n,2)+np.cos(phi)), z_nom])
  trnomls4=np.array([0,0,0,1])
  trnomls=np.vstack([trnomls1,trnomls2,trnomls3,trnomls4])
  invtrnomls=np.linalg.inv(trnomls)
```

```
print("The trasformation matrix is \n{}".format(invtrnomls))
### Original points transformed along the Leasts squares line
origptsetCol=np.transpose(origptset)
ptset = np.matmul(invtrnomls,origptsetCol)
newcords=(np.transpose(np.delete(ptset,3,0)))
print("The Transformed Least squares coordinates are \n{}".format(newcords))
### Plotting of the least squares points in different orientaions
x_coordinates=ptset[0]
y_coordinates=ptset[1]
z_coordinates=ptset[2]
### Ploting of the least square point along y axis
plt.figure(figsize=(10, 10))
plt.text(0,-0.00025,"origin")
plt.xlim([-0.01,0.01])
for i in range (len(x_coordinates)):
    plt.plot(x_coordinates[i],z_coordinates[i],'ro')
    plt.text(x_coordinates[i],z_coordinates[i]+5,i+1)
plt.plot(0,0,'go')
plt.title("Component Name: {}, ({} free edge)".format(source.replace(".xlsx",""),sheetname))
plt.xlabel('x - axis')
plt.ylabel('z - axis')
plt.show()
### Ploting of the least squre points along the z axis
plt.figure(figsize=(10, 2))
plt.axis('equal')
plt.ylim([-0.0001,0.0001])
for i in range (len(x_coordinates)):
    x=[0,x_coordinates[i]]
    y=[0,y_coordinates[i]]
    plt.plot(x,y,'r')
    plt.text(x_coordinates[i],y_coordinates[i]+0.0001,i+1)
    plt.plot(x_coordinates[i],y_coordinates[i],'ro')
plt.plot(0,0,'go')
plt.title("Component Name: {}, ({} free edge)".format(source.replace(".xlsx",""),sheetname))
plt.xlabel("X - axis")
plt.ylabel("Y - axis")
plt.show()
lssum=0
for i in range (int((np.size(di))/2)):
    lssum=lssum+np.power((ptset[0,i]),2)+np.power((ptset[1,i]),2)
print("The leastsqures sum is {}".format(lssum))
ptlsdist=[]
for i in range (int((np.size(di))/2)):
    ptls=np.sqrt((np.power(ptset[0,i],2))+(np.power(ptset[1,i],2)))
    ptlsdist.append(ptls)
Ptlstdist=np.hstack(ptlsdist)
```

```
  zonemag=np.max(ptlsdist)
  print("the Zone mag is {}\n\n".format(zonemag))
  return [deltatheta_x,deltatheta_y,delta_x,delta_y,phi,h,s,Lt,Mt,Nt,Pt,Qt,Rt,zonemag]
### Main program
output                                                                                    =
pd.DataFrame(columns=["FileName","sheetname","deltatheta_x","deltatheta_y","delta_x","delta_y","
PHI","pitch (h)","S","L","M","N","P","Q","R","LS zone radius"])  # Creates a pandas dataframe with
the following titles
for i in arr:   # Loop through the files in the folder
  source=i
  sheets=["left","right"]
  for sheetname in sheets:  # Loop through each sheet in the excel file
[deltatheta_x,deltatheta_y,delta_x,delta_y,phi,h,s,Lt,Mt,Nt,Pt,Qt,Rt,zonemag]=edges(source,sheetnam
e)             #          Reading          outputs          from          the          function
output.loc[output.shape[0]]=[source,sheetname,deltatheta_x,deltatheta_y,delta_x,delta_y,phi,h,s,Lt,Mt
,Nt,Pt,Qt,Rt,zonemag]  # writing the values to the pandas dataframe
output.to_excel('../Edgeswithpandas2.xlsx',  header=True,  index=False)  #  Converts  the  pandas
dataframe and saves it in excel format.
```

The output of the above program is (Only one output for example):

```
The offset for the x-coordinates is -113.09076712328768

deltatheta_x = -7.209670798812664e-16, deltatheta_y = -0.004412272070600839,

delta_x = -0.011353787608870425, delta_y = 2.786194537136048e-15

Phi = 0.004412272070600839, Pitch(h) = -2.1099808319849559e-13, s = -

9.309809494470342e-16

L = -1.6340041328935752e-13, M = -1.0, N = 0

P = -147.43517649572803, Q = 2.4090968772791326e-11, R = 0

X_nom = -0.011353787597648463, Y_nom = 2.786194535302374e-15, Z_nom = -

4.371700674230559e-07

The Transformed Least squares coordinates are

[[ 2.12086214e-03 2.58413429e-16 -6.33006157e-04]
[-8.35179200e-04 -1.25168607e-16 -2.54009056e+01]
[-7.91241707e-04 4.00075282e-15 -5.08014531e+01]
[-7.47292901e-04 -2.54686761e-15 -7.62018536e+01]
[ 2.96703778e-04 7.77184140e-16 -1.01601633e+02]
[ 3.40689301e-04 -1.91477229e-15 -1.27001556e+02]
[ 3.84788805e-04 -8.55344337e-16 -1.52400000e+02]
[-5.71371376e-04 -1.24690995e-15 -1.77801816e+02]
[-5.27552707e-04 -2.15951117e-16 -2.03203906e+02]
[-4.83580113e-04 -6.09493430e-16 -2.28603998e+02]
[-4.39571087e-04 4.22862303e-16 -2.54003617e+02]
[-1.39557617e-03 1.58939834e-15 -2.79403418e+02]
[ 2.64832122e-03 4.65896317e-16 -3.04804487e+02]]
```

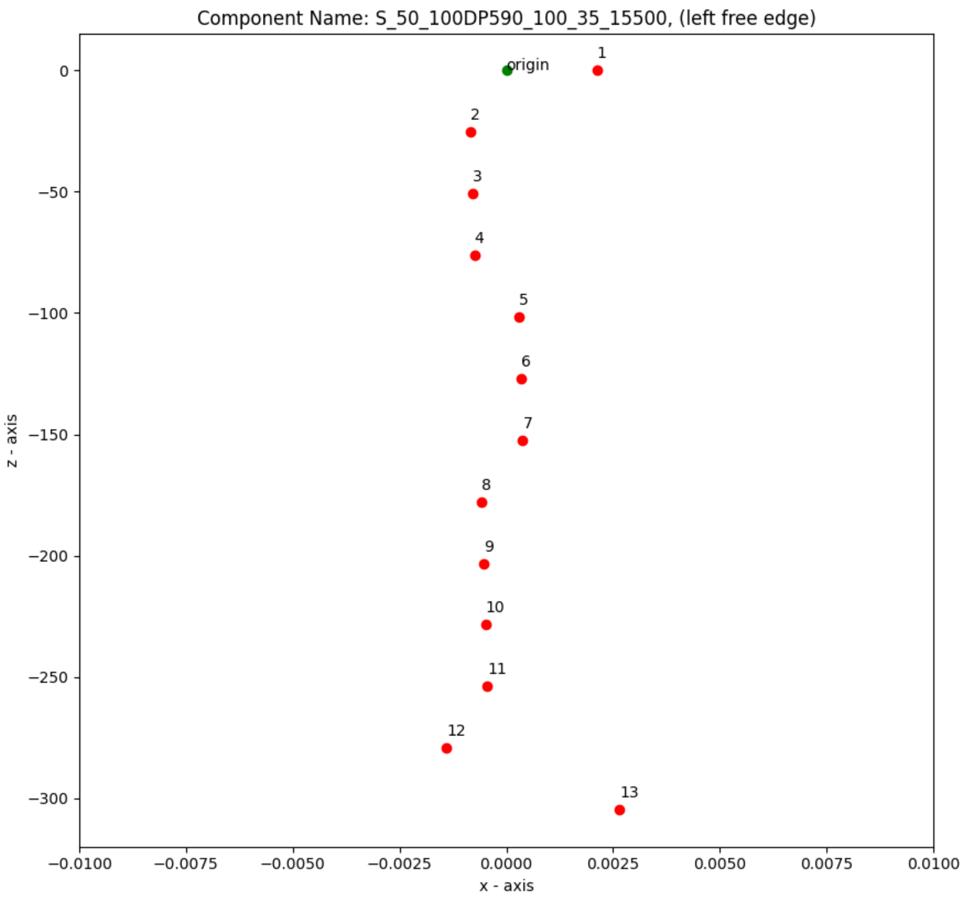


Fig H.1. The XZ-view of the Least Squares Fitted Edge of Stamped Component.

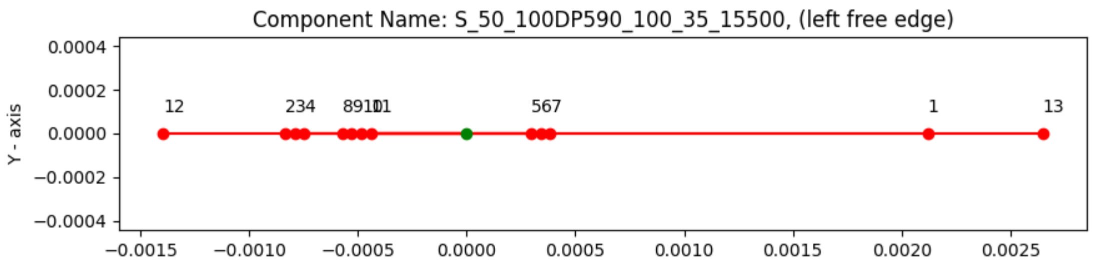


Fig H.2 The XY-view of the Least Squares Fitted Edge of Stamped Component.