Development of an advanced geometry toolkit framework for fitting complex topology-optimized mesh structures

By

Jani Harshit Girishkumar

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

Of the Requirements

For the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

Acknowledgements

I would like to specially thank my research guide Dr. Robert M Talyor for his constant support, continuous guidance, and research opportunity. I would like to thank Dr. Ashfaq Adnan and Dr. Kent Lawrence for taking their valuable time to serve on my thesis committee and their insightful suggestions.

Finally, I would like to acknowledge, Mechanical and Aerospace Engineering Department of University of Texas at Arlington for providing a great opportunity and support through my research work.

Dedication

I would like to dedicate my work to my parents, Girishkumar Jani and Kumud Jani. My thesis would have been impossible without their blessings. I would like to thank my friends for their much-needed motivation. My advisor was my backbone and support system for entire research work. My big thank to my advisor for spending his valuable time and energy on me.

ABSTRACT

Additive Manufacturing is playing a significant role in developing complex geometries which are not possible by conventional Manufacturing Processes. Topology optimization is playing key a role in deciding conceptual design for additive manufacturing. The output of topology optimization is rough and noisy surfaces. Fitting these surfaces poses challenging task to a designer as it is a tedious and a time-consuming process. The main aim of this research is to automate the process of smoothing noisy meshes.

In this research work, I have developed algorithms in MATLAB to create NURBS (Non-Uniform B-Spline) surface patches from given a set of control points. NURBS is a powerful tool in geometric modeling with flexibility. Different types of NURBS Surfaces are discussed along with examples. Each type has its usage. STEP standard has been used for geometry data exchange between MATAB and CAD Software. The algorithm to export NURBS into STEP file has been developed to support data exchange.

TABLE OF CONTENT

TABLE OF FIGURES

1 Introduction

The Procedure, advantages, and methods of optimization and additive manufacturing techniques will be discussed alongside with a problem that after topology optimization. The approach has been presented to tackle the ongoing tedious method after optimization phase.

1.1Background

We are optimizing many things in our daily file knowingly or unknowingly. Finding the fastest route from home to workplace, optimizing time schedule of tasks for better productivity, Varying the car's body shape for reducing drag, in structural optimization, finding the optimal material distribution or setting the thickness of the truss by varying design variables under constraints for example keeping design's von Mises stress under yield stress. First, an objective function is defined in the optimization problem, and often the aim is to minimize the thickness, reduce the overall weight, maximization of stiffness, etc. The following figure illustrates the general procedure in optimization.

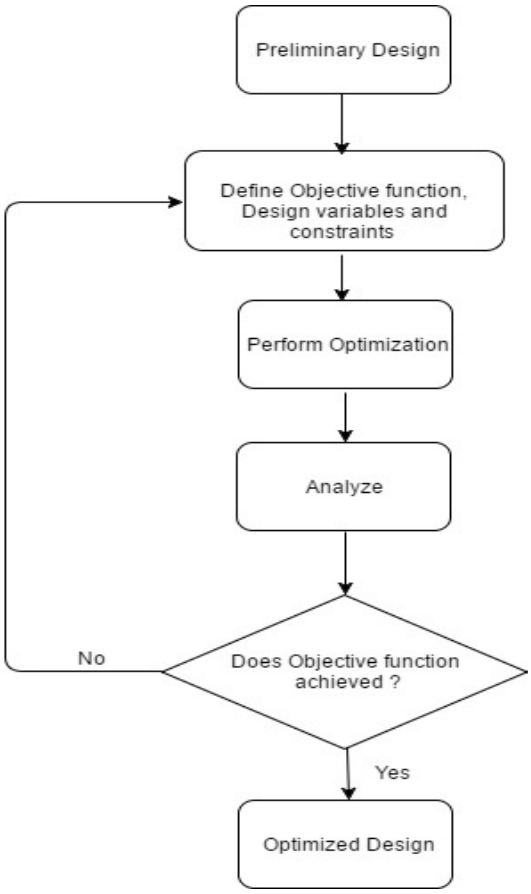


Figure 1 Optimization Process flowchart

The steps within the optimization process are 1) Define design and non-design spaces 2) Define an Objective Function, Design variables, and constraints 3) Perform optimization 4) Analyze the design and check whether objective function is met or not. Repeat the process until the objective function is not satisfied. According to the design variables and objective function, there are three main types of optimization.

1) Topology Optimization
2) Shape Optimization
3) Size Optimization

$$\underset{X}{Min}\ F(X) \tag{1}$$

$$S.T.\ G(X)\ \leq 0$$

$$X_{Min} \leq X \leq X_{Max}$$

Here, the objective function is to minimize the function F(X), X is a design variable, G(X) is constraints and bounds Xmin and Xmax represent upper and lower bound for the design variable. [1]

If design variable x is height, thickness, length than the process is size optimization.

If x controls the exterior curve (inner or outer radius), we are considering shape optimization.

If x govern whether a finite element of the geometry is void or solid, it is topology optimization.

1.1.1 Topology Optimization

Topology optimization is carried out in conceptual design stage where a configuration of the product, the number of holes are not known. Design variable for topology optimization is density. The below figure shows the overall topology optimization process.

Topology Optimization Flow Chart

CAD Model is first imported, and design and non-design spaces are decided. Finite element mesh is generated in design space, and loads, boundary condition, design variables, constraints and objective functions are defined. Topology optimization process distribute the material in design space according to constraints and objective function. Therefore, the result is coarse topology which is not suitable for manufacturing. In this type of optimization, the final

geometry is not predicted and can produce complex shapes that are extremely hard to produce with conventional subtractive manufacturing processes. Therefore, manufacturing constraints are applied before optimization process to make sure the resultant part is manufacturable with minimum cost. The coarse topology is smoothened by fitting different cross sections using spline surface tools. [12]

Density based approach

The design variable is density, and the value is either 0% or 100% at the end of the optimization process. The following figure 1.1 shows the A-arm with finite element mesh. The red area shows the design space and green shown the non-design space.

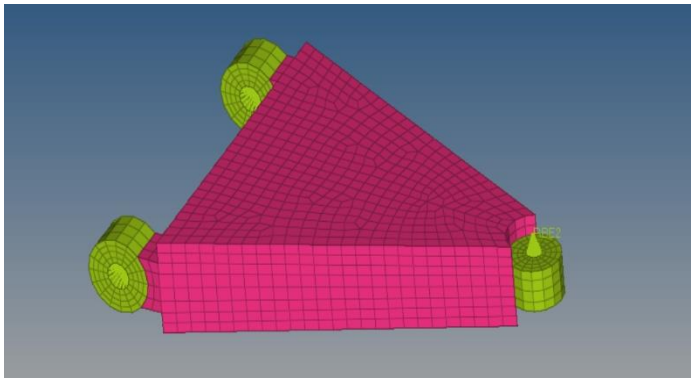


Figure 1.1 A-arm body with constraints

The resultant topology is shown below in figure 1.2. It consists of defects, noises, rough surfaces with complex geometry. Therefore, Designer uses CAE Tools to fit these noisy surfaces and curves using NURBS (Non-uniform rational B-spline surface) by picking cross sections.

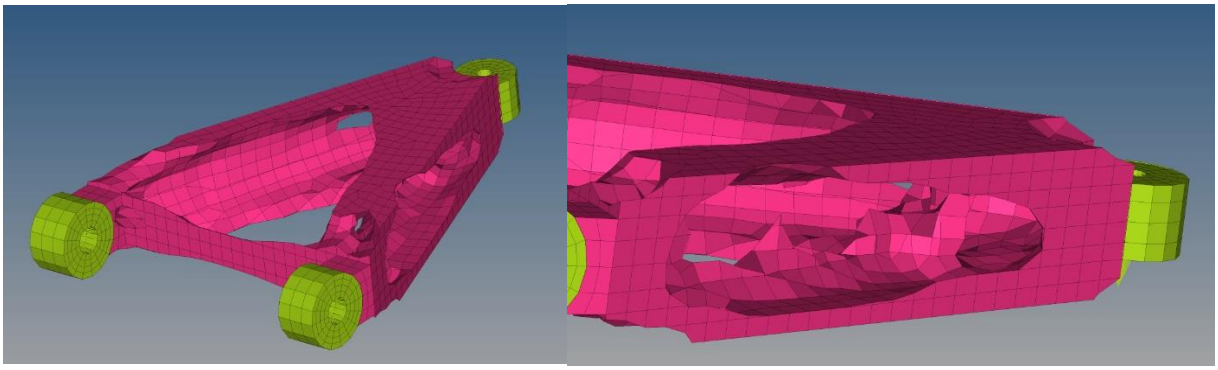
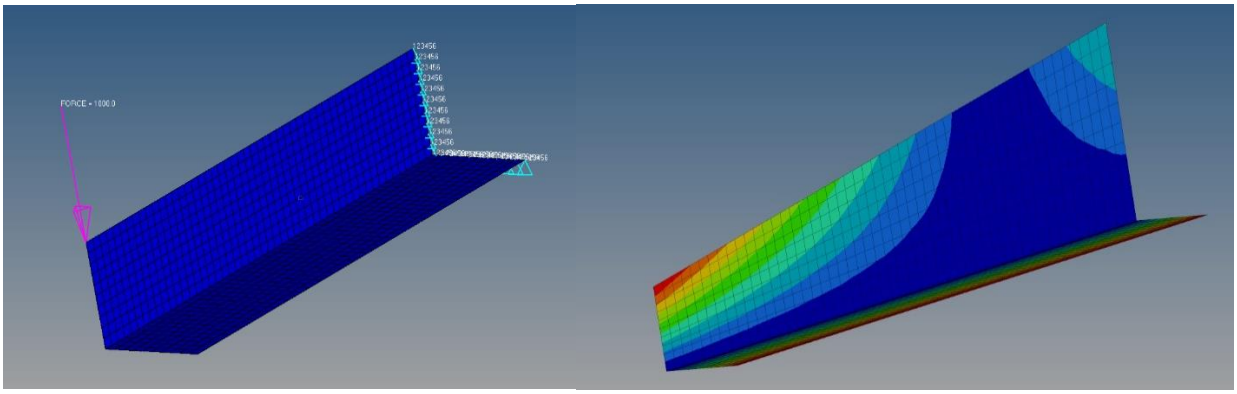


Figure 1.2 Topology Optimized Body of A-Arm

1.1.2 Size and shape optimization

Size optimization is a process of changing the radius or thickness parameter that is not associated with the overall shape of the product over a certain range with the defined objective function. Where, shape optimization uses height, length as a variable to change the shape of

the geometry to obtain objective process. The following figures portray the shape optimization.[1]



A B

Figure 1.3 shape optimization of cantilever beam A) Constraints B) Result

L-section cantilever bar is shown with applied 100N load at one end and constrained at other. The objective function is mass minimization with maximum nodal displacement less than 2mm. The right figure shows the result after shape optimization. The shape has changed, trapezoid shape has resulted from a rectangular shape. Size optimization uses thickness as a design variable. In this case, the thickness can be used for size optimization process.

Shape optimization changes the shape of the geometry, here, the height of each node is a design variable.

1.1.3 Additive Manufacturing

Additive Manufacturing technology uses different approach than subtractive manufacturing. In AM, 3-D Object is built by adding layer upon layer of material. AM requires 3D CAD model as an Input with STL or AMF file format. STL file contains the data that represents 3-D Objects created by CAD Software. [2]

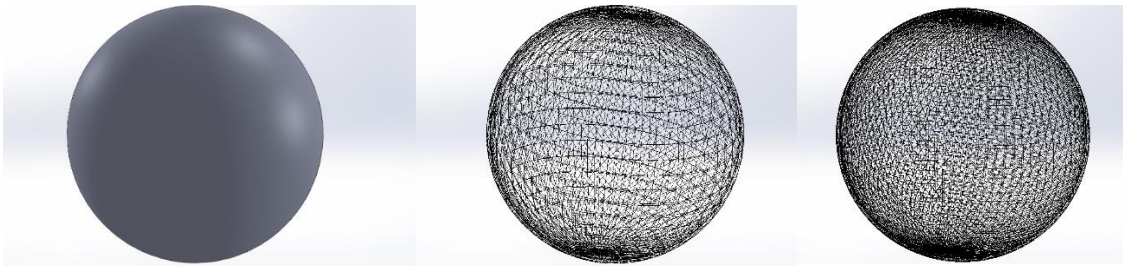


Figure 1.4 3D CAD Model and STL Files of sphere

STL means Stereolithography; it also called “Standard Triangle Language” or “Standard Tessellation Language.” STL file converts 3D Object into series of linked vertex and triangles, and it represents the surface geometry of the solid object. The number of triangles describes

the resolution of the given 3D Model. 3D Slicer in AM Machine cut the 3D Object into slices of predefined thickness as well as defines a path. Figure 1.4 represents the 3D CAD model of sphere with coarse and fine STL File.

The additive manufacturing process involves eight steps. [2]

1) CAD

2) STL Convert

3) File transfer to machine

4) Machine setup

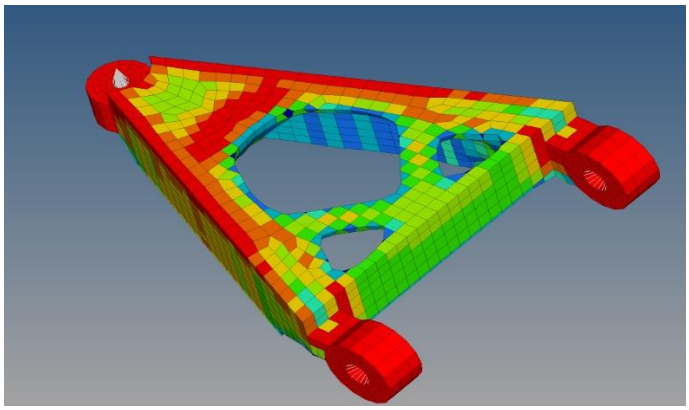
5) Build

6) Remove

7) Post-Process

8) Application

Topology optimization produces unpredictable shape. Additive manufacturing has a capability to produce complicated shapes without adding restrictions. On the other hand, in conventional manufacturing processes, the designer must provide manufacturing constraints before topology optimization to produce a manufacturable part. But in the additive manufacturing, no compromise should be made because of the capability of AM.



1.5 Topology optimization with manufacturing constraints

The Figure 1.5 shows the result manufacturing constraints. Here, symmetry and draw constraints are applied before optimization. There is significant difference in this result from previous result. These constraints affect the design and performance of the product. Other constraints like min member size, min hole size, and axisymmetric constraints are applied for conventional manufacturig processes.

1.2 Objective

After topology optimization, to fit the rough surfaces, the designer must fit the different cross sections to generate a surface. If a geometry is complex, containing many sub-assemblies than fitting cross section by hand would be a tedious and time-consuming process. The Objective of the main research is to automate the process of generating surfaces with various degree of freedom. Several steps are involved in automating the entire process described in given flow chart.

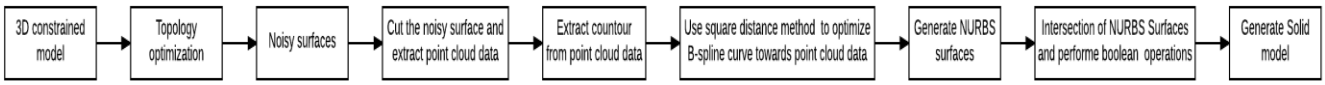


Figure 1.6 Conceptual flow chart for overall research work

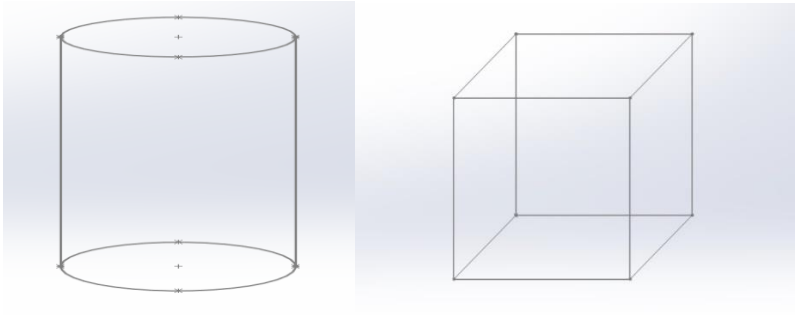
1.3 Method

The main aim of my research is to automate the process of creating NURBS Curves and surfaces from given a set of inputs as well as export the geometry into STEP AP 214 file for data exchange with CAD software. Different types of algorithm have been developed to generate distinct types of NURBS Surfaces.

2 Background work

2.1 What is geometric modeling?

Geometric Modeling is a mathematical way of representing curves, surfaces, and solids for visual representation in computers. Computer computations are required to construct an entity from mathematical definition. In fact, use of computers is a primary importance in Geometric Modeling. Without computation power, one cannot represent complex geometries with accuracy. Geometric Modeling uses linear algebra, vectors, matrix, polynomial interpolation to capture the mathematical definition of a geometric entity. Vector and Matrices are useful in geometric transformations like scaling, rotation, moving, etc. In polynomial interpolation curve passes through a set of control points. The approximation is used in free form curves like Bezier, B-spline curves and surfaces where curves are not passing through control points except first and last points. [3]



A B

Figure 2.1 2D Sketches A) Cylinder B) Cube Sketch

Figure 2.1 shows two objects. The objects can be a 3D solid or 2D sketch but the lines are placed in a way that gives 3D intuition. From visualization, one cannot interpret the result. Both, cube and cylinder are 2-D sketches. The main aim of the Geometric Modeling is to mathematically formulate an entity that is computer compatible.

2.2 Representation method of Curves and surfaces

The table shows three ways to represent a mathematical equation of a line, circle, surface.

Entity	Explicit Representation	Implicit Representation	Parametric Representation
Line	$Y = mx + b$	$Ax + By + C = 0$	$P_1 + t(P_1 - P_2)$
Circle	$Y = (r^2 - X^2)^{1/2}$	$X^2 + Y^2 - r^2 = 0$	$X = X_0 + r\cos(t)$ $Y = Y_0 + rsin(t)$
Surface	$Z = Ax + By + C$	$Ax + By + Cz + d = 0$	$X = a_0 + a_1u + a_2w$

Any geometric object can represent in three forms Explicit, Implicit and Parametric forms. The parametric form has been adapted because of advantages over other forms.

In curve and surface modeling, flexibility in modeling is of upmost important. implicit and explicit expressions are dependent on axis. On the other hand, parametric form provides independent variables, like u, w, and t in given example. The value of independent variables controls the shape of the overall geometry. In programming, it is easy to handle and model entities with parametric equations with independent variable which provides enormous flexibility.

2.2.1 Parametric representation of a circle:

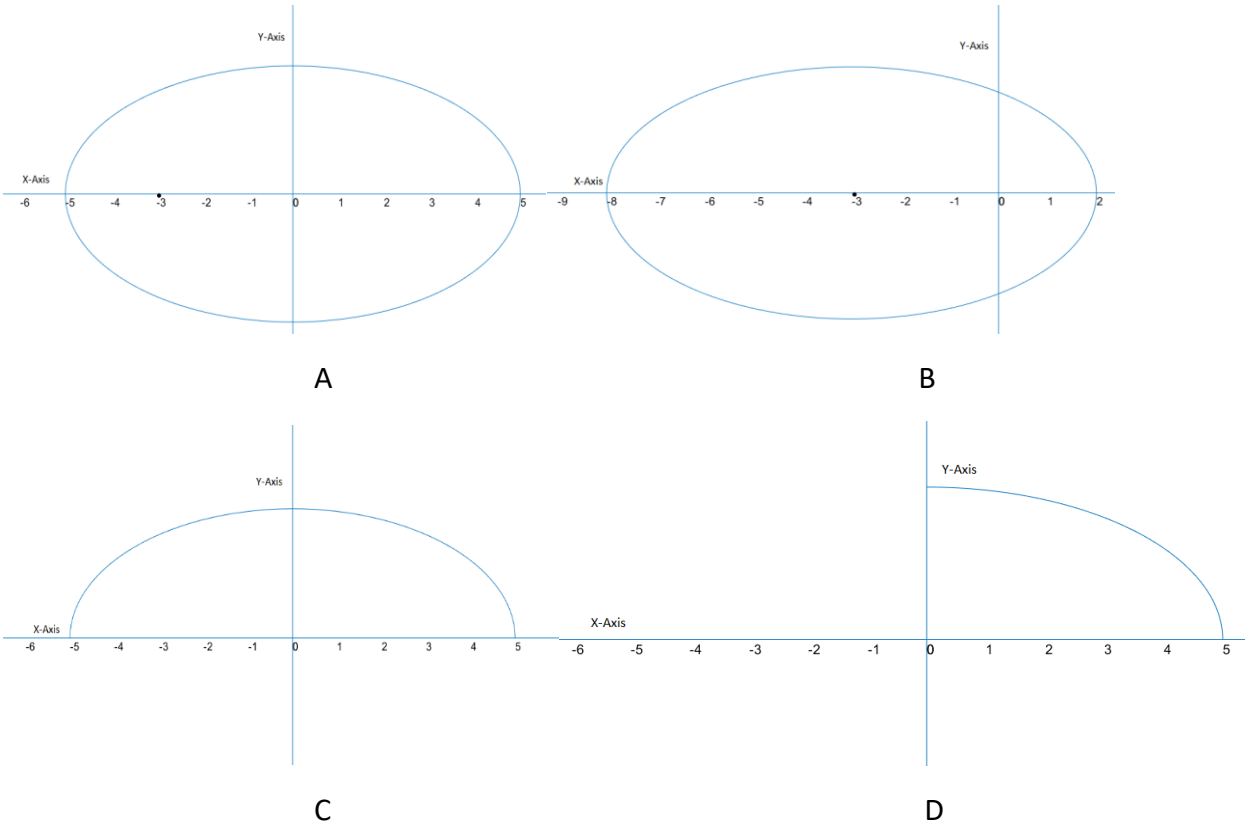


Figure 2.2 Parametric Representation of a circle A) Center (0,0) and radius 5 B) Center (-3,0) and radius 5 C) Semi-Circle D) Arc with radius 5

Figure2.2 shows the resultant circles from an equation from table. The figure 2.2 A shows circle with center (0,0) and radius 5. Figure 2.2 B shows the circle with different center (-3,0) and parameter value running from 0 to 2*pi. By changing the value of parameter from “0 to 2pi” to “0 to pi” we got semi- circle. Figure 2.2 D depicts the arc which is result of parameter value o to pi/2.

In above illustration, t was the only variable that was varied over a certain range. Now, lets consider radius as a variable and change the value over some interval.

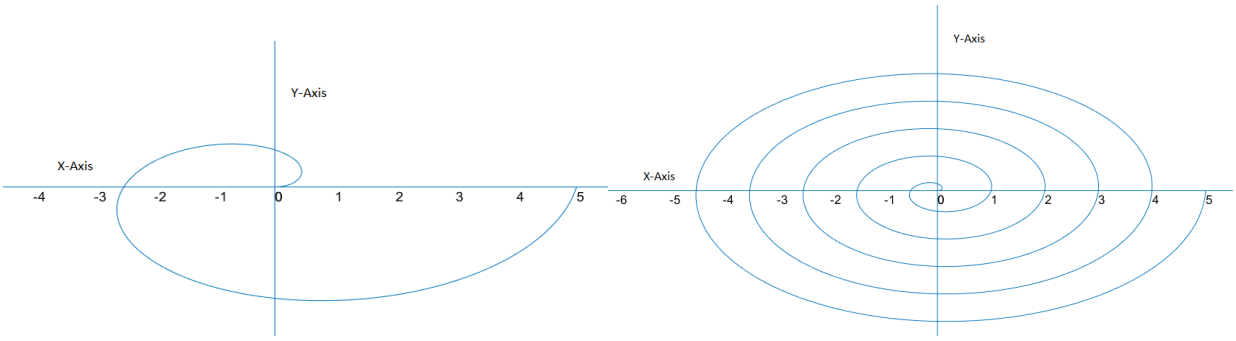


Figure 2.3 Spiral 2D Plot

Two variables have generated a spiral geometry. In the first figure 2.3 the domains are 0 <= r <= 5 and 0 <= t <= Pi. Where in the second figure 2.3, 0 <= r <= 5 and 0 <= t <= 10 * Pi.

Two variables created a spiral in a plane. Let's introduce third dimension to our circle definition and play with other variables.

$$Y = Y_0 + r\sin(t)$$

$$X = X_0 + r\cos(t)$$

$$Z = H\ ;$$

Here, new variable H is added. If we vary the value of H over certain range than the result will be cylinder or helix, depending upon whether the input is vector or matrix. Consider a vector as an input with variables, $X_0 = Y_0 = 0$, t = 0 to 2π, H = 5

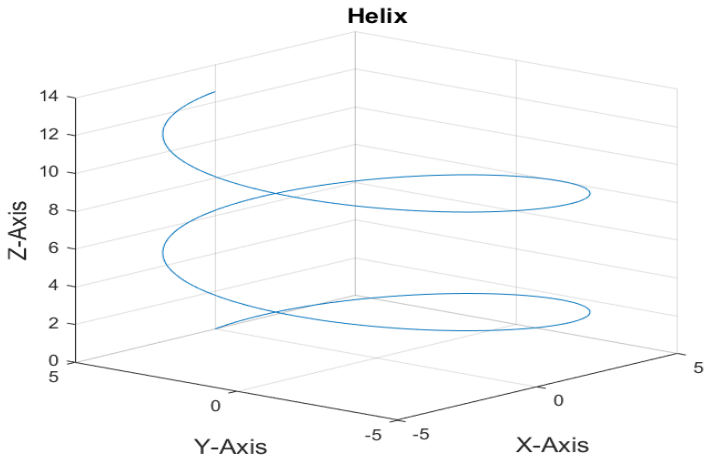


Figure 2.4 Helix

The following figure depicts a result for a matrix input.

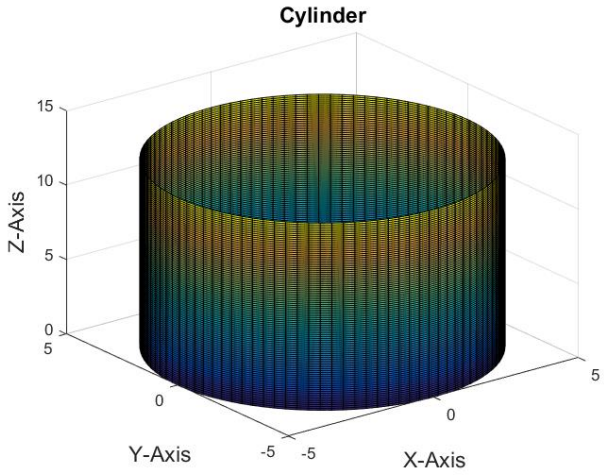


Figure 2.5 Cylinder

Starting from a simple circle, by adding few variables and changing magnitude resulted in a variety of geometries. This kind freedom is required to handle complicated geometries. Parametric representation is flexible, handy and programming friendly.

2.3 Curve Classification:

1) Plane curves and Space curves

If a curve lies in any of the following plane X-Y, Y-Z, X-Z than a curve is a plane curve. It may be closed, open, self-intersecting. Ellipse, Circle, Triangle, line, parabola, hyperbola are examples of a planar curve. Bezier and B-spline can be represented as a plane curve.

A curve which is not a plane curve, is a space curve. It does not lie in a single plane. The figure 2.4 shows that helix is a space curve. On the other hand, figure 2.2 circle and arc are plane curve.

2) Curves of free forms and known forms

Triangle, circle, ellipse, parabola, line is an example of known form where curve has predefined forms. Bezier, Hermite, B-spline and NURBS Curve are free form curve which are governed by the movement of a control points and the degree of the curves. Free form curves have a greater flexibility and advantages when it comes to fit complex curves and surfaces.[11]

3) Interpolation curves and approximation curves

Hermite is an example of polynomial interpolation. It interpolates tah point and passes through it. The curve that does not pass through data points called approximation curve. [4]

2.4 Free Form Curves

We have discussed earlier free form curves and types. In this section, mathematical representation is defined for each curve.

1) Hermite Curve:

Hermite or parametric cubic curve is an interpolation curve. It passes through data points to define a curve. There are ways to define Hermite curve.

A) Algebraic form
B) Geometric form
C) Four-point form

A) Algebraic form:

The algebraic form is given by following equation. [3]

$$X(u) = A_x * u^3 + B_x * u^2 + C_x * u + d_x \quad (2)$$

$$Y(u) = A_y * u^3 + B_y * u^2 + C_y * u + d_y$$

$$Z(u) = A_z * u^3 + B_z * u^2 + C_z * u + d_z$$

$$0 \leq u \leq 1$$

The algebraic form is common form of representing Hermite curve. The algebraic form is a third order algebraic equation that is why this curve also called as parametric cubic curve. It has 12 algebraic coefficients which define the curve shape, size and location is space. Playing with 12 coefficients is not very effective way to define and modify the curve as well as that does not make any sense of a curve. Therefore, practical approach, the geometric form has been developed from algebraic form.

B) Geometric form:

$$P(u) = (2u^3 - 3u^2 + 1)P(0) + (-2u^3 + 3u^2)P(1) + (u^3 - 2u^2 + u)P^u(0) + (u^3 - u^2)P^u(1)$$

The above equation represents the geometric form of a curve. P(u) is a vector form, where u is a parametric variable varying from 0 to 1. P(0) and P(1) are starting and end points respectively. $P^u(0)$ and $P^u(1)$ are starting and end tangent vectors respectively. [3]

Where,

$$B_1 = (2u^3 - 3u^2 + 1),\ B_2 = (-2u^3 + 3u^2),\ B_3 = (u^3 - 2u^2 + u),\ B_4 = (u^3 - u^2)$$

Here, $B_1, B_2, B_3, and\ B_4$ are basis functions of the Hermite curve and $P(0), P(1), P^u(0), and\ P^u(1)$ are geometric coefficients. It is easy to define and modify geometric coefficients than algebraic coefficients.

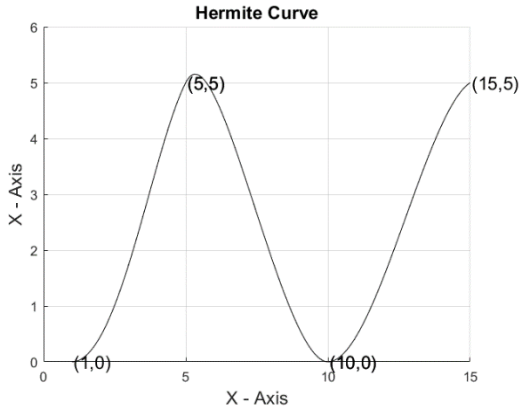


Figure 2.6 Hermite Curve

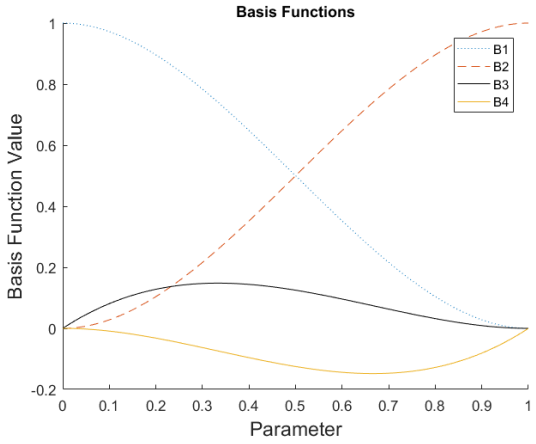


Figure 2.7 Hermite Basis Function

Basis functions of Hermite are axis independent. Only one variable u controls them. Basis functions are responsible for calculating intermediate points values. The figure 2.6 and 2.7 shows the Hermite curve and basis functions for three data points correspondingly.

C) Four-point form

In this point, the curve is defined by supplying four distinct points in space located at equal parametric interval. Four-point form is derived from algebraic from for four parameter values u = 0; u = 1/3, u = 2/3, and u = 1. Following equation, we get after solving set of equations for previously defined parametric values.

$$\begin{aligned} P(u) = {} & (-4.5u^3 + 9u^2 - 5.5u + 1)P_1 \\ & + (13.5u^3 - 22.5u^2 + 9u)P_2 \\ & + (-13.5u^3 + 18u^2 - 4.5u)P_3 + (4.5u^3 - 4.5u^2 + u)P_4 \end{aligned} \tag{3}$$

Here, P_1, P_2, P_3, P_4 are data points where curve passes through.

2) Bezier Curve:

In designing complex structures, intuition about the geometric object is important. A curve or surface defined a way that designer can have the intuition about the change of shape, more control over design, etc. Hermite curve interpolates the given control points and passes through it. Changing the shape of the Hermite curve is not intuitive and does not give a clear idea about how the curve is going to change the shape. It has limited control and. Bezier curve provides a solution to this problem. [3]

$$P(u) = \sum_{i=0}^{n} P_i B_{i,n}(u) \quad (4)$$

$$0 \leq u \leq 1$$

$$Where, B_{i,n} = \binom{n}{i} u^i (1-u)^{n-i} \quad (5)$$

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$$

$P(u)$ represents Bezier curve with parameter u over domain 0 to 1. $B_{i,n}$ is a Bernstein polynomials that govern the behavior of the curve with (n+1) control points. The Bernstein polynomial gives an nth – degree polynomial. Therefore, Degree of the Bezier curve depends upon several control points. Bezier is approximation curve. It only interpolates first and last points and approximated by intermediate points. [13]

Properties of Bezier Curve:

- The tangent of the curve is defined by first two points and last two points $P_0, P_1, and\ P_n, P_{n-1}$. Similarly, the curvature at end points defined by first three points P_0, P_1, P_2 and last three points and $P_n,\ P_{n-1}, P_{n-2}$ as follows,

$$K_0 = \frac{2|(P_1 - P_0) * (P_2 - P_1)|}{3|(P_1 - P_0)|^3}$$

$$K_1 = \frac{2|(P_{n-1} - P_{n-2}) * (P_n - P_{n-1})|}{3|(P_n - P_{n-1})|^3}$$

  Where, K_0, K_1 represents curvature at first and last points respectively.
- According to the definition of Bernstein polynomial, u and (1-u) are symmetric. Symmetric means, reversing the order of the control points does not change the shape of the curve.
- In geometric modeling, the transformation is a vital part. The Bezier curve is invariant under geometric transformations.
- Convex hull property: convex hull is polygon formed by the control vertices. This property states that the resultant curve always lies under convex hull. This property defines a bound for the curve as the curve never goes out of the convex hull. The following figures illustrate this property.

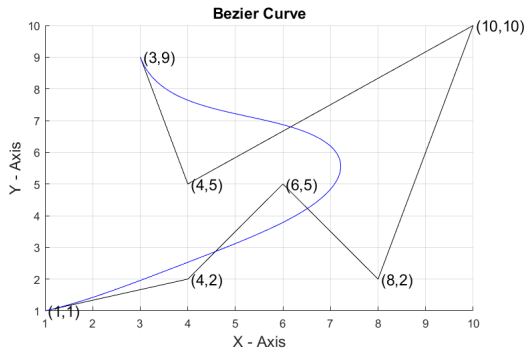


Figure 2.8 Bezier Curve

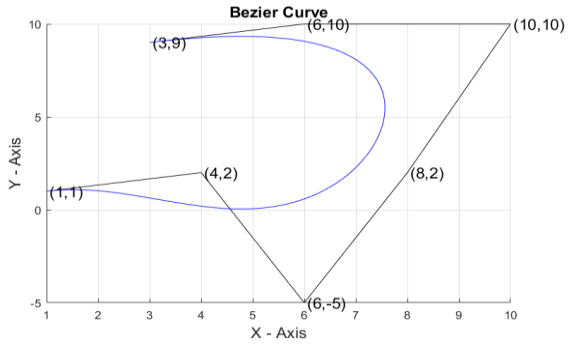


Figure 2.9 Modified Bezier Curve

- Partition of unity:

$$\sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} = 1$$

Summation of all the basis function at any parameter value u is equal to 1. Bezier curve is defined by assigning distinct weights to different coordinates.

Sometimes it is hard to handle higher degree curve. So, composite Bezier curves used to overcome this problem. The application of composite Bezier curve gets limited as a designer must deal with continuity issue. B- spline curve is a generalization of a Bezier curve and has advantages over Bezier curve as discussed in the following section.

3) B-spline Curve

B-spline curve consists of more than one curve segments and knot vectors, divide the curve into different segments. Dividing a curve into several segments provides a local modification control over curve. Bezier curve does not have local modification control. Modifying one control point changes the whole curve in the Bezier curve. On the other hand, changing one control point, only certain number of segments get modified. The degree of the curve is dependent on a number of control points in Bezier. Degree dependency on number control points poses a problem when the designer is using more than 15 points to define a single curve segment. Handling a 14^{th} degree curve is not recommended for some applications like finding an intersection of two curves. The B-spline curve has a solution for dependency problem. The degree of the B-spline curve is independent of a number of control points used. The designer can explicitly define a desired degree of the curve. B-spline curve requires three inputs number of control points, knot vector, and degree. B-spline curve interpolates first and end, and approximates intermediate points. Mathematical Formulation: [3]

$$P(u) = \sum_{i=0}^{n} P_i N_{i,k} \qquad (6)$$

$$Where, \quad N_{i,k} = \frac{(u - t_i) N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u) N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

$N_{i,k}$ is basis function for B-spline curve. It is a recursive function and uses previous values to calculate the next value. The first basis function is defined as follows.

$$N_{i,1} = 1 \; if \; t_i \leq u \leq t_{i+1}$$

$$= 0 \; Otherwise$$

t represents knot vector, and parameter K defined the degree (K-1) of the curve. Knot vector is responsible for breaking the curve into different segments. The standard way of defining knot vector presented below. Total number of knot vector is defined as n+k+1 (number of control points + degree of curve).

$$t_j = 0 \qquad if \; j < K \tag{7}$$

$$t_j = i + K + 1 \quad if \;\; K \leq j \leq n$$

$$t_j = n - K + 2 \quad if \;\; j > n$$

The knot vector definition presented above is for clamped b-spline curve. Modifying the knot vector results in the different shape of the curve. The parameter value u has ranged from 0 to n-k+2.

$$0 \leq u \leq n - K + 2$$

If the designer is using 5 control points with 2nd degree curve, then parameter range is $0 \leq u \leq 4$. Curve is devided into 4 sections 0….1, 1….2, 2….3, and 3….4.

Classification of the b-spline curve

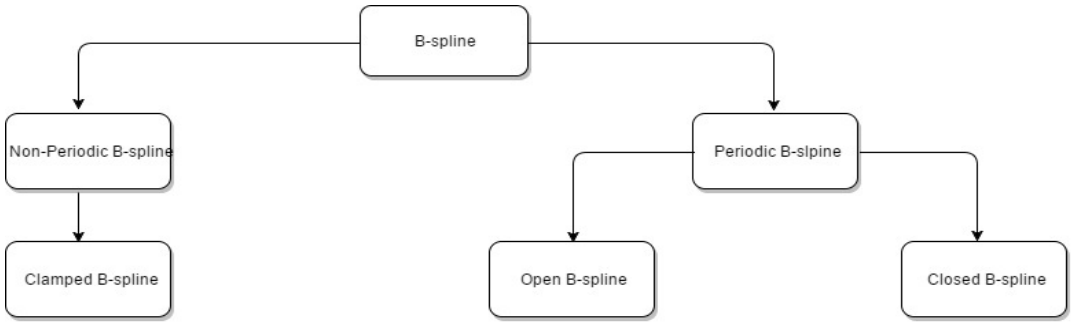


Figure 2.10 B-spline classification

B-spline is classified in two ways, Non-periodic and periodic. Behaviour of knot vector plays an inportant role in classifying b-spline curve. In the next chapter, classification has been discussed along with the algorithm for B-spline curves and surfaces.

Properties:

- Local modification control
- Convex Hull proprerty
- Partition of unity
- Invarience in transformations

4) Non Uniform Rational B-spline (NURBS):

NURBS is more powerful than b-spline. NURBS is a versatile tool in geometric modeling, and it has become industry standard. NURBS is a rational version of a B-spline curve. The B-spline curve cannot represent a circle, conic curves, ellipse, etc. NURBS has an advnatage of representing known form curves with added degree of freedom. The mathematical formulation is described below. [7]

$$P(u) = \frac{\sum_{i=0}^{n} h_i P_i N_{i,k}(u)}{\sum_{i=0}^{n} h_i N_{i,k}(u)} \quad (8)$$

Here, h_i are weights assigned to each control points. NURBS is the generalization of a B-spline curve. Weights are extra degree of freedom, one can easily modfy weight at any control point to obtain desired geometry.

2.5 STEP AP 214

In geometric modeling, Geometric construction and data exchange between CAD Package is an important task. STEP standard has been depicted to complete data exchange task. ISO 10303 known as " Automation systems and integration – product data representation and exchange". ISO 10303 informally known as STEP which means " Standard for the Exchange of Product model data". STEP file has the capability to represent 2D Sketch or 3D Model data into CAD software. ISO 10303 consists of many parts. [8]

1) Description Methods
    - Part 11 - Express data modeling language
    - Part 12 - Express-I
    - Part 14 - Express X
2) Implementation Methods
    - Part 21 - STEP FILE: Clear text encoding of the exchange structure
    - Part 22 - SDAI Standard data acess interface specification
    - Part 28 - STEP XML
3) Conformance Testing methodology and framework
    - Part 31 - General Concepts
    - Part 32 - Requirement on testing laboratories and clients

- Part 34 - Abstract test method for applization protocol implementations

4) Application Protocols
    - Part 203 - Configuration controlled 3D Designs of mechanical parts and assemblies
    - Part 214 - Core data for automotive mechanical design processes
5) Integrated Application Resource Models
    - Part 101 - Graughting
    - Part 104 - Finite Elment Aalysis
6) Integrated General Resources
    - Part 41 - Fundamentals of product description and support
    - Part 42 - Geometric and topological representation

In many CAD Softwares, to save a 3D CAD Model data, AP203 and AP214, two application protocols have been used widely. In this research work, data structure for AP214 has been adapted for data exchange.

Data Structure for ISO 10303 - 21. [10]

ISO-10303-21;
HEADER;
FILE_DESCRIPTION(
/* description */ (),
/* implementation_level */);
FILE_NAME(
/* name */
/* time_stamp */,
/* author */
/* organization */
/* preprocessor_version */ ' ',
/* originating_system *
/* authorization */
FILE_SCHEMA (('AUTOMOTIVE_DESIGN ‘));
ENDSEC;
DATA;
#31=EDGE_CURVE('',#39,#40,#35,.T.);
#32=EDGE_CURVE('',#**40**,#**41**,#**36**,.T.);
#**36**=B_SPLINE_CURVE_WITH_KNOTS('',3,(#94,#95,#96,#97),.UNSPECIFIED.,.**F**.,.**F**.,(4,4),(0,1),.UNSPECIFIED.);
#94=CARTESIAN_POINT('',(31.000000,0.000000,0.000000));
#95=CARTESIAN_POINT('',(32.680000,-1.960000,4.350000));
#96=CARTESIAN_POINT('',(28.330000,3.110000,8.660000));
#97=CARTESIAN_POINT('',(31.000000,0.000000,31.000000));
#33=EDGE_CURVE('',#41,#42,#37,.T.);
#34=EDGE_CURVE('',#42,#39,#38,.T.);

```
#39=VERTEX_POINT('',#70);
#70=CARTESIAN_POINT('',(0.000000,0.000000,0.000000));
#40=VERTEX_POINT('',#71);
#71=CARTESIAN_POINT('',(31.000000,0.000000,0.000000));
#41=VERTEX_POINT('',#72);
#72=CARTESIAN_POINT('',(31.000000,0.000000,31.000000));
#42=VERTEX_POINT('',#73);#55=(NAMED_UNIT(*)PLANE_ANGLE_UNIT());
#56=DIMENSIONAL_EXPONENTS(0.,0.,0.,0.,0.,0.);
#57=PLANE_ANGLE_MEASURE_WITH_UNIT(PLANE_ANGLE_MEASURE(0.01745329252),#55);
#58=(CONVERSION_BASED_UNIT('DEGREES',#57)NAMED_UNIT(#56)PLANE_ANGLE_UNIT());
#59=(NAMED_UNIT(*)SI_UNIT($,.STERADIAN.)SOLID_ANGLE_UNIT());
#44=SHAPE_DEFINITION_REPRESENTATION(#45,#62);
#45=PRODUCT_DEFINITION_SHAPE('Document','',#47);
#46=PRODUCT_DEFINITION_CONTEXT('3D Mechanical Parts',#51,'design');
#47=PRODUCT_DEFINITION('A','First version',#48,#46);
ENDSEC;
END-ISO-10303-21;
```

The file consists of two sections, 1) Header 2) Data. The header section contains information like file description, file name, creator of the file, time, etc. Data section represents the Geometry data, Units, Application protocols. Instance number is used to declare data for example, cartesian point is defined as #Instance Numer=CARTESIAN_POINT('Name',(X-coordinate,Y-coordinate,Z-coordinate)). Instance number helps in the mapping of entity. From above example, Edge curve is defined by two vertex points and one B-spline curve. Therefore, two vertex points and B-spline Curve are mapped into edge curve entity. The boolean and logical values are written in capital letters. In B-spline curve entity, two ".F." boolean operators are presented which states that the curve is not closed and self intersecting. [6]

2.6 Significance of NURBS in Additive Manufacturing

Complex structures are possible to produce with Additive Manufacturing without any constraints. As discussed earlier, conventional manufacturing processes have restrictions in developing complex geometries. Topology optimization method gives us a conceptual design which can be modified by the designers the way they want. [5]

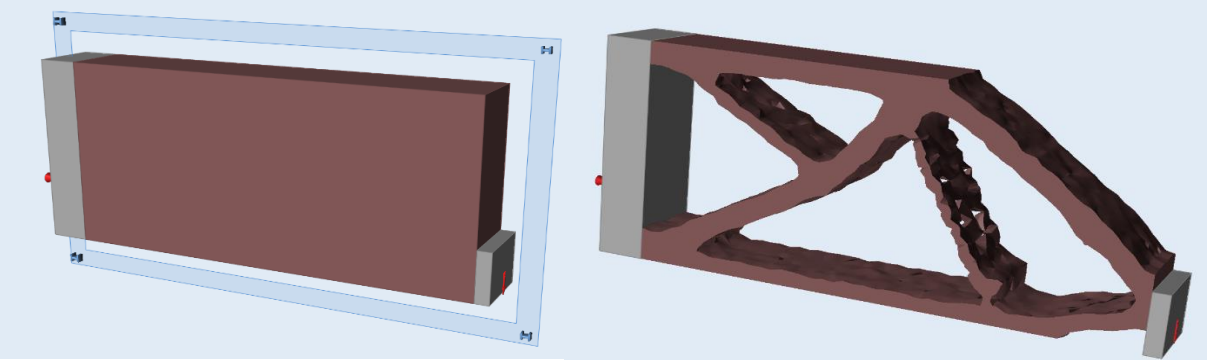


Figure 2.11 Sample block

Figure 2.12 optimized body

Sample block has been taken to illustrate the importance of NURBS in figure2.11. Solidthinking Inspire tool is used for the optimization process. 1000 N load is applied at the right end, and opposite end is fixed. Topology optimization is run with the objective of mass minimization.

Figure 2.12 shows the result of topology optimization consists of rough surfaces. To fit these sections, NURBS modeling is required. Standard CAD Tools can be used, but a designer may not get flexibility. Figure 2.13 illustrates the usage of NURBS tools in fitting one cross section. Designers have freedom to modify this cross section by moving control points, handles, and faces. [5]

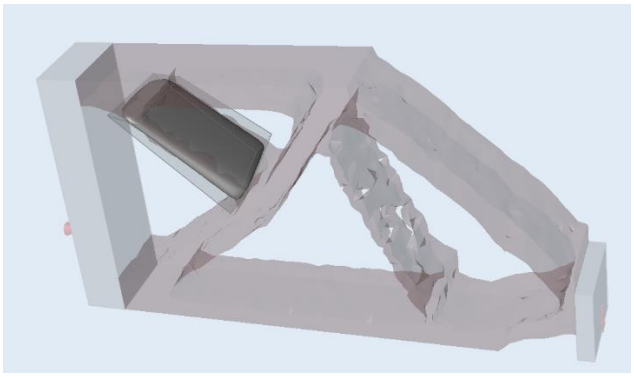


Figure 2.13 NURBS fit

Chapter 3 Methodology

In methodology segment, detailed understanding of properties and algorithms have been presented for B-spline, Bezier, NURBS and STEP AP 214 file along with examples.

3.1 Bezier Curve Formulation

$$P(u) = \sum_{i=0}^{n} P_i B_{i,n}(u) \tag{9}$$

$$0 \leq u \leq 1$$

$$Where, B_{i,n} = \binom{n}{i} u^i (1-u)^{n-i}$$

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$$

Bezier Curve properties:

1) Global propagation.

The figure 3.1 shows the 5th degree Bezier curve with 6 control points. Mathematical formulation of this curve:

$$P(u) = \sum_{i=0}^{5} P_i B_{i,5}(u)$$

$$P(u) = P_0 B_{0,5} + P_1 B_{1,5} + P_2 B_{2,5} + P_3 B_{3,5} + P_4 B_{4,5} + P_5 B_{5,5} \tag{10}$$

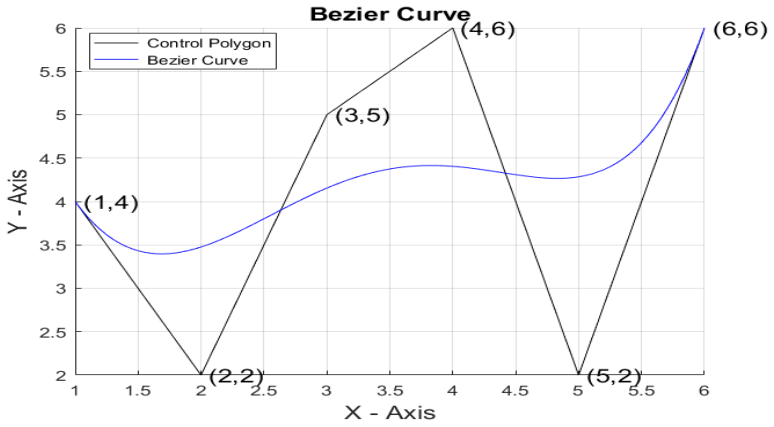


Figure 3.1 Bezier curve with 6 control points

Let's modify the control point from (4,6) to (4,7). The resulting figure 3.2 shows below. Red dotted line is the modified Bezier curve and effect is shown clearly. Moving one control point affects the whole curve. This propagation called global propagation because changing one coordinate alter the whole curve segment. Consider equation 10 and change the 5th

control point by adding position vector $\overrightarrow{v} = \hat{j}$.The resultant vector for new curve is given by W(u),

$$W(u) = P_0B_{0,5} + P_1B_{1,5} + P_2B_{2,5} + P_3B_{3,5} + (P_4 + \vec{v})B_{4,5} + P_5B_{5,5}$$

$$W(u) = P_0B_{0,5} + P_1B_{1,5} + P_2B_{2,5} + P_3B_{3,5} + P_4B_{4,5} + \vec{v}B_{4,5} + P_5B_{5,5}$$

$$W(u) = P_0B_{0,5} + P_1B_{1,5} + P_2B_{2,5} + P_3B_{3,5} + P_4B_{4,5} + P_5B_{5,5} + \vec{v}B_{4,5}$$

$$From\ equiation\ (10)$$

$$W(u) = P(u) + \vec{v}B_{4,5} \qquad (11)$$

The equation states that new curve is nothing but original vector $P(u)$ plus multiplication of basis $B_{4,5}$ and position vector $\vec{v}$. Hence, change in one control point affects the whole curve segment. The figure 3.2 shows the result.

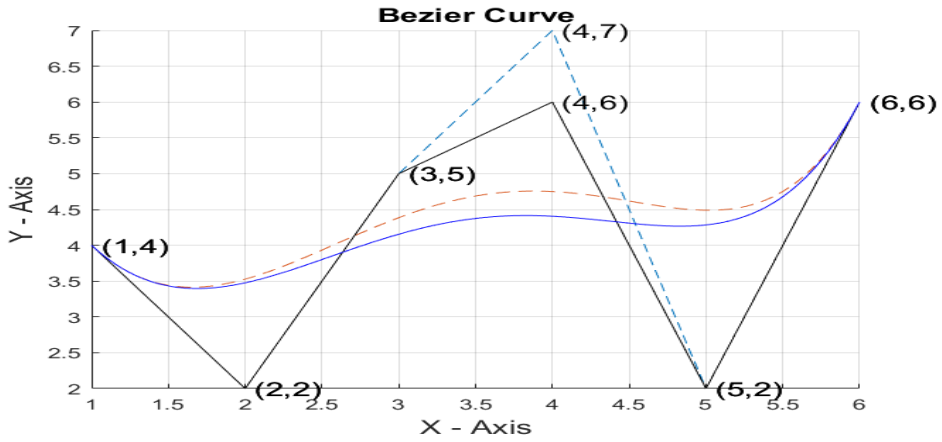


Figure 3.2 Modified Bezier Curve

The red dotted line demonstrates the new Bezier curve and the blue line is the original Bezier curve. In some applications, global propagation is not required. If designer wants to modify some portion of the curve then local modification plays important part. B-spline curve has local modification capability. Algorithm has been developed from mathematical definition. The figure 3.3 represents the flowchart for the algorithm. The Bezier curve computation is simple and straight forward. One for loop is running from o to number of control points which calculates the basis function and calculating X, Y and Z components. The degree is dependent on number of control points. The algorithm requires X, Y, Z Coordinates and output is curve segment.

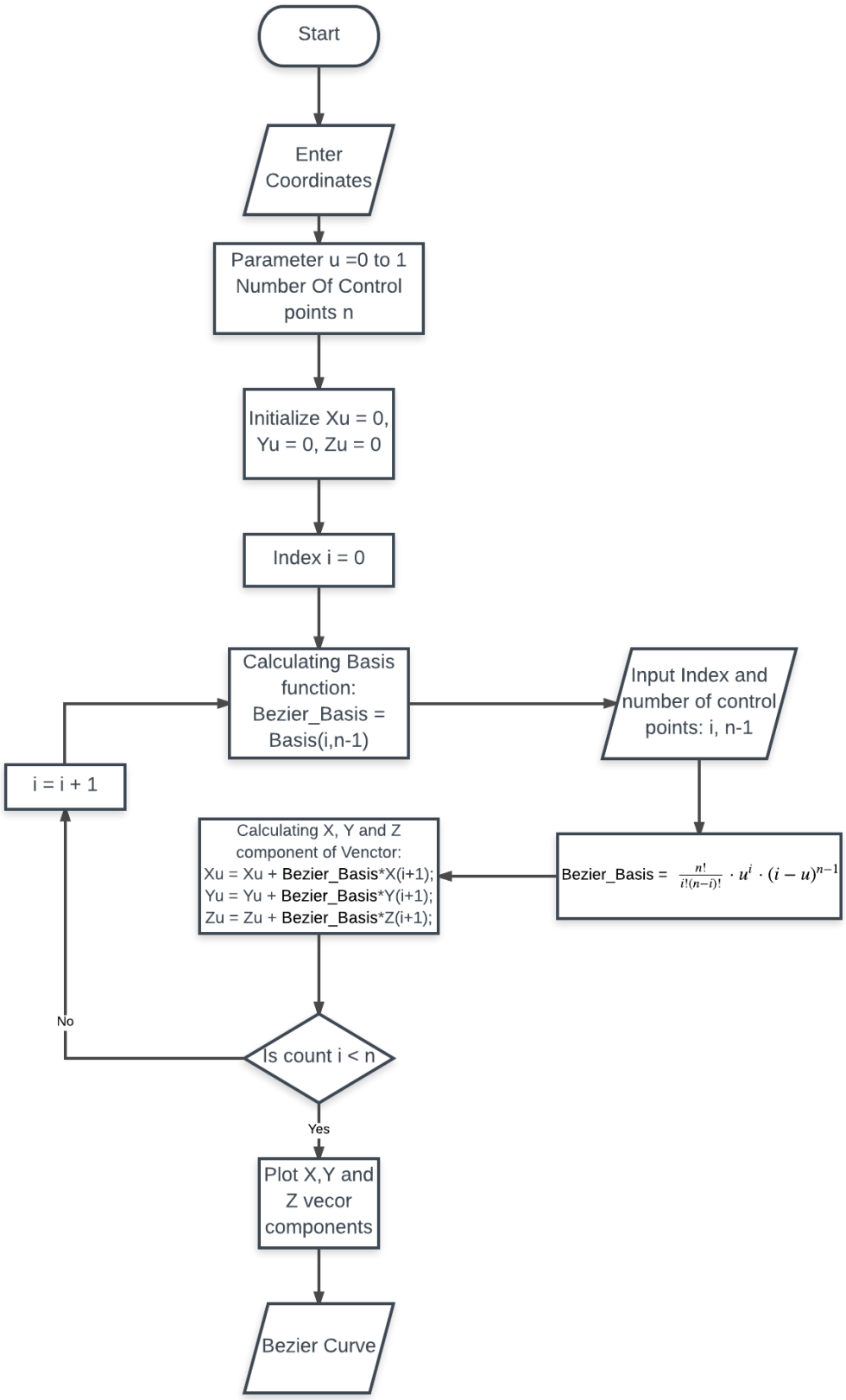


Figure 3.3 Algorithm for Bezier curve

3.1.1 Control Points and Basis functions

Behavior and influence of the curve governed by basis functions. The following figure 3.4 and 3.5 shows the basis functions for 4 and 5 control points.

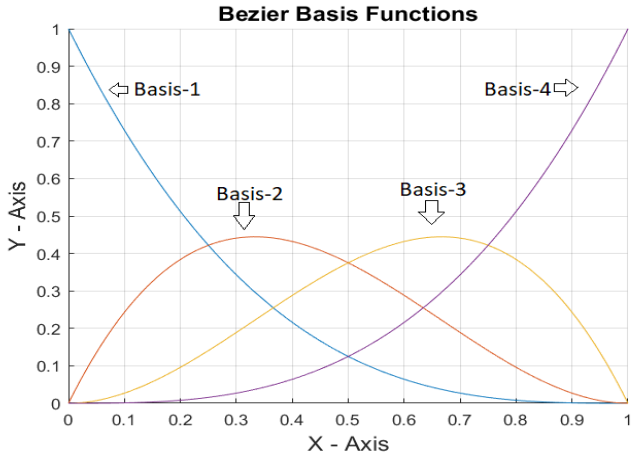


Figure 3.4 Bezier Curve basis functions for 4 control points

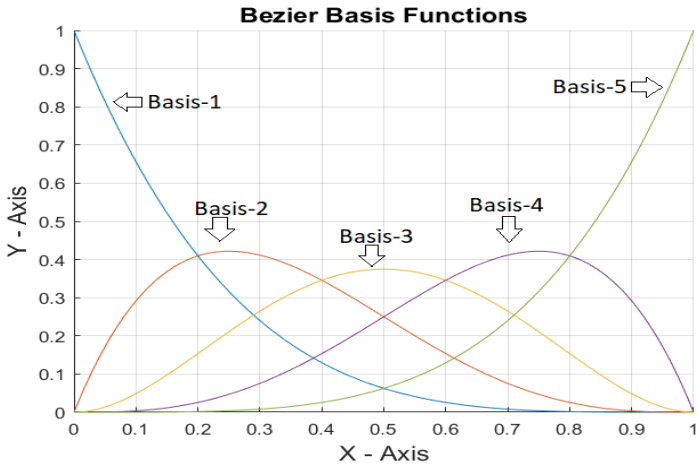


Figure 3.5 Bezier Curve basis functions for 5 control points

The first basis function has 100% influence on the first control point at u = 0. As parameter value increases the influence of the first basis functions decreases. Similarly, Basis-4 has full control at u = 1 on the last control point. At u = 1/3 and u = 2/3, Basis-2 and Basis-3 are most influential respectively. All the basis function starts at u = 0 and ends at u = 1. This behavior shows the global propagation. Contrary, B-spline curve's basis functions do not have the influence on full range. All the basis functions are symmetric to u=0.5. Basis-1 is a mirror of basis-4, same way basis-2 is a mirror of basis-3 in figure 3.4. Basis functions act as weights for control points. Defining coincident point increases the weightage of that coordinate. The subsequent series of figures demonstrate the effect of adding more weight to the single control point.

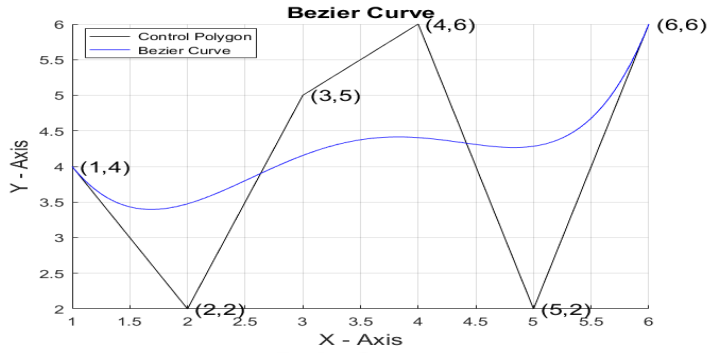


Figure 3.6 a Single Point

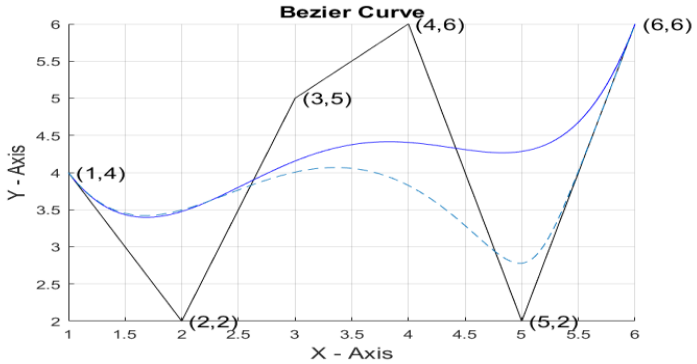


Figure 3.6 b Two coincident points

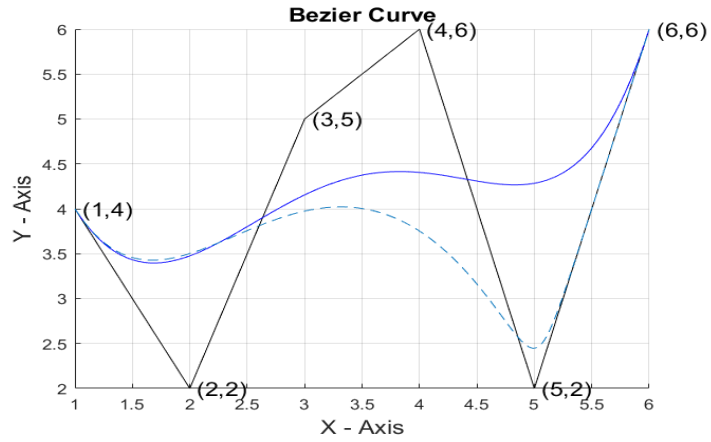


Figure 3.6 c Three coincident points

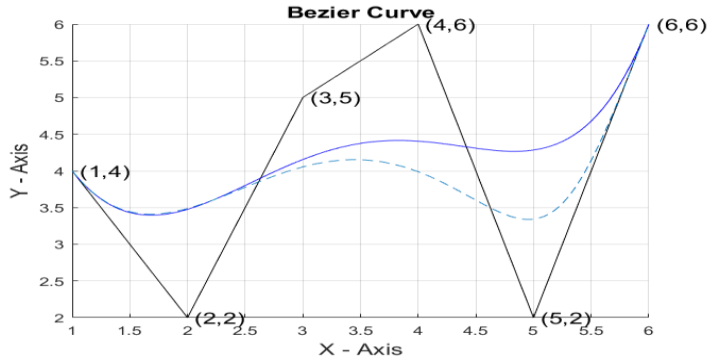


Figure 3.6 d Four coincident points

Figure 3.6 effect of multiple coincident points on the Bezier curve

The figure 3.6 (a) is defined by 6 control points. The figure 3.6 (b) displays defining the same control point (5,2) twice. The dotted line is the new curve and solid blue line depicts original curve. As we increase the coincident points, the curve is getting pulled towards the coincident point. The figure 3.6 (c) shows the same control point again. Result is clearly seen, the curve is pulled more and more as we add more equivalent ontrol points. In other words, we are adding weightages to a single contrl point.

3.1.2 Composite Bezier Curve

Single bezier curve segment is not always suitable for modeling. Multiple curve srgment is required to reduce the degree of the curve and complexity. Sometimes defining a single bezier curve with 14 or 20 control point is not suitable. Therefore, it is feasible to break the curve into multiple curve segments. Continuity issue comes into play for composite curve segments. Parametric continuity between curve segments is defined. Parametric continuity defines in following ways.[4]

C^0 = curves are attached

C^1 = tangent continuous (First derivatives are same)

C^2 = Curvature Continuous (Second Derivatives are same)

C^n = nth derivatives are same

The figures 3.7 to 3.10 shows the pictorial representation of types of continuity.

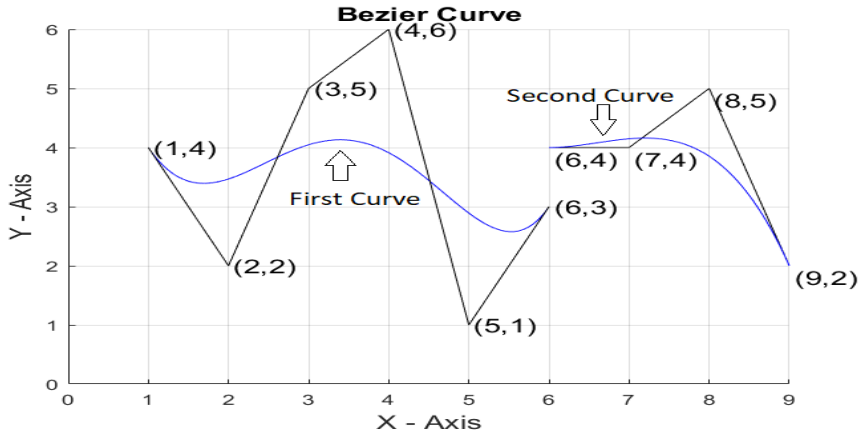


figure 3.7 Discontinuous Bezier Curves

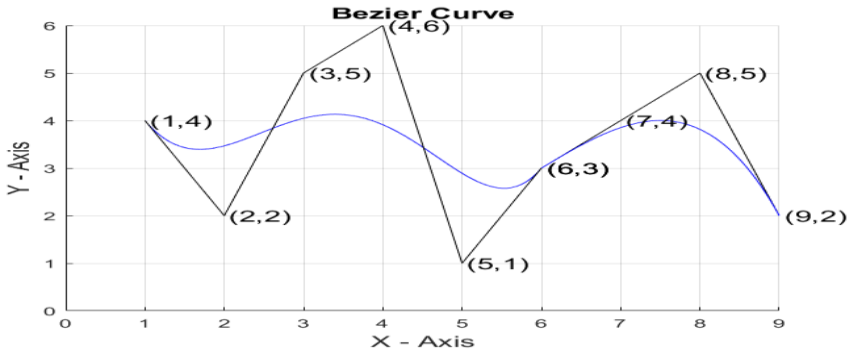


figure 3.8 Bezier curves with C^0 continuity

When last point of the first curve and first point of the second curve are joint that represents C^0 continuity. C^0 ensures that curves are joint without any continuity.

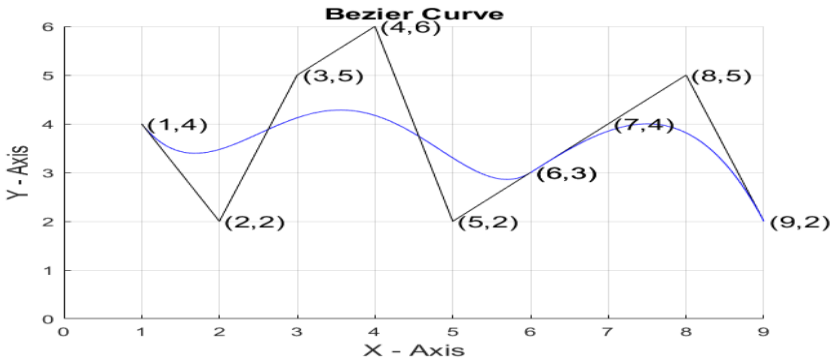


Figure 3.9 Bezier curves C^1 continuity

Ths figure 3.9 portrays tangent contunuity. Where, tangent vectors are same for both the curve. Accorging to the property of the bezier curve, first two point and last two points are tangent of the curve. To make tangent continuous there is a one condition. Three point must be co-linear of both the curve segments as illustrated in figure (5,2), (6,3). (7,4). The last two points of the first curve and first two points of the second curve must be c-linear to make sure both the curves are tangent ontinuous at the meeting point.

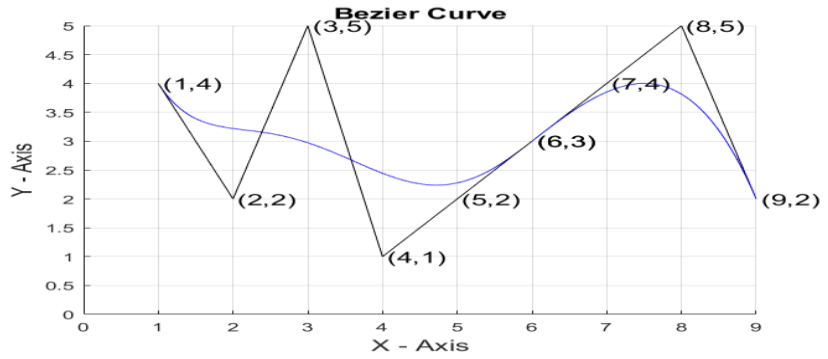


Figure 3.10 Bezier curves C^2 continuity

Tangent continuity sometime does not work in keen corners. Therefore, curvature continuity C^2 is required. Curvature of the bezier curve is defined by three start and end points. Procedure is same, five points have to be co-linear inorder to satisfy curvature continuity.

If we modify point (6,3) than we have to change other points inorder to maintain curature continuity which is sometimes not quit efficient in modeling. The handeling become tedious and time consuming.

So, Bezier curve is flexible and has advantages over Hermite curve. But it has several disadvantages like global propogation, degree dependancy. B-spline resolves the issue of global propogation and degree dependency.

3.2 B-spline basis function

The B-spline curve is defined by following equation 12.

$$P(u) = \sum_{i=0}^{n} P_i N_{i,k} \tag{12}$$

$$Where, \quad N_{i,k} = \frac{(u - t_i)N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

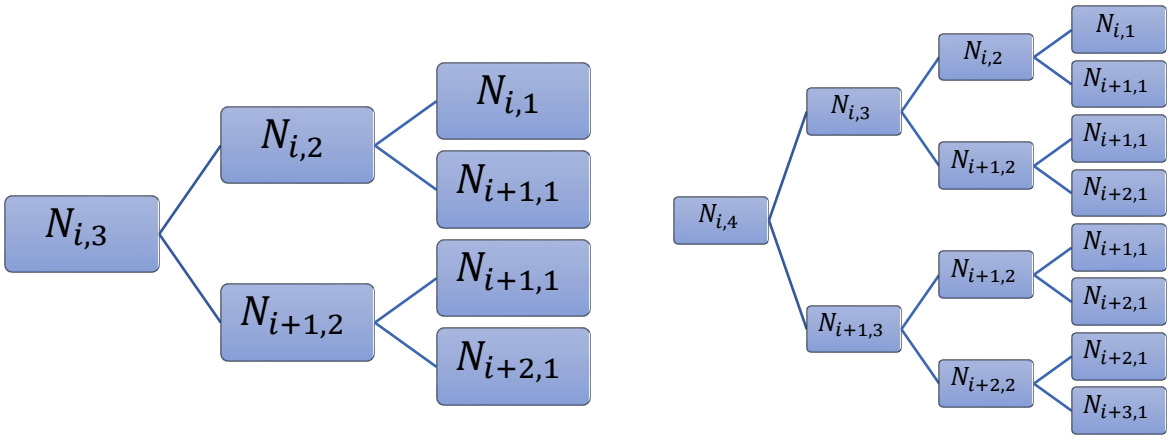
$$N_{i,1} = 1 \; if \; t_i \leq u \leq t_{i+1}$$
$$= 0 \; Otherwise$$

$$Parameter, \qquad 0 \leq u \leq n - K + 2$$

Where, Knot Vector

$$t_j = 0 \qquad if \; j < K \tag{12a}$$
$$t_j = i + K + 1 \quad if \;\; K \leq j \leq n$$
$$t_j = n - K + 2 \;\; if \;\; j > n$$

To understand the behavior of the B-spline curve, first, understand the basis function and the knot vector. The basis function is a recursive function and uses previous two values.



The flowchart shows the recursive behavior of the basis function. The following calculations are done for 7 control points with 4th order.

To calculate $N_{i,4}$ where i = 0 to 5, we need to calculate $N_{i,3}$, $N_{i,2}$, $N_{i,1}$. From above equation for knot vector t = (0 0 0 0 1 2 3 4 4 4 4) and parameter u = 0 to 4.

For the first order, k = 1

$N_{0,1}$ = 1 at u = 0

$N_{1,1}$ = 1 at u = 0

$N_{2,1}$ = 1 at u = 0

$N_{3,1}$ = 1 at $0 \le u < 1$

$N_{4,1}$ = 1 at $1 \le u < 2$

$N_{5,1}$ = 1 at $2 \le u < 3$

$N_{6,1}$ = 1 at $3 \le u \le 4$

Second Order, K = 2

According to equation 12,

$$N_{0,2} = \frac{(u - t_0)N_{0,1}(u)}{t_1 - t_0} + \frac{(t_2 - u)N_{1,1}(u)}{t_2 - t_1}$$

In these calculations, we often encounter terms like 0/0. For calculation purpose, we are considering 0/0 = 0. After simplifying above equation,

$N_{0,2}$ = 0

Similarly applying equation 12 for rest of the calculations.

$N_{1,2}$ = 0

$N_{2,2} = (1-u)N_{3,1}$

$N_{3,2} = uN_{3,1} + (2-u)N_{4,1}$

$N_{4,2} = (u-1)N_{4,1} + (3-u)N_{5,1}$

$N_{5,2} = (u-2)N_{5,1} + (4-u)N_{6,1}$

$N_{6,2} = (u-3)N_{6,1}$

Third Order, k = 3

$N_{0,3}$ = 0

$N_{1,3} = (1-u)^2 N_{3,1}$

$N_{2,3} = u((1-u) + \frac{(2-u)}{2}) N_{3,1} + (2-u)^2 N_{4,1}$

$N_{3,3} = \frac{1}{2}[u^2 N_{3,1} + N_{4,1}(u(2-u) + (3-u)(u-1) + (3-u)^2 N_{5,1}]$

$N_{4,3} = \frac{1}{2}[(u-1)^2 N_{4,1} + N_{5,1}((u-1)(3-u) + (2-u)(u-2)) + (2-u)(4-u)N_{6,1}]$

$N_{5,3} = \frac{(u-2)^2}{2} N_{5,1} + \frac{[(4-u)(u-2)+(4-u)(u-3)]}{2} N_{6,1}$

$N_{6,3} = (u-3)^2 N_{6,1}$

For fourth order k = 4,

$N_{0,4} = (1-u)^3 \boldsymbol{N_{3,1}}$

$N_{1,4} = \boldsymbol{N_{3,1}}[u(1-u)^2 + \frac{u(1-u)(2-u)}{2} + \frac{u(2-u)^2}{2}] + \boldsymbol{N_{4,1}} \frac{(2-u)^2}{2}$

$N_{2,4} = \boldsymbol{N_{3,1}}[(1-u)u^2 + \frac{(2-u)u^2}{2} + \frac{(3-u)u^2}{6}] + \boldsymbol{N_{4,1}}[\frac{(2-u)u^2}{2} + \frac{u(2-u)(3-u)}{6} + \frac{(u-1)(3-u)^2}{6}] + \boldsymbol{N_{5,1}} \frac{(3-u)^3}{6}$

$$N_{3,4} = \boldsymbol{N_{3,1}} \frac{u^3}{6} + \boldsymbol{N_{4,1}}\left[\frac{(2-u)u^2}{6} + \frac{u(u-1)(3-u)}{6} + \frac{(4-u)(u-1)^2}{6}\right] + \boldsymbol{N_{5,1}}\left[\frac{(3-u)^3}{6} + \frac{(u-1)(3-u)(4-u)}{6} + \frac{(u-2)(2-u)(4-u)}{6}\right] + \boldsymbol{N_{6,1}} \frac{(2-u)(4-u)^2}{6}$$

$$N_{4,4} = \boldsymbol{N_{4,1}} \frac{(u-1)^3}{6} + \boldsymbol{N_{5,1}}\left[\frac{(3-u)(u-1)^2}{6} + \frac{(u-2)(u-1)(2-u)}{6} + \frac{(4-u)(u-2)^2}{6}\right] + \boldsymbol{N_{6,1}}\left[\frac{(u-2)(4-u)^2}{4} + \frac{(u-1)(2-u)(4-u)}{6} + \frac{(u-3)(4-u)}{4}\right]$$

$$N_{5,4} = \boldsymbol{N_{6,1}}\left[\frac{(4-u)(u-2)^2}{4} + \frac{(u-4)(u-3)(4-u)}{2} + (4-u)(u-3)^2\right] + \boldsymbol{N_{5,1}} \frac{(u-2)^2}{4}$$

$$N_{6,4} = (u-3)^3 \boldsymbol{N_{6,1}}$$

Algorithm has been developed in MATLAB for plotting and calculating the values for basis functions. Figure 3.11 represents flow chart for basis function algorithm. In this algorithm, the inputs are desired order, parameter u, and coordinates. First, second order basis functions are calculated and stored in a zero matrix. In a second iteration, using the previous calculated values of second order, rest of the calculations are carried out. The algorithm is generalized for any number of control points and order. Order of the curve must not exceeds the number of control points. The MATLAB code is attached in an APPENDIX.

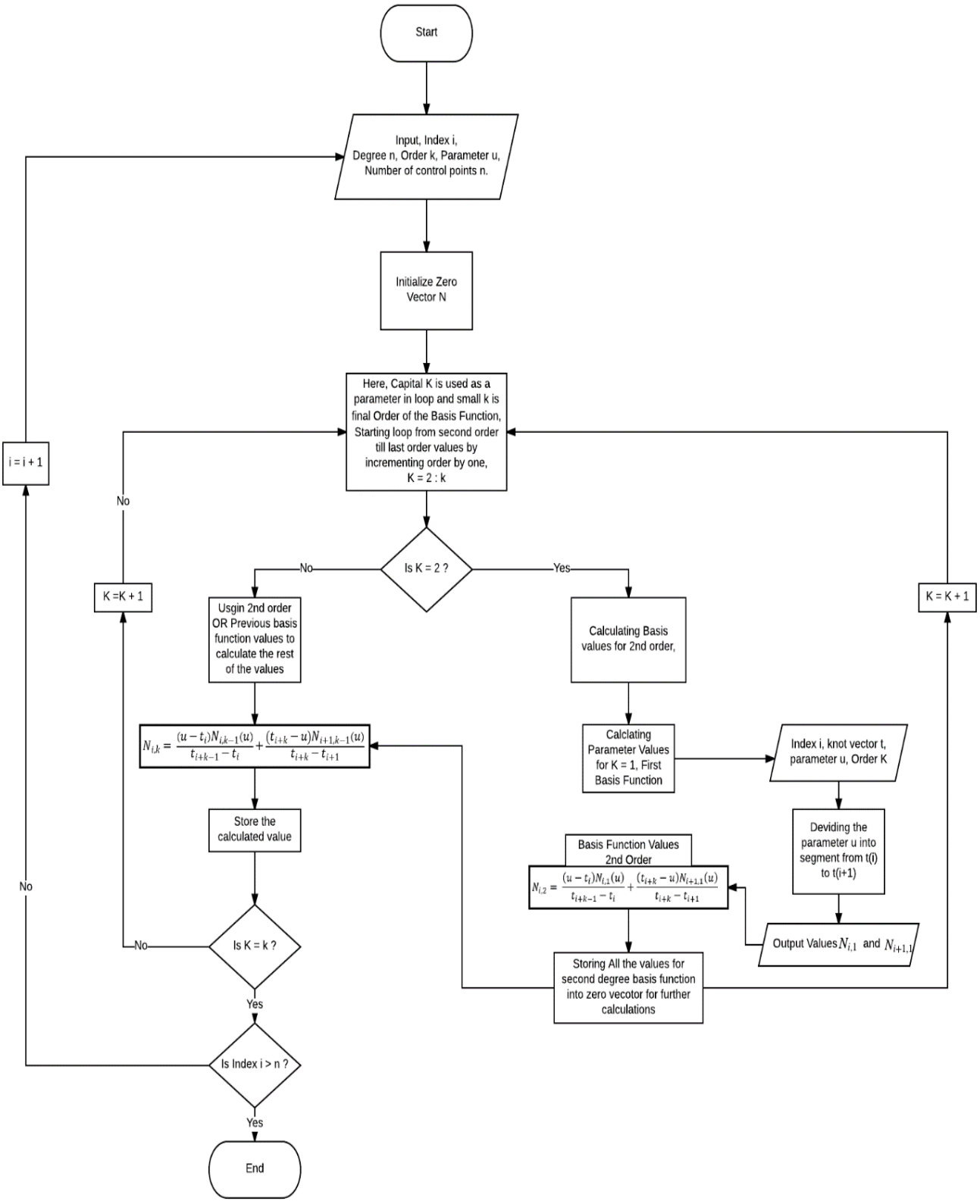


Figure 3.11 B-spline Basis Function Algorithm

The results from the algorithm for k = 4 (3rd degree, 7 control points)

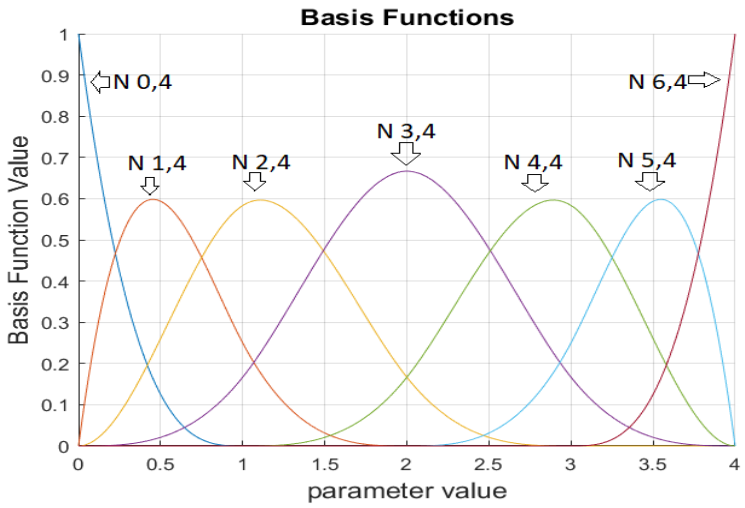


Figure 3.12 B-spline Basis Function for 8 control points and 4th order

The figure 3.12 illustrates the plot of basis functions with respect to parameter value. Local control is clearly seen as basis functions do not cover the whole parameter range except one basis function $N_{3,4}$. But as we increase the number of control points by keeping order four, none of the basis function covers the complete parameter range as shown in figure 3.13. The figure 3.14 and 3.15 represents the basis function for 3rd and 2nd order respectively.

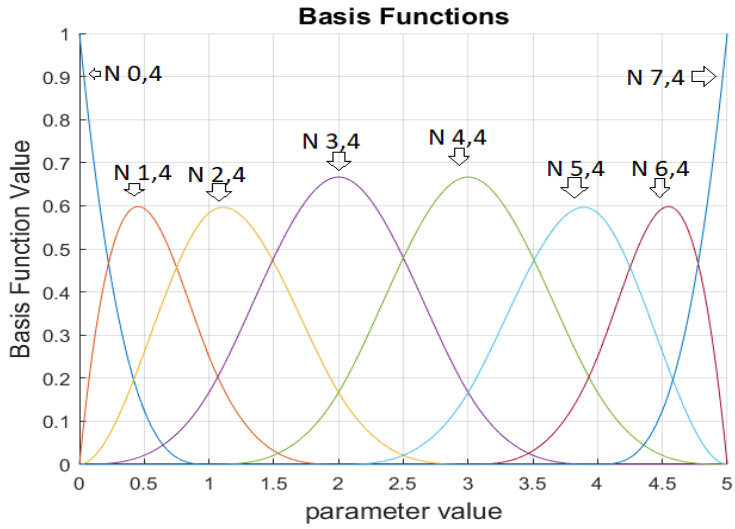


Figure 3.13 B-spline Basis Function for 9 control points and 4th order

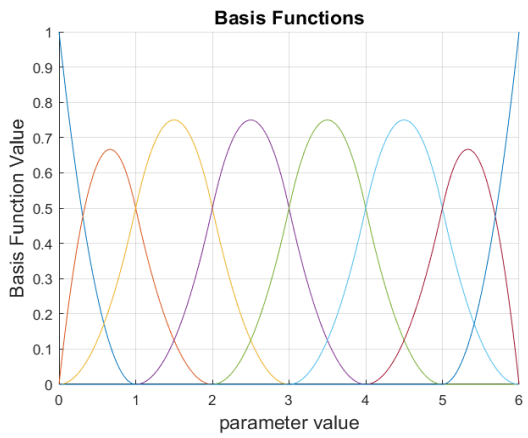


Figure 3.14 Basis Function k=3, n=7

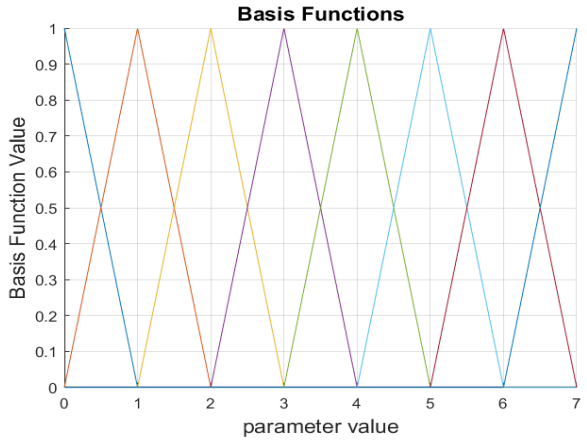


Figure 3.15 Basis Function k=3, n=7

3.3 B-spline curve formulation

$$P(u) = \sum_{i=0}^{n} P_i N_{i,k} \qquad 13$$

Equation 13 defines the B-spline curve. Consider the above example of basis function and understands the construction of the curve. Expanding the equation 13 for N = 6 (7 control points) and k = 4 (3rd degree).

$$P(u) = P_0 N_{0,4} + P_1 N_{1,4} + P_2 N_{2,4} + P_3 N_{3,4} + P_4 N_{4,4} + P_5 N_{5,4} + P_6 N_{6,4} \qquad 14$$

In above basis function values, each basis function consists of different segments. Putting all the values in equation 14 and separating the curve segment results the following equations.

$$P(u) = P_0\,(1-u)^3 + P_1[u(1-u)^2 + \frac{u(1-u)(2-u)}{2} + \frac{u(2-u)^2}{2}] + P_2[(1-u)u^2 + \frac{(2-u)u^2}{2} + \frac{(3-u)u^2}{6}] + P_3\frac{u^3}{6} \qquad \text{for } 0 \le u < 1$$

$$P(u)_1 = P_1\frac{(2-u)^2}{2} + P_2[\frac{(2-u)u^2}{2} + \frac{u(2-u)(3-u)}{6} + \frac{(u-1)(3-u)^2}{6}] + P_3[\frac{(2-u)u^2}{6} + \frac{u(u-1)(3-u)}{6} + \frac{(4-u)(u-1)^2}{6}] + P_4\frac{(u-1)^3}{6} \qquad \text{for } 1 \le u < 2$$

$$P(u)_2 = P_2\frac{(3-u)^3}{6} + P_3[\frac{(3-u)^3}{6} + \frac{(u-1)(3-u)(4-u)}{6} + \frac{(u-2)(2-u)(4-u)}{6}] + P_4[\frac{(3-u)(u-1)^2}{6} + \frac{(u-2)(u-1)(2-u)}{6} + \frac{(4-u)(u-2)^2}{6}] + P_5\frac{(u-2)^2}{4} \qquad \text{for } 2 \le u < 3$$

$$P(u)_3 = P_3\frac{(2-u)(4-u)^2}{6} + P_4[\frac{(u-2)(4-u)^2}{4} + \frac{(u-1)(2-u)(4-u)}{6} + \frac{(u-3)(4-u)}{4}] + P_5[\frac{(4-u)(u-2)^2}{4} + \frac{(u-4)(u-3)(4-u)}{2} + (4-u)(u-3)^2] + P_6\,(u-3)^3 \qquad \text{for } 3 \le u \le 4$$

As mentioned earlier, b-spline curve is made up of different curve segments. Here, four vectors represent four curve segments. Each segment is controlled by four control points. According to the property, each curve segment is controlled by k number of control points. The algorithm has been developed to generate the clamed B-spline curve. As we have seen in previous basis function example, each basis function consists of several curve segments. First segment 0 to 1 is in four basis functions $N_{0,4}$, $N_{1,4}$, $N_{2,4}$, $N_{3,4}$. Basis functions $N_{1,4}$, $N_{2,4}$, $N_{3,4}$, $N_{4,4}$ consists of 1 to 2 segment. As B-spline curve consists of different segments, in given algorithm, I am calculating basis function one by one, extracting and saving each curve segment values. Figure 3.16 portrays the B-spline curve formulation algorithm.

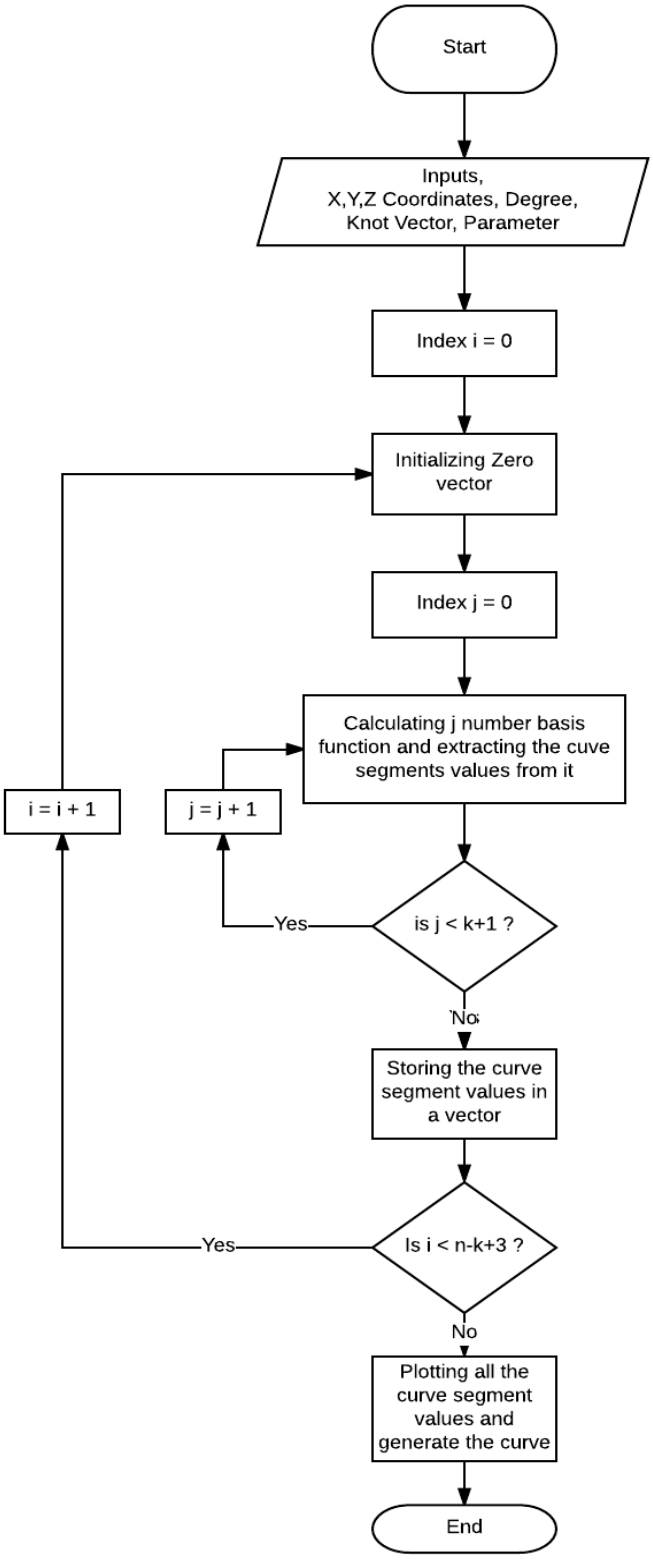


Figure 3.16 B-spline Curve Algorithm

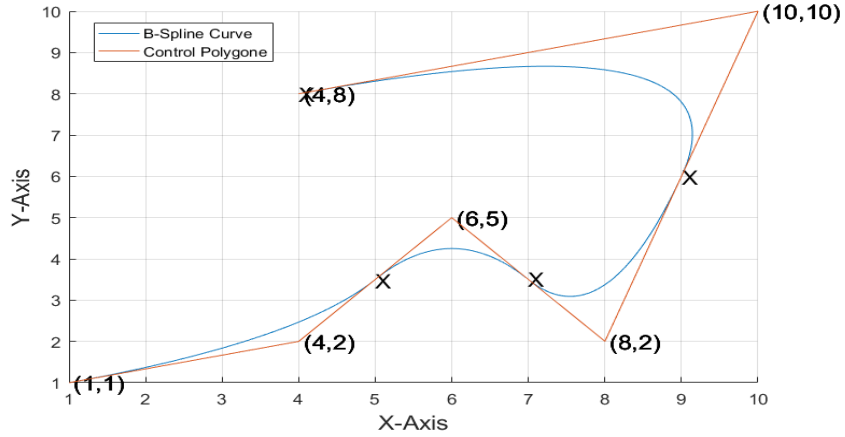


Figure 3.17 clamped B-spline Curve

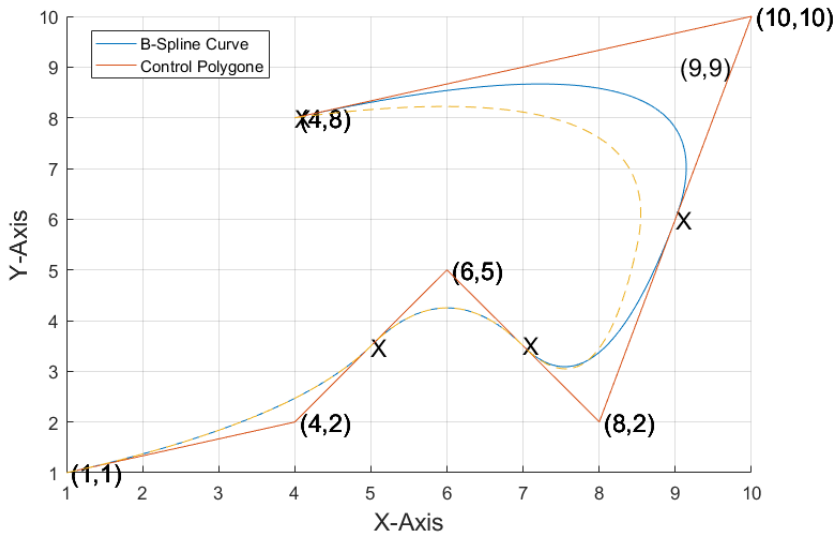


Figure 3.18 Modified Clamped B-spline Curve

The figure 3.17 represents the B-spline curve with 7 control points and 3th order (2rd degree) curve. The presented B-spline curve is clamped curve as the curve touches first and last control points. The cross symbol divide the curve into different segments. Each curve segment is influenced by k number of control points. As we increase the k value, influence also increases. In figure 3.18, control point is changed from (10,10) to (9,9). The dotted yellow line shows the new curve and blue line is the original curve. Local modification can be seen clearly. Some portion of the curve is affected as we move a coordinate. Local modification is a powerful tool of the B-spline curve. The local modification depends upon the order of the curve. By keeping the number of control points and order same we can get Bezier curve from B-spline curve. Figure 3.19 illustrates the result of B-spline to Bezier.

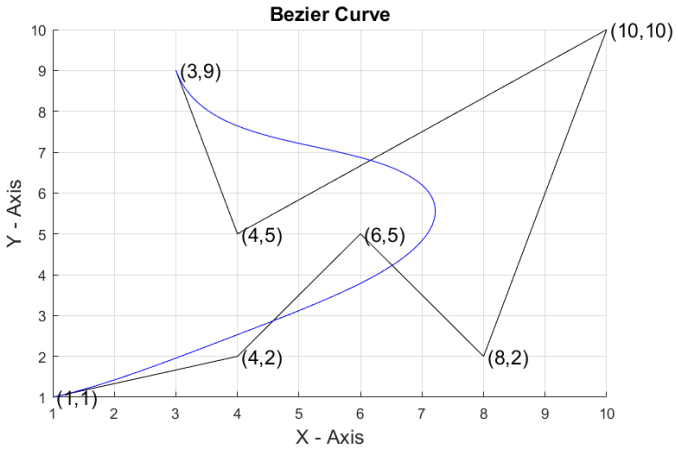


Figure 3.19 B-spline to Bezier curve

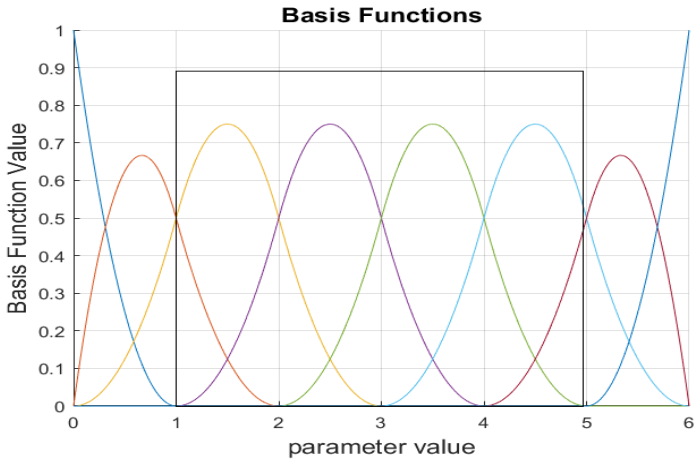


Figure 3.20 Periodic Behavior of the b-spline basis function

The figure is for 3rd order with 8 control points with knot vector = [0 0 0 1 2 3 4 5 6 6 6]. Examine the basis function is the box. The open b-spline curve does not pass through first and last points. The basis function behavior and knot vector is defined in different manner basis functions are presented below.

These basis functions are defined with uniform knot vector = [0 1 2 3 4 5 6 7 8 9 10]. They have similar shape with different position. They are periodic in nature. Open B-spline curve uses periodic basis functions. The figure below shows the basis functions for open curve. The clamped curve has first and last basis function which starts from 1 and goes to 0. In open curve, all the basis functions have identical shape.

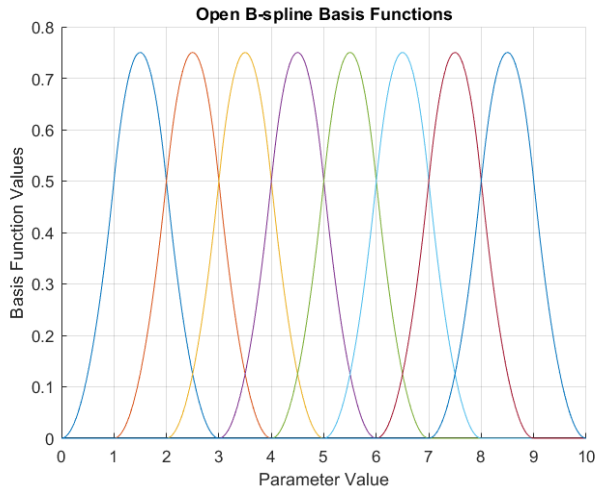


Figure 3.21 Basis functions for open b-spline curve

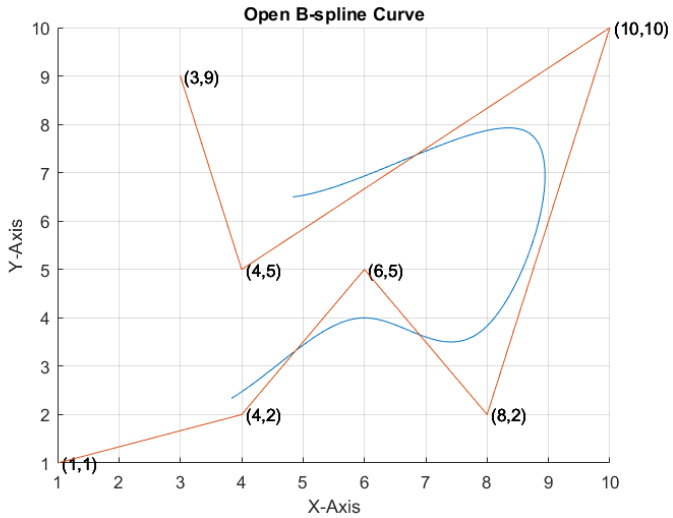
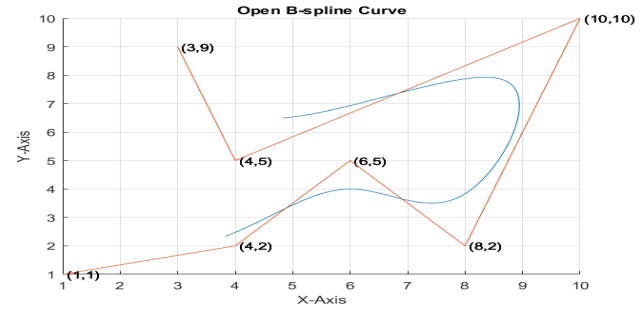


Figure 3.22 Open B-spline Curve

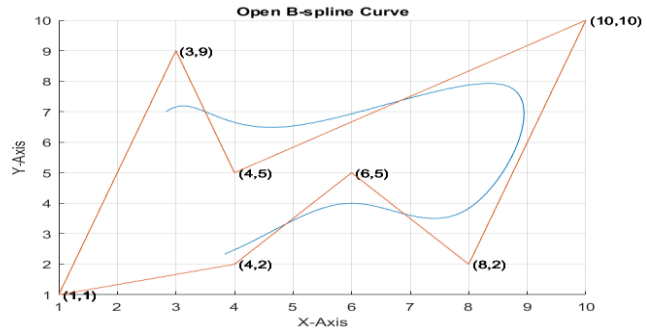
Compare figures 3.21 and 3.20 for open and clamped basis functions; Open b-spline curve uses the similar set of basis functions. The B-spline curve is termed periodic curve since it uses one sort of basis function repetitively.

3.3.1 Closed B-spline Curve



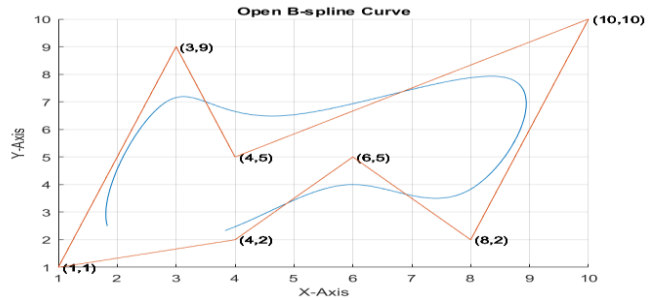

```
X= [ 1 4 6 8 10 4 3]
Y= [ 1 2 5 2 10 5 9]
```

Figure 3.23 a



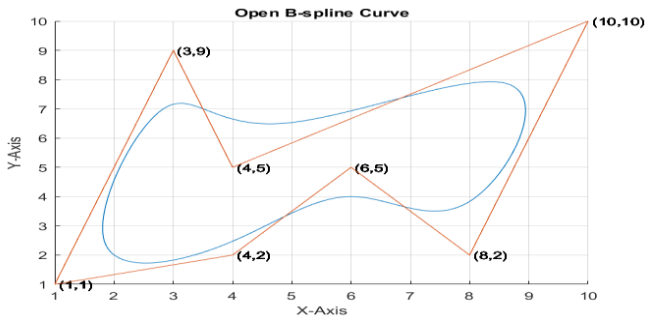

```
X= [ 1 4 6 8 10 4 3 1]
Y= [ 1 2 5 2 10 5 9 1]
```

Figure 3.23 b




```
X= [ 1 4 6 8 10 4 3 1 4]
Y= [ 1 2 5 2 10 5 9 1 2]
```

Figure 3.23 c




```
X= [ 1 4 6 8 10 4 3 1 4 6]
Y= [ 1 2 5 2 10 5 9 1 2 5]
```

Figure 3.23 d

Figure 3.23 Construction of a Closed B-spline curve

It uses the concept of open B-spline basis functions by repeating the control points to generate the closed contour. Figure 3.23 demonstrates the effect of repeating each control point one by one. Reaping each control point creates new curve segment.

3.4 NURBS (Non-Uniform Rational B-spline)

As discussed earlier NURBS is the powerful tool in spline discussion as it is a generalization of B-spline and Bezier Splines. NURBS provides weight for each control point. Here, third variable is added in the definition. [12]

$$P(u) = \begin{matrix} X(u) \\ Y(u) \\ h(u) \end{matrix}$$

$h(u)$ acts as a weight for X and Y control point. Homogeneous co-ordinate system plays an important role in NURBS design. For clarification, I have represented two different curves in figure 3.24. Red curve shows the curve constructed by three variables X, Y and h. Where, Blue curve shows the projection of red curve onto X, Y plane.

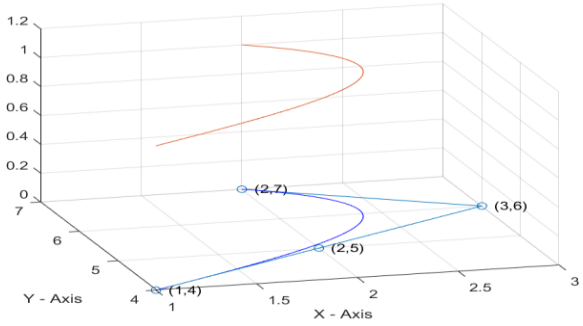


Figure 3.24 B-spline curve in a homogeneous space

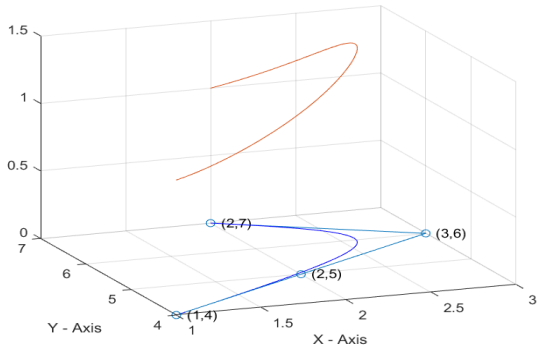


Figure 3.25 Effect of increasing the weight

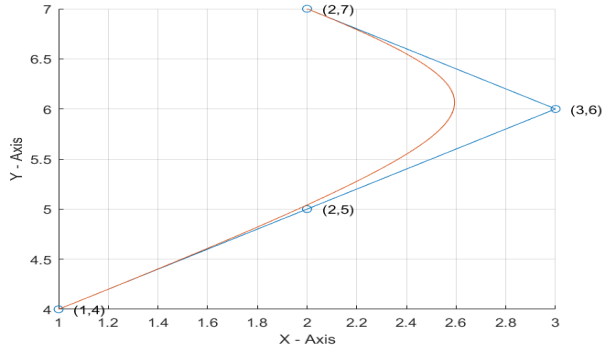


Figure 3.26 projection on X-Y plan

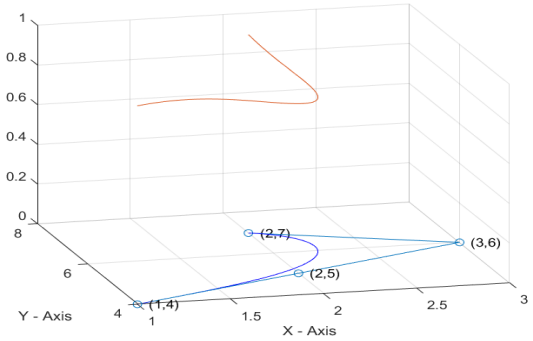


Figure 3.27 Effect of decreasing the weight

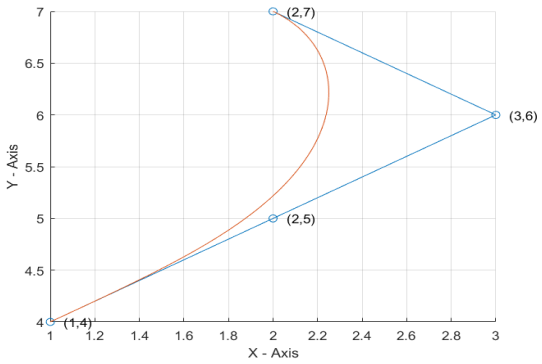


Figure 3.28 projection on X-Y plan

In the figure 3.25, weight of control point (3,6,) has been changed from 1 to 2 and 0.5. The result shows the change in 3D as well as the projection figure 3.26. By keeping co-ordinate fix, if we vary the third parameter h, we can get multiple outcome.

Let's consider clamped B-spline curve example and see the effect by varying the weights. Take control point (10,10) in the figure 3.29. In the illustration 3.30, shows the result of doubling the weight. Next figure 3.31 shows the result for weight = 0.5. So, as we increase the weight, the curve pulls towards the control points. Similarly, Decreasing the weight push the curve from the control point. The effect of push and pull depicted below. Weights must be positive integer of float value; negative weight provides undesirable curve shape. Keeping all weights to one will give B-spline curve.

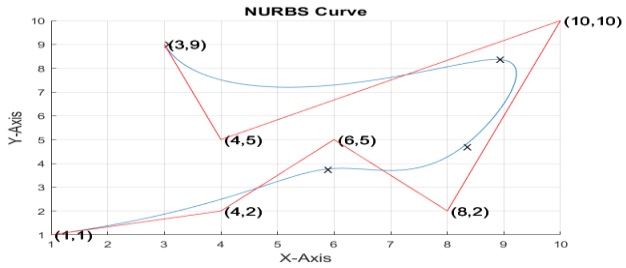


Figure 3.29 NURBS Curve with weightage 2 on the control point (10,10)

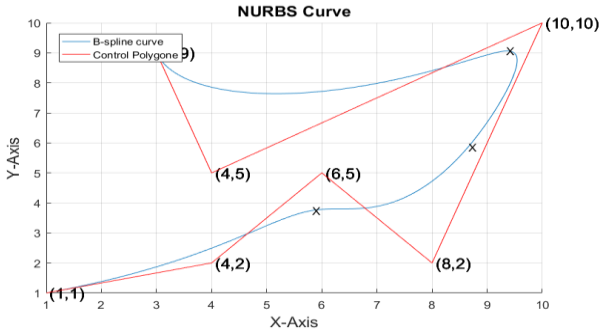


Figure 3.30 NURBS Curve with weightage 4 on the control point (10,10)

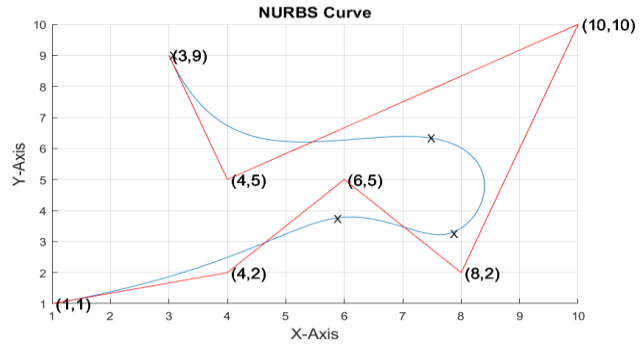


Figure 3.31 NURBS Curve with weightage 0.5 on the control point (10,10)

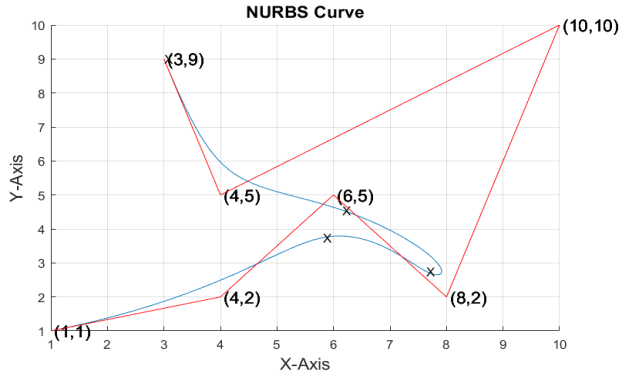


Figure 3.32 NURBS Curve with weightage 0.1 on the control point (10,10)

3.5 B-spline and NURBS Surface

Surface representation is a function of two parametric variables P = P(u,w). u and w are two parametric directions of the surface. Previous sections were concentrated on representation of curves. Understanding the working and properties of curve is important in surface construction.[7]

$$P(u,w) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} N_{i,k}(u) N_{j,L}(w) \qquad 15$$

The equation 15 represents the definition of B-spline surface. $N_{i,k}(u)$ and $N_{i,k}(w)$ are basis function is u and w directions respectively. P_{ij} is input control point matrix. (N+1) and (M+1) are number of control points in two directions. K and L are order in u and w directions respectively. B-spline curve has different curve segments, B-spline surface consists of different patches. Single patch is made up of two curve segments from u and w.

Control point matrix for 6 by 7 control points.

$$P_{ij} = \begin{matrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} & P_{16} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} & P_{26} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} & P_{36} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} & P_{46} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} & P_{56} \end{matrix}$$

Consider an example for n = 5, m = 5, K = 3, L = 3

$$P(u,w) = \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij}N_{i,3}(u)N_{j,3}(w)$$

$$= P_{0,0}N_{0,3}(u)N_{0,3}(w) + P_{0,1}N_{0,3}(u)N_{1,3}(w) + P_{0,2}N_{0,3}(u)N_{2,3}(w) + P_{0,3}N_{0,3}(u)N_{3,3}(w) + P_{0,4}N_{0,3}(u)N_{4,3}(w) + P_{0,5}N_{0,3}(u)N_{5,3}(w) +$$

$$P_{1,0}N_{1,3}(u)N_{0,3}(w) + P_{1,1}N_{1,3}(u)N_{1,3}(w) + P_{1,2}N_{1,3}(u)N_{2,3}(w + P_{1,3}N_{1,3}(u)N_{3,3}(w) + P_{1,4}N_{1,3}(u)N_{4,3}(w) + P_{1,5}N_{1,3}(u)N_{5,3}(w) +$$

$$P_{2,0}N_{2,3}(u)N_{0,3}(w) + P_{2,1}N_{2,3}(u)N_{1,3}(w) + P_{2,2}N_{2,3}(u)N_{2,3}(w) + P_{2,3}N_{2,3}(u)N_{3,3}(w) + P_{2,4}N_{2,3}(u)N_{4,3}(w) + P_{2,5}N_{2,3}(u)N_{5,3}(w) +$$

$$P_{3,0}N_{3,3}(u)N_{0,3}(w) + P_{3,1}N_{3,3}(u)N_{1,3}(w) + P_{3,2}N_{3,3}(u)N_{2,3}(w) + P_{3,3}N_{3,3}(u)N_{3,3}(w) + P_{3,4}N_{3,3}(u)N_{4,3}(w) + P_{3,5}N_{3,3}(u)N_{5,3}(w) +$$

$$P_{4,0}N_{4,3}(u)N_{0,3}(w) + P_{4,1}N_{4,3}(u)N_{1,3}(w) + P_{4,2}N_{4,3}(u)N_{2,3}(w) + P_{4,3}N_{4,3}(u)N_{3,3}(w) + P_{4,4}N_{4,3}(u)N_{4,3}(w) + P_{4,5}N_{4,3}(u)N_{5,3}(w) +$$

$$P_{5,0}N_{5,3}(u)N_{0,3}(w) + P_{5,1}N_{5,3}(u)N_{1,3}(w) + P_{5,2}N_{5,3}(u)N_{2,3}(w) + P_{5,3}N_{5,3}(u)N_{3,3}(w) + P_{5,4}N_{5,3}(u)N_{4,3}(w) + P_{5,5}N_{5,3}(u)N_{5,3}(w)$$

$$= N_{0,3}(u)[P_{0,0}N_{0,3}(w) + P_{0,1}N_{1,3}(w) + P_{0,2}N_{2,3}(w) + P_{0,3}N_{3,3}(w) + P_{0,4}N_{4,3}(w) + P_{0,5}N_{5,3}(w)\ ] +$$

$$N_{1,3}(u)[P_{1,0}N_{0,3}(w) + P_{1,1}N_{1,3}(w) + P_{1,2}N_{2,3}(w) + P_{1,3}N_{3,3}(w) + P_{1,4}N_{4,3}(w) + P_{1,5}N_{5,3}(w)\ ] +$$

$$N_{2,3}(u)[P_{2,0}N_{0,3}(w) + P_{2,1}N_{1,3}(w) + P_{2,2}N_{2,3}(w) + P_{2,3}N_{3,3}(w) + P_{2,4}N_{4,3}(w) + P_{2,5}N_{5,3}(w)\ ] +$$

$$N_{3,3}(u)[P_{3,0}N_{0,3}(w) + P_{3,1}N_{1,3}(w) + P_{3,2}N_{2,3}(w) + P_{3,3}N_{3,3}(w) + P_{3,4}N_{4,3}(w) + P_{3,5}N_{5,3}(w)\ ] +$$

$$N_{4,3}(u)[P_{4,0}N_{0,3}(w) + P_{4,1}N_{1,3}(w) + P_{4,2}N_{2,3}(w) + P_{4,3}N_{3,3}(w) + P_{4,4}N_{4,3}(w) + P_{4,5}N_{5,3}(w)\ ] +$$

$$N_{5,3}(u)[P_{5,0}N_{0,3}(w) + P_{5,1}N_{1,3}(w) + P_{5,2}N_{2,3}(w) + P_{5,3}N_{3,3}(w) + P_{5,4}N_{4,3}(w) + P_{5,5}N_{5,3}(w)\ ]$$

The extension gives idea about the behavior of the surface. First curve is created in w parameter direction using control points and extended in another direction u. It is a recursive process and double iteration, one for raw and one for column, extract each control point. The algorithm has been developed. The flow chart below represents the working of the algorithm. The algorithm requires control point matrix, order, knot vector and parameters. Two separate loops are running, one for u direction and other for w direction. Index starts from 0 to number of segments (n-k+2) or (m-l+2). Extracting each segment from both basis function and creating a patch one by one and finally merging all patches to create the final surface. The patch concept illustrated in the figures from 3.35 to 3.37. The result is shown in figure 3.34.

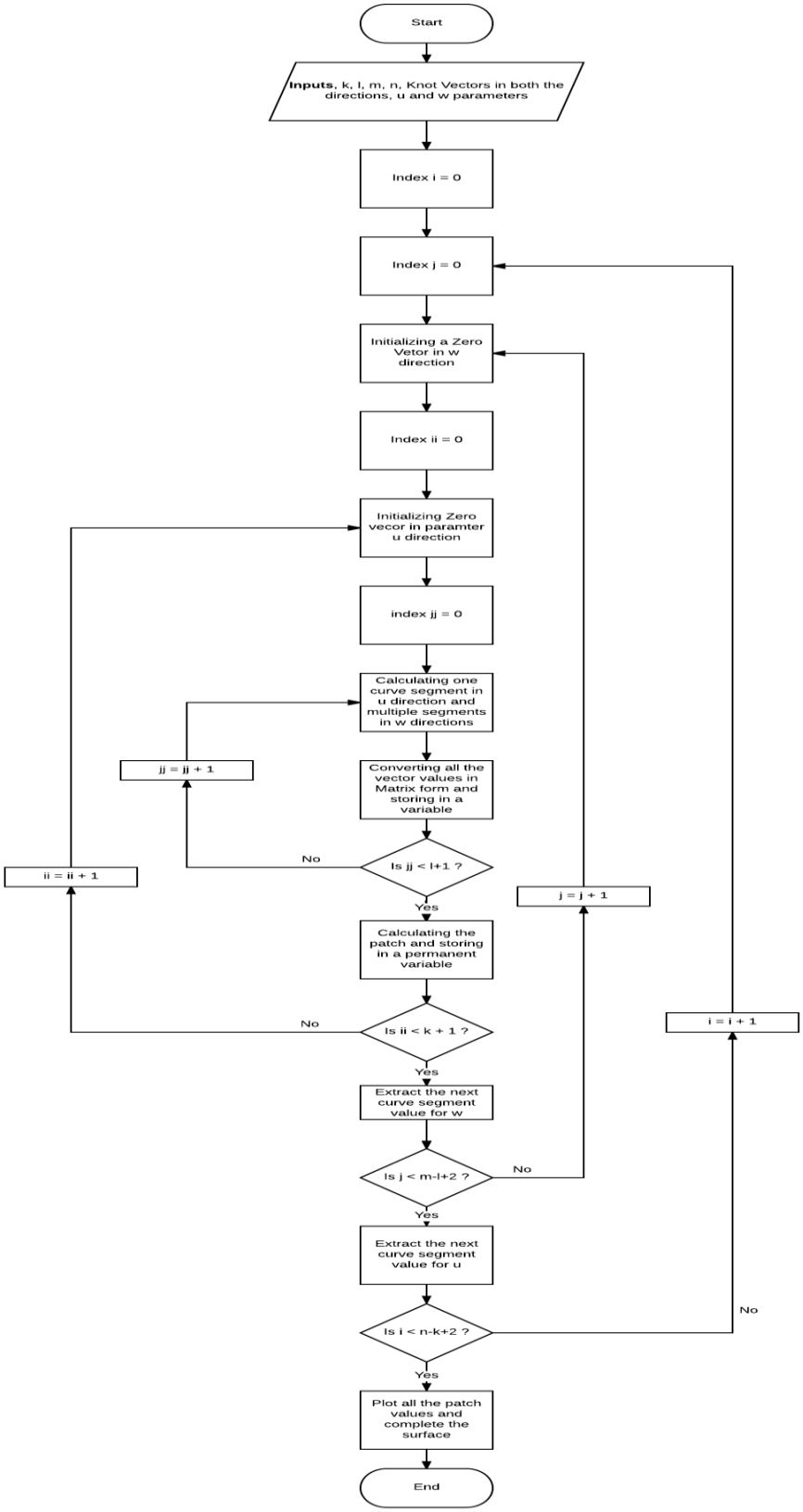


Figure 3.33 Algorithm for Surface construction

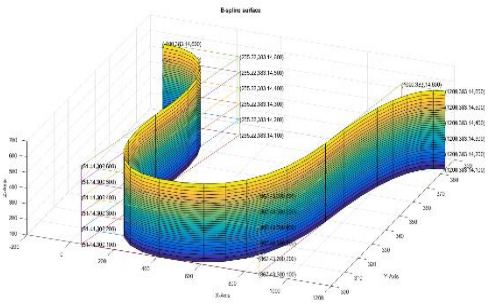


Figure 3.34 B-spline Surface

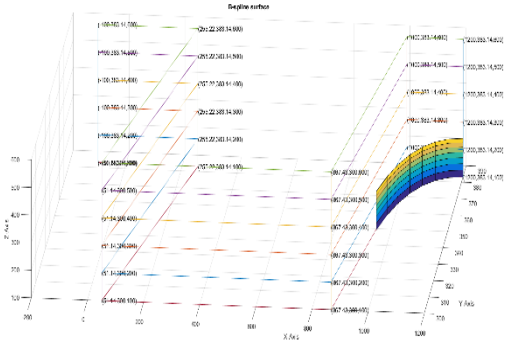


Figure 3.35 B-spline Surface single patch

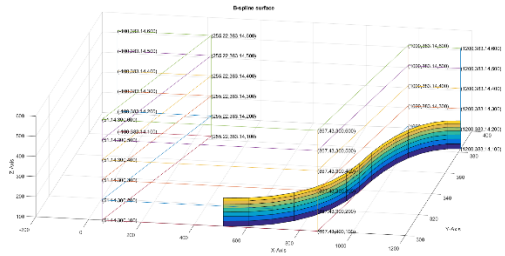


Figure 3.36 B-spline Surface two patches

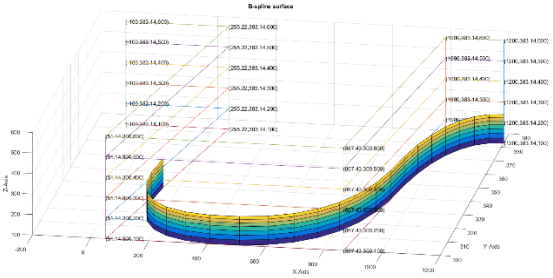


Figure 3.37 B-spline Surface three patches

The figure 3.34 displays the complete B-spline surface defined by 20 control points. The figure 3.35 shows the single patch. The figure 3.36 illustrates the two-combined patched and the following figure 3.37 is constructed by three patches. The 3D Polygon is referred as a control net.

3.5.1 NURBS Surface

$$P(u,w) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} h_{ij}P_{ij}N_{i,k}(u)N_{j,L}(w)}{\sum_{i=0}^{n}\sum_{j=0}^{m} h_{ij}N_{i,k}(u)N_{j,L}(w)} \qquad 16$$

NURBS Surface uses weightages as an additional degree of freedome than B-spline Surfaces. Weightages given to the coordinates for further modification. The figure 3.38 and 3.39 illustrates the effect of weightages on the previous model. [3]

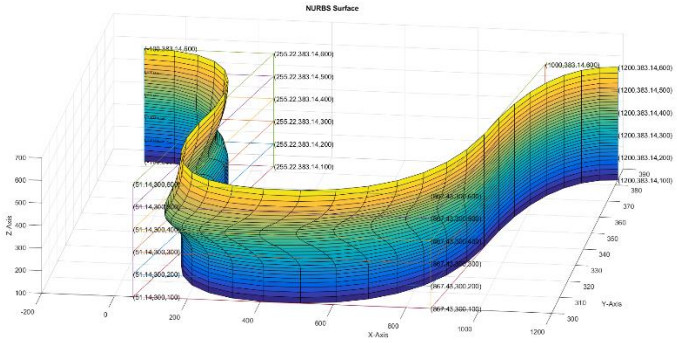


Figure 3.38 NURBS Surface with weightage 2 on (51.14,300,400) control point

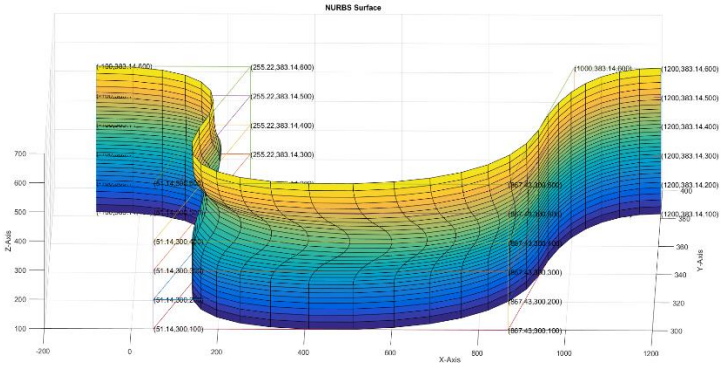


Figure 3.39 NURBS Surface with weightage 0.5 on (51.14,300,400) control point

Here, weightage of Coordinate (51.14,300,400) has been changed from 1 to 2 in the figure 3.38 which represents the effect of pull. In the next fiure 3.39 depicts push effect with weightage of 0.5.

3.6 STEP File data structure

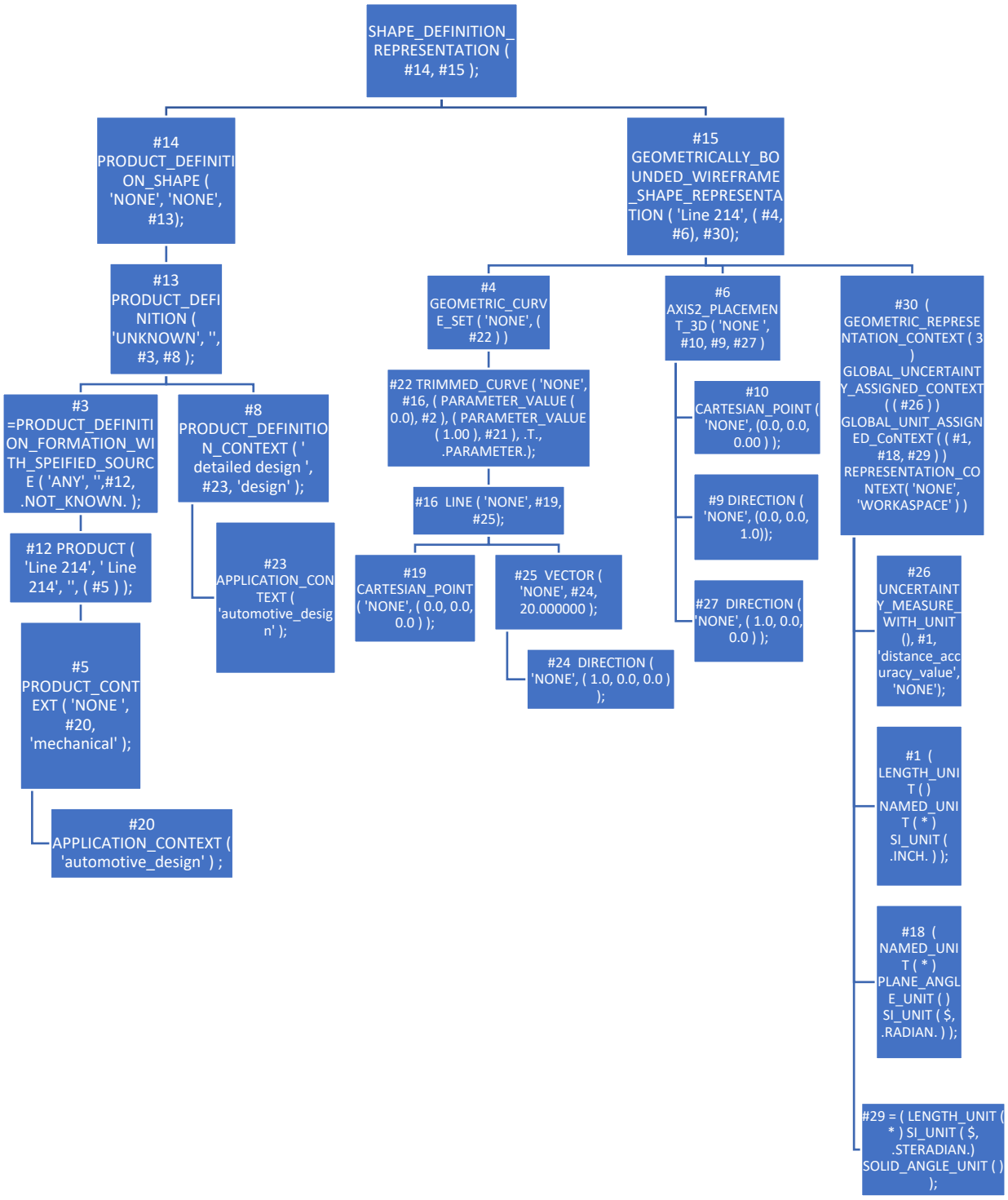
STEP Standard has been adopted for data exchange between MATLAB and other CAD Software. STEP has ASCII file structure. Therefore, working with ASCII Structure is easy. STEP file is using different entities defined in AP 214 to capture the geometric definition. First, simple geometric entities have been tested to understand the logic of STEP File. The following image shows the STEP File data structure for line geometry.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (( 'STEP AP214' ),'1' );
FILE_NAME ('Line 214.STEP','2016-07-28T20:06:28',( '' ),( '' ),'SwSTEP 2.0','Solidworks 2015','' );
FILE_SCHEMA (( 'AUTOMOTIVE_DESIGN' ));
ENDSEC;

DATA;

#7 = SHAPE_DEFINITION_REPRESENTATION ( #14, #15 );
#11 = PRODUCT_RELATED_PRODUCT_CATEGORY ( 'part', '' ,( #12 ) ) ;
#17 = APPLICATION_PROTOCOL_DEFINITION ( 'draft international standard', 'automotive_design', 1998, #23 );
#28 = APPLICATION_PROTOCOL_DEFINITION ( 'draft international standard', 'automotive_design', 1998, #20);
#14 = PRODUCT_DEFINITION_SHAPE ( 'NONE', 'NONE', #13);
#13 = PRODUCT_DEFINITION ( 'UNKNOWN', '', #3, #8 );
#3 = PRODUCT_DEFINITION_FORMATION_WITH_SPEIFIED_SOURCE ( 'ANY', '',#12, .NOT_KNOWN. );
#12 = PRODUCT ( 'Line 214', ' Line 214', '', ( #5 ) );
#5 = PRODUCT_CONTEXT ( 'NONE ', #20, 'mechanical' );
#20 = APPLICATION_CONTEXT ( 'automotive_design' ) ;
#8 = PRODUCT_DEFINITION_CONTEXT ( ' detailed design ', #23, 'design' );
#23 = APPLICATION_CONTEXT ( 'automotive_design' );
#15 = GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION ( 'Line 214', ( #4, #6), #30);
#4 = GEOMETRIC_CURVE_SET ( 'NONE', ( #22 ) ) ;
#22 = TRIMMED_CURVE ( 'NONE', #16, ( PARAMETER_VALUE ( 0.000000000000000000), #2 ), ( PARAMETER_VALUE ( 1.000000000000000000 ), #21 ), .T., .PARAMETER.);
#16 = LINE ( 'NONE', #19, #25);
#19 = CARTESIAN_POINT ( 'NONE', ( 0.000000000000000000, 0.000000000000000000, 0.000000000000000000 ) );
#25 = VECTOR ( 'NONE', #24, 20.000000 );
#24 = DIRECTION ( 'NONE', ( 1.000000000000000000, 0.000000000000000000, 0.000000000000000000 ) );
#2 = CARTESIAN_POINT ( 'NONE', ( 0.000000000000000000, 0.000000000000000000, 0.000000000000000000 ) );
#21 = CARTESIAN_POINT ( 'NONE', ( 20.000000, 0.000000000000000000, 0.000000000000000000 ) );
#6 = AXIS2_PLACEMENT_3D ( 'NONE ', #10, #9, #27 );
#10 = CARTESIAN_POINT ( 'NONE', (0.000000000000000000, 0.000000000000000000, 0.000000000000000000 ) );
#9 = DIRECTION ( 'NONE', (0.000000000000000000, 0.000000000000000000, 1.000000000000000000));
#27 = DIRECTION ( 'NONE', ( 1.000000000000000000, 0.000000000000000000, 0.000000000000000000 ) );
#30 = ( GEOMETRIC_REPRESENTATION_CONTEXT ( 3 ) GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT ( ( #26 ) ) GLOBAL_UNIT_ASSIGNED_CONTEXT ( ( #1, #18, #29 ) ) REPRESENTATION_CONTEXT( 'NONE', 'WORKASPACE' ) );
#26 = UNCERTAINTY_MEASURE_WITH_UNIT (LENGTH_MEASURE( 1.00000000000000100E-005), #1, 'distance_accuracy_value', 'NONE');
#1 = ( LENGTH_UNIT ( ) NAMED_UNIT ( * ) SI_UNIT ( .INCH. ) );
#18 = ( NAMED_UNIT ( * ) PLANE_ANGLE_UNIT ( ) SI_UNIT ( $, .RADIAN. ) );
#29 = ( LENGTH_UNIT ( * ) SI_UNIT ( $, .STERADIAN.) SOLID_ANGLE_UNIT ( ) );

ENDSEC;
END-ISO-10303-21;
```

Figure 3.40 Data structure of the STEP File

To understand correctly, the following flow chart has been developed on page 48.

Different entities have the different role to play. For example, AXIS2_PLACEMENT_3 Entity represents the 3 D Coordinate systems, GEOMETRIC_CURVE_SET shows the geometry line with magnitude and directions, PRODUCT_DEFINITION_SHAPE used for defining Application protocol and the name of the file. Similarly, B-spline Surface has B_SPLINE_SURFACE_WITH_KNOTS. Inputs are required to adequately represent surface. The algorithm has been developed to write out STEP File from MATLAB by first calculating all the necessary data and converting into strings. Each entity has its inputs, restrictions and internal mapping.


SHAPE_DEFINITION_REPRESENTATION (#14, #15);
#14 PRODUCT_DEFINITION_SHAPE ('NONE', 'NONE', #13);
#15 GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION ('Line 214', (#4, #6), #30);
#13 PRODUCT_DEFINITION ('UNKNOWN', '', #3, #8);
#4 GEOMETRIC_CURVE_SET ('NONE', (#22))
#6 AXIS2_PLACEMENT_3D ('NONE ', #10, #9, #27)
#30 (GEOMETRIC_REPRESENTATION_CONTEXT (3) GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT ((#26)) GLOBAL_UNIT_ASSIGNED_CoNTEXT ((#1, #18, #29)) REPRESENTATION_CONTEXT('NONE', 'WORKASPACE'))
#3 =PRODUCT_DEFINITION_FORMATION_WITH_SPEIFIED_SOURCE ('ANY', '',#12, .NOT_KNOWN.);
#8 PRODUCT_DEFINITION_CONTEXT (' detailed design ', #23, 'design');
#22 TRIMMED_CURVE ('NONE', #16, (PARAMETER_VALUE (0.0), #2), (PARAMETER_VALUE (1.00), #21), .T., .PARAMETER.);
#10 CARTESIAN_POINT ('NONE', (0.0, 0.0, 0.00));
#9 DIRECTION ('NONE', (0.0, 0.0, 1.0));
#27 DIRECTION ('NONE', (1.0, 0.0, 0.0));
#12 PRODUCT ('Line 214', ' Line 214', '', (#5));
#23 APPLICATION_CONTEXT ('automotive_design');
#16 LINE ('NONE', #19, #25);
#19 CARTESIAN_POINT ('NONE', (0.0, 0.0, 0.0));
#25 VECTOR ('NONE', #24, 20.000000);
#24 DIRECTION ('NONE', (1.0, 0.0, 0.0));
#26 UNCERTAINTY_MEASURE_WITH_UNIT (), #1, 'distance_accuracy_value', 'NONE');
#5 PRODUCT_CONTEXT ('NONE ', #20, 'mechanical');
#1 (LENGTH_UNIT ( ) NAMED_UNIT (*) SI_UNIT (.INCH.));
#18 (NAMED_UNIT (*) PLANE_ANGLE_UNIT ( ) SI_UNIT ($, .RADIAN.));
#20 APPLICATION_CONTEXT ('automotive_design') ;
#29 = (LENGTH_UNIT (*) SI_UNIT ($, .STERADIAN.) SOLID_ANGLE_UNIT ( ));

ENTITY b_spline_curve_with_knots

```
(* SCHEMA step_merged_ap_schema; *)

ENTITY b_spline_curve_with_knots
   SUBTYPE OF (b_spline_curve);
      knot_multiplicities : LIST [2:?] OF INTEGER;
      knots : LIST [2:?] OF parameter_value;
      knot_spec : knot_type;
   DERIVE
      upper_index_on_knots : INTEGER := SIZEOF(knots);
   WHERE
      wr1:
         constraints_param_b_spline(degree, upper_index_on_knots, upper_index_on_control_points, knot_multiplicities, knots);
      wr2:
         SIZEOF(knot_multiplicities) = upper_index_on_knots;
END_ENTITY;
```

Explicit Attributes

Entity **b_spline_curve_with_knots** *has the following local and inherited explicit attributes:*

Attribute	Type	Defined By
name	label (STRING)	representation_item
degree	INTEGER	b_spline_curve
control_points_list	LIST OF cartesian_point (ENTITY)	b_spline_curve
curve_form	b_spline_curve_form (ENUM)	b_spline_curve
closed_curve	LOGICAL	b_spline_curve
self_intersect	LOGICAL	b_spline_curve
knot_multiplicities	LIST OF INTEGER	b_spline_curve_with_knots
knots	LIST OF parameter_value (REAL)	b_spline_curve_with_knots
knot_spec	knot_type (ENUM)	b_spline_curve_with_knots

Figure 3.41 B-spline curve entity [9]

The entity for b spline curve is shown in the figure 3.41. It shows certain attributes with different types.

```
#26 = GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION ( 'Spline.stp', ( #21, #9 ), #13 ) ;
#27 = B_SPLINE_CURVE_WITH_KNOTS ( 'NONE', 2 ,(#28,#29,#30,#31,#32,#33),.UNSPECIFIED., .F., .F.,(3,1,1,1,3),(0,0.25,0.5,0.75,1),.UNSPECIFIED. ) ;
#28 = CARTESIAN_POINT ( 'NONE',( 1.0000000000000000000, 1.0000000000000000000, 0.0000000000000000000) ) ;
#29 = CARTESIAN_POINT ( 'NONE',( 4.0000000000000000000, 2.0000000000000000000, 0.0000000000000000000) ) ;
#30 = CARTESIAN_POINT ( 'NONE',( 6.0000000000000000000, 5.0000000000000000000, 0.0000000000000000000) ) ;
#31 = CARTESIAN_POINT ( 'NONE',( 8.0000000000000000000, 2.0000000000000000000, 0.0000000000000000000) ) ;
#32 = CARTESIAN_POINT ( 'NONE',( 10.000000000000000000, 10.000000000000000000, 0.0000000000000000000) ) ;
#33 = CARTESIAN_POINT ( 'NONE',( 4.0000000000000000000, 5.0000000000000000000, 0.0000000000000000000) ) ;
```

The screenshot taken from spline STEP File. The curve is made up of 6 control points with 2nd degree (3rd order). The curve is clamped and not self-intersecting. Knot vector has value from 0 to 1 with k order multiplicity. The input must be accurate to run the function. Otherwise it will give an error. The similar way for surface representation, the entity called b_spliine_surface_with_knots used.

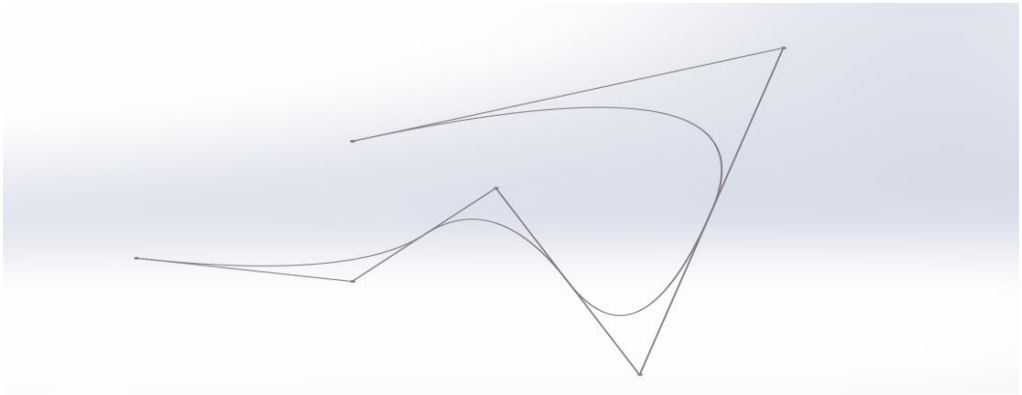


Figure 3.42 Result imported in the SolidWorks for curve

The above step file developed in MATLAB and imported in the SolidWorks as depicted in the figure 3.42. Control polygon is optional in this case.

ENTITY b_spline_surface_with_knots

```
(* SCHEMA step_merged_ap_schema; *)

ENTITY b_spline_surface_with_knots
   SUBTYPE OF (b_spline_surface);
      u_multiplicities : LIST [2:?] OF INTEGER;
      v_multiplicities : LIST [2:?] OF INTEGER;
      u_knots : LIST [2:?] OF parameter_value;
      v_knots : LIST [2:?] OF parameter_value;
      knot_spec : knot_type;
   DERIVE
      knot_u_upper : INTEGER := SIZEOF(u_knots);
      knot_v_upper : INTEGER := SIZEOF(v_knots);
   WHERE
      wr1:
         constraints_param_b_spline(SELF\b_spline_surface.u_degree, knot_u_upper, SELF\b_spline_surface.u_upper, u_multiplicities, u_knots);
      wr2:
         constraints_param_b_spline(SELF\b_spline_surface.v_degree, knot_v_upper, SELF\b_spline_surface.v_upper, v_multiplicities, v_knots);
      WR3:
         SIZEOF(u_multiplicities) = knot_u_upper;
      WR4:
         SIZEOF(v_multiplicities) = knot_v_upper;
END_ENTITY;
```

Explicit Attributes

Entity **b_spline_surface_with_knots** *has the following local and inherited explicit attributes:*

Attribute	Type	Defined By
name	label (STRING)	representation_item
u_degree	INTEGER	b_spline_surface
v_degree	INTEGER	b_spline_surface
control_points_list	LIST OF LIST OF cartesian_point (ENTITY)	b_spline_surface
surface_form	b_spline_surface_form (ENUM)	b_spline_surface
u_closed	LOGICAL	b_spline_surface
v_closed	LOGICAL	b_spline_surface
self_intersect	LOGICAL	b_spline_surface
u_multiplicities	LIST OF INTEGER	b_spline_surface_with_knots
v_multiplicities	LIST OF INTEGER	b_spline_surface_with_knots
u_knots	LIST OF parameter_value (REAL)	b_spline_surface_with_knots
v_knots	LIST OF parameter_value (REAL)	b_spline_surface_with_knots
knot_spec	knot_type (ENUM)	b_spline_surface_with_knots

Figure 3.43 B-spline surface entity [9]

The following figure3.44 and 3.45 represents the output from MATLAB and STEP Files correspondingly.

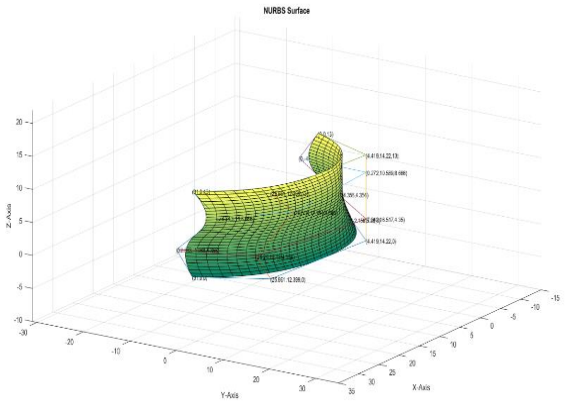


Figure 3.44 B-spline Surface

Figure 3.45 B-spline STEP File

Chapter 4 Results

Algorithm requires control point information to create a surface. Sample block has been used to recreate the surface generated in inspire software. Figure 2.11 shows the topology optimization result. Sample cross sections have been generated using PolyNURBS tools as shows in figure 4.1. Control points have been used in the algorithms to recreate the surface as shown in the figure 4.2

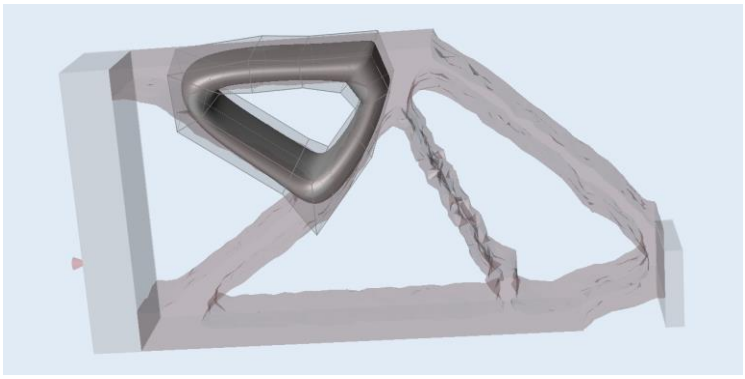


Figure 4.1 Closed polyNURBS fit

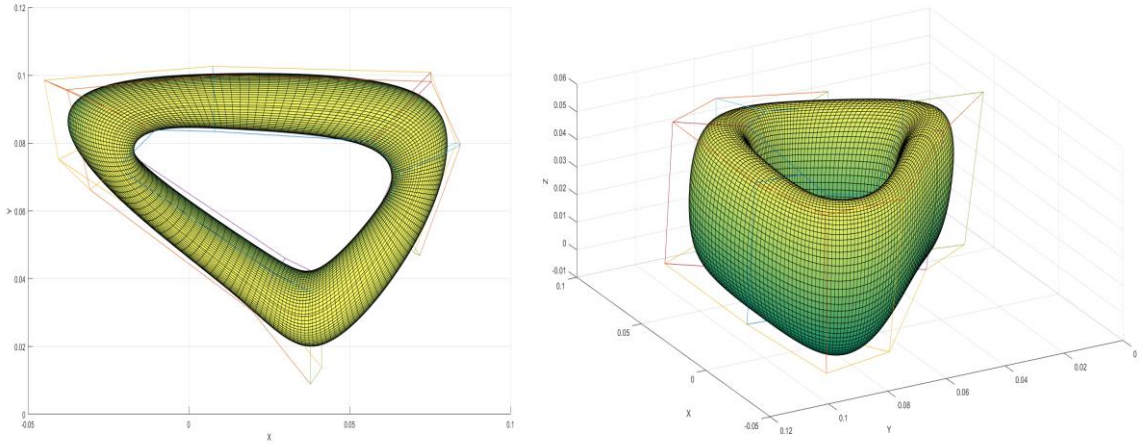


Figure 4.2 MATLAB result for closed contour

Designer can vary the degree in u, w or both the direction, control points, and weightages to modfy the shape. The figure 4.3 shows the similar contour for 2nd order in u and w directions. Algorithms can be find in Appendix. Figure 4.5 represents the result for partially closed surface. Algoeihm is capable of producing complicated geometries with ease of handeling.

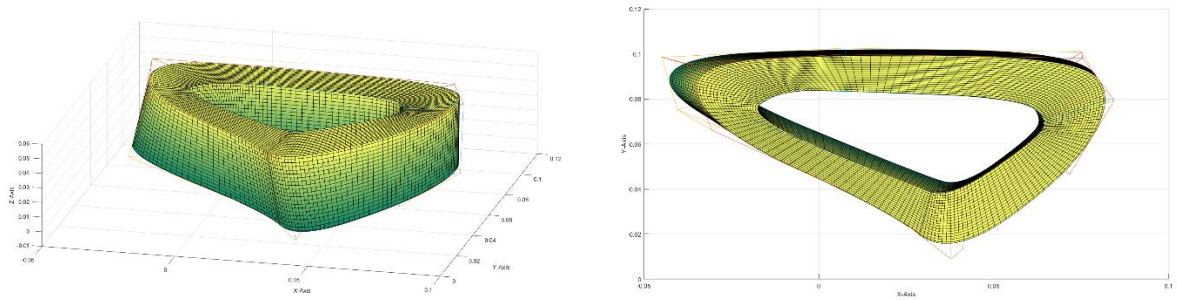


Figure 4.3 MATLAB surface result for 2nd order

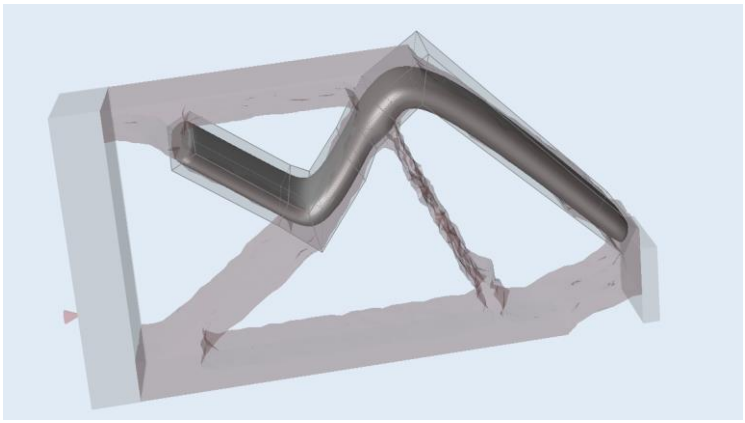


Figure 4.4 Open cross section fit using PolyNURBS tool

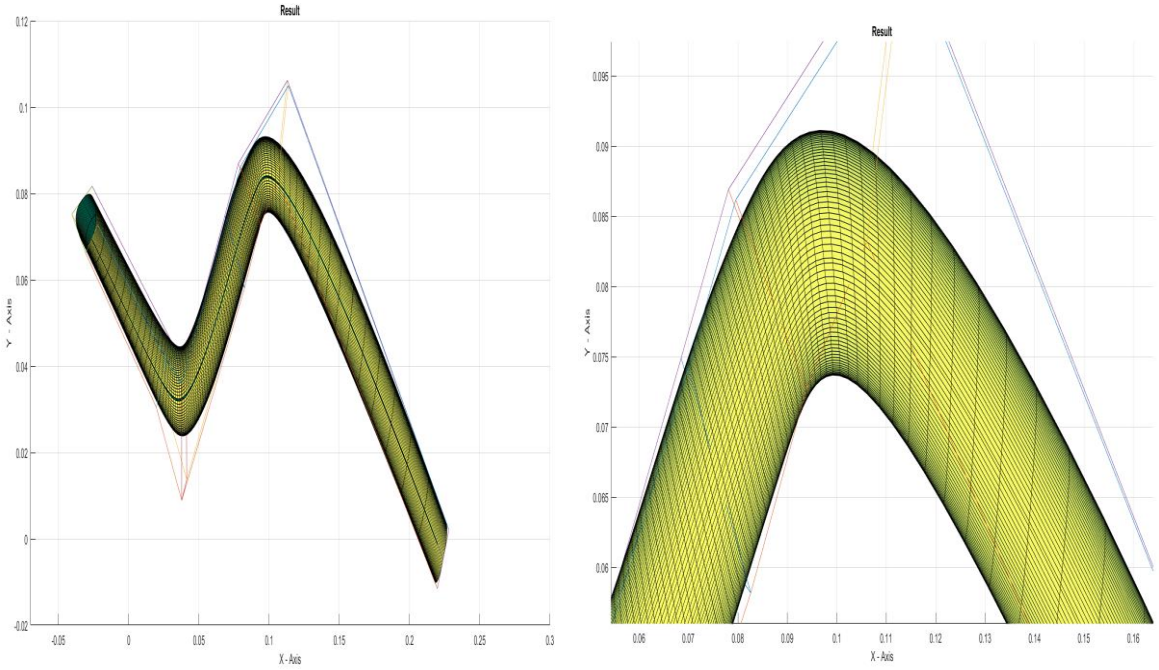


Figure 4.5 MATLAB result for partially closed contour

Chapter 5 Conclusion

In this research work, different types of free form curves and surfaces have been studied and built framework in the MATLAB to support the creation of various types of surfaces. Designer has various degrees of freedom to modify the NURBS Surface. Some information is required to run the Algorithm like control points, knot vector and degree. One can easily change the shape of the surface by varying knot vector, dragging the control points. Detailed discussion of the different types of curves and surfaces have been discussed with examples.

STEP AP 214 standard for data exchange has been studied and implemented. Data structure has been discussed along with examples to illustrate the working of STEP file. STEP file is compatible in CAD and CAE software. Therefore, it is easy to use the generated geometry from MATAB in CAD Software.

Chapter 6 Future work

The main aim of the research work is to automate the process of creation of NURBS surfaces on rough topology. It requires intersection, smooth blending of two surfaces, Boolean operation to complete the task. I have created basic building block. Further operations can be performed using these algorithms to achieve the task.

In some parametric CAD Software, STEP file does not preserve the control points of the NURBS surface. To overcome this limitation, CAD software provides their API to create specific application. One can use these APIs to create an application to create an NURBS surface to retain the control points.

Appendix A Bezier Curve code

BezierCurve.m

```
clear;clc;
% input coordinate values
X = [1 2 3 4 5 6 ] ;
Y = [4 2 5 1 2 3];
% Parameter
u = 0:0.01 :1 ;
% Number of control points
n = numel(X) ;
% Initial Values
X_vector= 0;
Y_vector = 0;

for i = 0 : n-1
% Calculating Basis Function
B = basis(i,n-1) ;
% Calculating X and Y vectors
X_vector = X_vector + B.*X(i+1);
Y_vector = Y_vector + B.*Y(i+1);
grid on

end

hold on
% plotting X and Y component vector
plot(X,Y,'k')
plot(X_vector,Y_vector,'b')
```

```
title('Bezier Curve','fontsize',14)

xlabel(' X - Axis ','fontsize',14)

ylabel(' Y - Axis ','fontsize',14)

for i = 1: numel(X)

txt = ['(',num2str(X(i)),',',num2str(Y(i)),')'];

text(X(i)+0.06,Y(i),txt,'fontsize',14)

end

grid on

legend(' Control Polygon ',' Bezier Curve ')
```

basis.m

```
function [B] = basis(i,n)
%Parameter
u = 0: 0.01:1;
% Calculating Polynomial
C = combination(n,i);

U = u.^i ;
Ui = (1-u).^(n-i) ;
B = U.*C.*Ui ;
end
```

combination.m

```
function [out] = combination(n,i)

X = factorial (n);
Y = factorial (i);
Z = factorial (n-i);

out = X./(Y*Z);

end
```

Appendix B B-spline Basis Function code

BasisFunction.m

```
% N Represents the basis funtion final value
function [N] = BasisFunction(ii , t , k , u , n)
% initializing Zero vector for basis function values
N = zeros( n+1 , length(u) );

for K = 2 : k

    if   K == 2
        % Calculating Basis function values for 2nd order
        for i = ii : ( ii + k - 2 )

            FirstOrder = Primary_basis( i , t , u, K-1 ) ;
            Numerator = ( u - t ( i + 1 ) ) ;
            Denominator = t ( i + K ) - t ( i + 1 ) ;

            if Denominator ~= 0
            N( i + 1 , : ) = FirstOrder.*( Numerator./ Denominator ) ;
            end

            FirstOrder = Primary_basis( i + 1 , t , u , K-1 ) ;
            Numerator = ( t( i + K + 1 ) - u ) ;
            Denominator = t( i + K + 1 ) - t( i + 2 ) ;

            if Denominator ~= 0
 N( i + 1 , :) = N( i+1 , :)+FirstOrder.*( Numerator./ Denominator ) ;
            end
        end
    else
         % Rest of the basis function calculations
          for i = ii:(ii + k - K)
              Numerator =( u - t( i + 1 ) ) ;
              Denominator = t( i + K ) - t( i + 1 ) ;

            if Denominator ~= 0
           N( i + 1 ,:) = N( i + 1 ,:).*( Numerator./ Denominator ) ;
            end

              Numerator = (t( i + K + 1 ) - u ) ;
              Denominator = t( i + K + 1 ) - t( i + 2 ) ;

            if Denominator ~= 0
       N( i+1 ,:)=N( i+1 ,:) + N( i+2 ,:).*(Numerator./ Denominator);
            end
          end
    end
end
end
```

Primary_basis.m

```
function [ First_BasisFunc ] = Primary_basis( i , t , u, K )
% Using Indexing concept in parameter u to calculate the first order
values
if i < t ( end - K + 1 )
First_BasisFunc = ( u >= t( i + 1 ) & u < t( i + 2 ) ) ;
else
First_BasisFunc = ( u >= t( i + 1 ) & u <= t( i + 2 ) ) ;
end
```

Appendix C Clamped B-spline Curve code

B_Spline_Cuve_Clamped.m

```
clear;clc;
% Input Coordinates
X=[ 1 4 6 8 9 4 3];
Y=[ 1 2 5 2 9 5 9];
% Input Order
k = 4;

% Number of control points
n = length(X) - 1 ;
% initializing Variables
ij = 1;
K = k - 1;

Spacing = 500;
T = zeros(1,n+k+1) ;
% knot vector
for i = 0 : n + k

    if ( i < k )

        T( i + 1 ) = 0;

    elseif ( i >= k && i <= n )

        T( i + 1 ) = i - k + 1;

    else

        T( i + 1 ) = n - k + 2;

    end

end
% Adjusting spacing
```

```
while 1

        if rem( Spacing, T( end ) ) == 0
            break
        end
          Spacing = Spacing + 1;
end
% Parameter range
u = linspace(T( k ) , T( n + 2 ) , Spacing) ;
hold all
% Initializaing loop from 0 to number of curve segments
for i = 0: n - k + 1

    Temp = BasisFunction (i , T , k , u , n) ;

    Ax =  Temp ( i + 1 , Primary_basis ( K , T , u , 1 ) )  ;
   % initializing zero vector for X and Y Component
    X_Vector_Segment = zeros( 1 , numel( Ax ) ) ;
    Y_Vector_Segment = zeros( 1 , numel( Ax ) ) ;
  % Extracting curve segments from basis function
    for ii = i : i + k - 1

    BasisFunc_Value = BasisFunction ( ii , T , k , u , n ) ;
  % Calculating the value for a curve segment
    X_Vector_Temp = BasisFunc_Value ( ii + 1 ,Primary_basis( K , T , u
, 1 ) ) .* X( ii + 1 ) ;
    Y_Vector_Temp = BasisFunc_Value ( ii + 1 ,Primary_basis( K , T , u
, 1 ) ) .* Y( ii + 1 ) ;

    X_Vector_Segment = X_Vector_Segment + X_Vector_Temp ;
    Y_Vector_Segment = Y_Vector_Segment + Y_Vector_Temp ;

    end
    K = K + 1 ;
    xlabel('X-Axis')
    ylabel('Y-Axis')
   % Storing curve segments in a single vector
    for i = 1 : Spacing/T(end)

        X_Vector( ij ) = X_Vector_Segment( i ) ;
        Y_Vector( ij ) = Y_Vector_Segment( i ) ;
        ij = ij + 1 ;

    end
    % plotting control points
    for i = 1: numel(X)

        txt = ['(',num2str(X(i)),',',num2str(Y(i)),')'];
        text(X(i)+0.06,Y(i),txt)

    end
end
```

```
 K = k - 1 ;
 % Plotting the curve
 hold on
 plot(X_Vector,Y_Vector)
 % marking curve segments in a plot
 for i = 0 : n - k + 1

 Index_X = X_Vector( 1 , Primary_basis( K , T , u , 1 ) ) ;
 Index_Y = Y_Vector( 1 , Primary_basis( K , T , u , 1 ) ) ;
 Ix = Index_X( end ) ;
 Iy = Index_Y( end ) ;
 text( Ix , Iy , 'X' )
 K = K + 1 ;

 end
 % Control polygone
 plot(X,Y,'r')
 legend('Bezier curve','Control Polygone','Location','northwest')
 grid on
title('B-spline Curve')
```

Appendix D Closed B-spline Curve code

Bspline_Closed_Curve.m

```
clear;clc;
% Input Coordinates
X=[ 1 4 6 8 9 4 3];
Y=[ 1 2 5 2 9 5 9];
% Input order
k = 4 ;
% Number of control points
n = length(X) - 1 ;
% Initializing Variables
ij = 1 ;
K = k - 1 ;

Spacing = 5000;
% Defining Knot Vector
t =0 : n + k ;
% Adjusting Spacing
while 1
        if rem( Spacing,n+k ) == 0
            break
        end
          Spacing = Spacing + 1;
end
% Parameter u
 u = linspace( t( 1 ) , t( end ) , Spacing ) ;
```

```
hold all

for i = 0 : n
 % Defining a constraint for closed curve for repeat control points
 if ( i == n - k + 2 )

     K = k - 1 ;

 end

    Temp = BasisFunction( i , t , k , u , n ) ;

    Segment_Size = ( Temp ( i + 1 , Primary_basis( K , t , u , k ) ));
     % initializing zero vector for X and Y Component
    X_Vector_Segment = zeros( 1 , numel( Segment_Size ) ) ;
    Y_Vector_Segment = zeros( 1 , numel( Segment_Size ) ) ;
    % Extracting Curve segments
    for ii = i : i + k - 1
    % constraint  for repeating control point
         if ( i <= n - k + 1 )
    jz = ii ;
         else
    jz = ii- ( n - k + 2 ) ;
         end

         if ii >= n + 1
             ii = ii + 1 ;
         end

    BasisFunc_Value = BasisFunction( jz , t , k , u , n ) ;
     % Calculating the value for a curve segment
    X_Vector_Temp = ( BasisFunc_Value( jz + 1 , Primary_basis( K , t ,
u , k ) ) ).*X( mod( ii + 1 , n + 2 ) ) ;
    Y_Vector_Temp = ( BasisFunc_Value( jz + 1 , Primary_basis( K , t ,
u , k ) ) ).*Y( mod( ii + 1 , n + 2 ) ) ;

    X_Vector_Segment = X_Vector_Segment + X_Vector_Temp ;
    Y_Vector_Segment = Y_Vector_Segment + Y_Vector_Temp ;

    end

    K = K + 1 ;

    xlabel('X-Axis')
    ylabel('Y-Axis')
    % Storing curve segments in a single vector
    for i = 1 : length( X_Vector_Segment )

        X_Vector( ij ) = X_Vector_Segment( i ) ;
        Y_Vector( ij ) = Y_Vector_Segment( i ) ;
        ij = ij + 1 ;
```

```
    end
     % plotting control points
     for i = 1: numel(X)

        txt = ['(',num2str(X(i)),',',num2str(Y(i)),')'];
        text(X(i)+0.06,Y(i),txt)

    end
end
hold all
% Plotting the curve
 plot(X_Vector,Y_Vector)
% Plotting Control polygon
 plot(X,Y)
 grid on
 title(' Closed B-spline Curve','FontSize',14)
```

Appendix E Open B-spline Curve code

Bspline_Open_Curve.m

```
 % Input Coordinates
X=[ 1 4 6 8 10 4 3 ];
Y=[ 1 2 5 2 10 5 9 ];
% Input Order
k = 4;
% Number of control points
n = length(X) - 1 ;
% initializing Variables
ij = 1 ;
K = k - 1 ;
Spacing = 500;
% knot vector
T =0 : n+k ;
% Adjusting spacing
while 1

        if rem( Spacing,n+k ) == 0

            break
        end

          Spacing = Spacing + 1 ;
end
% Parameter range
u = linspace( T( 1) , T(end) , Spacing ) ;
hold all
N = zeros( n+1 , length(u),k );
```

```
for i = 0 : n - k + 1

    Temp = BasisFunction( i , T , k , u , n ) ;

    Ax =  Temp ( i + 1 , Primary_basis( K , T , u , 1 ) )  ;
   % initializing zero vector for X and Y Component
    X_Vector_Segment = zeros( 1 , numel( Ax ) ) ;
    Y_Vector_Segment = zeros( 1 , numel( Ax ) ) ;
  % Extracting curve segments from basis function
    for ii = i : i + k - 1

    BasisFunc_Value = BasisFunction ( ii , T , k , u , n ) ;
  % Calculating the value for a curve segment
    X_Vector_Temp = BasisFunc_Value ( ii + 1 , Primary_basis( K , T ,
u , 1 ) ) .* X( ii + 1 ) ;
    Y_Vector_Temp = BasisFunc_Value ( ii + 1 , Primary_basis( K , T ,
u , 1 ) ) .* Y( ii + 1 ) ;

    X_Vector_Segment = X_Vector_Segment + X_Vector_Temp ;
    Y_Vector_Segment = Y_Vector_Segment + Y_Vector_Temp ;

    end

    K = K + 1 ;
    xlabel('X-Axis')
    ylabel('Y-Axis')
   % Storing curve segments in a single vector
    for i = 1 : Spacing/T(end)

        X_Vector( ij ) = X_Vector_Segment( i ) ;
        Y_Vector( ij ) = Y_Vector_Segment( i ) ;
        ij = ij + 1 ;

    end
    % plotting control points
    for i = 1: numel(X)

        txt = ['(',num2str(X(i)),',',num2str(Y(i)),')'];
        text(X(i)+0.06,Y(i),txt)

    end

end
 K = k - 1 ;
 % Plotting the curve
 hold on
 plot(X_Vector,Y_Vector)
 % Control polygone
 plot(X,Y,'r')
 legend('Bezier curve','Control Polygone','Location','northwest')
 grid on
title('B-spline Curve')
```

Appendix F NURBS Curve code

NURBS_Curve.m

```
clear;clc;
% Input Coordinates
X=[ 1 4 6 8 10 4 3];
Y=[ 1 2 5 2 10 5 9];
% Input Weightages
W =[1 1 1 1 1 1 1];
% Input Order
k = 4;
% Number of control points
n = length(X) - 1 ;
% initializing Variables
ij = 1 ;
K = k - 1 ;

Spacing = 500;
T = zeros(1,n+k+1) ;
% knot vector
for i = 0 : n + k
    if ( i < k )
        T( i + 1 ) = 0;
    elseif ( i >= k && i <= n )
        T( i + 1 ) = i - k + 1;
    else
        T( i + 1 ) = n - k + 2;
    end
end
% Adjusting spacing
while 1

        if rem( Spacing, T( end ) ) == 0

            break
        end

          Spacing = Spacing + 1 ;
end
% Parameter range
u = linspace( T( k ) , T( n + 2 ) , Spacing ) ;
hold all
% Initializaing loop from 0 to number of curve segments
for i = 0 : n - k + 1

    Temp = BasisFunction ( i , T , k , u , n ) ;

    Segment_Size =  Temp ( i + 1 , Primary_basis ( K , T , u , 1 ));
    % initializing zero vector for X and Y Component
    X_Vector_Segment = zeros( 1 , numel( Segment_Size ) ) ;
```

```
    Y_Vector_Segment = zeros( 1 , numel( Segment_Size ) ) ;
    Weight = zeros( 1 , numel( Segment_Size ) );
    % Extracting curve segments from basis function
    for ii = i : i + k - 1

    BasisFunc_Value = BasisFunction ( ii , T , k , u , n ) ;
     % Calculating the value for a curve segment
    X_Vector_Temp = BasisFunc_Value ( ii + 1 , Primary_basis( K , T ,
u , 1 ) ) .* X( ii + 1 ).*W(ii+1) ;
    Y_Vector_Temp = BasisFunc_Value ( ii + 1 , Primary_basis( K , T ,
u , 1 ) ) .* Y( ii + 1 ).*W(ii+1) ;
      % Calculating the Weight value for a curve segment
    Weight1 =  BasisFunc_Value ( ii + 1 , Primary_basis( K , T , u , 1
) ).*W(ii+1);
    X_Vector_Segment = X_Vector_Segment + X_Vector_Temp ;
    Y_Vector_Segment = Y_Vector_Segment + Y_Vector_Temp ;
    Weight = Weight + Weight1 ;
    end

    K = K + 1 ;

    xlabel('X-Axis','fontsize',14)

    ylabel('Y-Axis','fontsize',14)
%    %  rationalizing
    X_Vector_Segment_rational = X_Vector_Segment./Weight ;
    Y_Vector_Segment_rational = Y_Vector_Segment./Weight ;

    for i = 1 : Spacing/T(end)

        X_Vector_rational( ij ) = X_Vector_Segment_rational( i ) ;
        Y_Vector_rational( ij ) = Y_Vector_Segment_rational( i ) ;
        ij = ij + 1 ;

    end

    for i = 1: numel(X)

        txt = ['(',num2str(X(i)),',',num2str(Y(i)),')'];
        text(X(i)+0.06,Y(i),txt,'fontsize',14)

    end

end

 K = k - 1 ;
 hold on
 plot( X_Vector_rational,Y_Vector_rational)

 for i = 0 : n - k + 1

 Index_X = X_Vector_rational( 1 , Primary_basis( K , T , u , 1 ) ) ;
```

```
 Index_Y = Y_Vector_rational( 1 , Primary_basis( K , T , u , 1 ) ) ;
 Ix = Index_X( end ) ;
 Iy = Index_Y( end ) ;
 text( Ix , Iy , 'X' )
 K = K + 1 ;

 end
 plot(X,Y,'r')
 legend('B-spline curve','Control Polygone','Location','northwest')
 grid on
title('NURBS Curve','fontsize',14)
```

Appendix G Closed NURBS Surface code

NURBS_Closed_Surface.m

```
clear;clc
% Input X, Y,and Z Coordinates
X=[0.061698 0.071774 0.070062 0.059986
    0.065628 0.084467 0.083066 0.064227
    0.057123 0.075364 0.075205 0.056964
    0.0080743 0.0075549 0.0074432 0.0079626
    -0.010412 -0.037753 -0.044874 -0.017533
    -0.021948 -0.0306409 -0.040444 -0.025983
    0.026669 0.019722 0.022996 0.029943
    0.037719 0.037811 0.04136 0.04094
  ];

Y=[0.061738 0.046845 0.047594 0.062487
    0.070868 0.079404 0.080115 0.071579
    0.080518 0.098145 0.10084 0.083217
    0.083421 0.099943 0.10264 0.08612
    0.083868 0.095738 0.098536 0.08666
    0.072696 0.066214 0.075214 0.081697
    0.041549 0.030444 0.03492 0.046025
    0.036079 0.0089147 0.013787 0.040951
    ];

Z=[0.050461 0.049951 0.00043675 0.00094618
    0.050416 0.050713 0.0021967 0.0018999
    0.049938 0.049571 -0.0010824 -0.0007154
    0.049866 0.05076 0.0001538 -0.00074082
    0.049152 0.053339 -0.00095205 -0.005139
    0.05 0.05 0 0
    0.050865 0.049426 -0.00093744 0.00050112
    0.054991 0.05253 -0.0028496 -0.00038885
   ];
Weightages=[ 1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
```

```
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1 ];

% input order in u direction
k=4;
% input order in w direction
l=3;
Size = size(X) ;
% Number of control points in u direction
n=Size(1)-1;
% Number of Control points in w direction
m=Size(2)-1;

% Knot vector in w direction
T = linspace(0 , 1 , m + l + 1 ) ;
% Knot vector in u direction
t = linspace(0 , 1 , n + k + 1 ) ;

USpacing = 200;
% Adjusting Spacing in u and w directions
while 1
        if rem(USpacing,n + k)==0
            break
        end
          USpacing=USpacing+1;
end

WSpacing=(m+l).*(USpacing./(k+n));
% Parameter value for u
u = linspace( 0, 1 , USpacing ) ;
% Parameter value for w
w = linspace(0,1,WSpacing) ;

L = l - 1;
K = k - 1 ;

for i=0 :n
    % Defining a constraint in u direction  to repeat the control
points
     if ( i == n - k + 2 )

     K = k - 1 ;
     end
    Colume=1;

    L=l-1;
    % Defining a constraint in w direction  to repeat the control
points
    for j=0  : m
```

```
    if ( j == m - l + 2 )

     L = l - 1 ;

    end
    % Calculating the size of column in a patch
    Patch_u_dimension =(USpacing./(n+k));
    % initializing zero vector in a u direction segment

Xu_Vector_Segment=zeros(1,Patch_u_dimension);Yu_Vector_Segment=zeros(1
,Patch_u_dimension);Zu_Vector_Segment=zeros(1,Patch_u_dimension);
    % Converting vector into matrix form

[Xu_Vector_Segment]=meshgrid(Xu_Vector_Segment);[Yu_Vector_Segment]=me
shgrid(Yu_Vector_Segment);[Zu_Vector_Segment]=meshgrid(Zu_Vector_Segme
nt);
     Weight1=zeros(1,Patch_u_dimension);[Weight1]=meshgrid(Weight1);

    for ii = i :i + k -1
    % constraint  for repeating the control point in u direction
         if ( i <= n - k + 1 )
    jz = ii ;
         else
    jz = ii- ( n - k + 2 ) ;
         end

         if ii >= n + 1
             ii = ii + 1 ;
         end

    % Calculating the size of raw in a patch
    Patch_w_dimension=(WSpacing./(m+l));
    % initializing zero vector in a w direction segment

Xw_Vector_Segment=zeros(1,Patch_w_dimension);Yw_Vector_Segment=zeros(1
,Patch_w_dimension);Zw_Vector_Segment=zeros(1,Patch_w_dimension);
    % Converting vector into matrix form

[Xw_Vector_Segment]=meshgrid(Xw_Vector_Segment);[Yw_Vector_Segment]=me
shgrid(Yw_Vector_Segment);[Zw_Vector_Segment]=meshgrid(Zw_Vector_Segme
nt);
     Weight=zeros(1,Patch_w_dimension);[Weight]=meshgrid(Weight);

    for jj = j:j+l-1
    % constraint  for repeating the control point in w direction
         if ( j <= m - l + 1 )
    zj = jj ;
         else
    zj = jj - ( m - l + 2 ) ;
         end
```

```
        if jj >= m + 1
            jj = jj + 1 ;
        end
    % Calculating values of basisfunctions in u direction
    BasisFunc_u_direction=BasisFunction(jz,t,k,u,n);

    % Extracting Curve segments in u direction
    Ua=(BasisFunc_u_direction(jz+1,Primary_basis(K,t,u,1)));
    Ub=(BasisFunc_u_direction(jz+1,Primary_basis(K,t,u,1)));
    Uc=(BasisFunc_u_direction(jz+1,Primary_basis(K,t,u,1)));
    Uw=(BasisFunc_u_direction(jz+1,Primary_basis(K,t,u,1)));

    % Calculating values of basisfunctions in w direction
    BasisFunc_w_direction=BasisFunction(zj,T,l,w,m);

    % Extracting Curve segments in w direction
    Wx=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*X(mod( ii
+ 1 , n + 2 ), mod( jj + 1 , m + 2 ));
    Wy=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*Y(mod( ii
+ 1 , n + 2 ), mod( jj + 1 , m + 2 ));
    Wz=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*Z(mod( ii
+ 1 , n + 2 ), mod( jj + 1 , m + 2 ));
    W
=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*Weightages(mod(
ii + 1 , n + 2 ), mod( jj + 1 , m + 2 ));

    [Ua,Wx]=meshgrid(Ua,Wx);
    [Ub,Wy]=meshgrid(Ub,Wy);
    [Uc,Wz]=meshgrid(Uc,Wz);
    [Uw,W]=meshgrid(Uw,W);

Xw_Vector_Segment=Xw_Vector_Segment+Wx;Yw_Vector_Segment=Yw_Vector_Seg
ment+Wy;Zw_Vector_Segment=Zw_Vector_Segment+Wz;
    Weight=Weight+W;

    end
    Weight1=Weight1+Weight.*Uw;
    % Forming a patch

Xu_Vector_Segment=(Xu_Vector_Segment+Ua.*Xw_Vector_Segment);Yu_Vector_
Segment=(Yu_Vector_Segment+Ub.*Yw_Vector_Segment);Zu_Vector_Segment=(Z
u_Vector_Segment+Uc.*Zw_Vector_Segment);
    end
    % Rationalizing the X,Y, and Z vectors

Xu_Vector_Segment_Rational=Xu_Vector_Segment./Weight1;Yu_Vector_Segmen
t_Rational=Yu_Vector_Segment./Weight1;Zu_Vector_Segment_Rational=Zu_Ve
ctor_Segment./Weight1;
    % extracting and Storing the patch values
    for I=1:length(Xu_Vector_Segment)
        if i==0
            Raw=1;
```

```
        else
            Raw=i*length(Xu_Vector_Segment)+1;
        end
    for kk=1:length(Xu_Vector_Segment)
        Xu_Vector(Raw,Colume)=Xu_Vector_Segment_Rational(I,kk);
        Yu_Vector(Raw,Colume)=Yu_Vector_Segment_Rational(I,kk);
        Zu_Vector(Raw,Colume)=Zu_Vector_Segment_Rational(I,kk);
        Raw=Raw+1;

    end
      Colume=Colume+1;
    end

    L=L+1;
    end
    K=K+1;

end

hold all

TEMP = size(Xu_Vector)  ;
% Closing the curve by repeating the first raw and column values
for i = 1 : TEMP(1)
    Xu_Vector(i,TEMP(2)+1) = Xu_Vector(i,1) ;
    Yu_Vector(i,TEMP(2)+1) = Yu_Vector(i,1) ;
    Zu_Vector(i,TEMP(2)+1) = Zu_Vector(i,1) ;
end
TEMP = size(Xu_Vector)  ;
for i = 1 : TEMP(2)
    Xu_Vector(TEMP(1)+1,i) = Xu_Vector(1,i) ;
    Yu_Vector(TEMP(1)+1,i) = Yu_Vector(1,i) ;
    Zu_Vector(TEMP(1)+1,i) = Zu_Vector(1,i) ;
end
% Plotting the closed surface
 surf(Xu_Vector,Yu_Vector,Zu_Vector)
colormap summer
% Plotting the control net
plot3(X,Y,Z)
plot3(X',Y',Z')
grid on
xlabel('X-Axis')
ylabel('Y-Axis')
zlabel('Z-Axis')
```

Appendix H Partially Closed NURBS Surface code

NURBS_Clamped_Closed_Surface.m

```
clear;clc
% Input X,Y,and Z Coordinates and weightage
X=[0.061698 0.071774 0.070062 0.059986
    0.065628 0.084467 0.083066 0.064227
    0.057123 0.075364 0.075205 0.056964
    0.0080743 0.0075549 0.0074432 0.0079626
    -0.010412 -0.037753 -0.044874 -0.017533
    -0.021948 -0.0306409 -0.040444 -0.025983
    0.026669 0.019722 0.022996 0.029943
    0.037719 0.037811 0.04136 0.04094
   0.061698 0.071774 0.070062 0.059986];

Y=[0.061738 0.046845 0.047594 0.062487
    0.070868 0.079404 0.080115 0.071579
    0.080518 0.098145 0.10084 0.083217
    0.083421 0.099943 0.10264 0.08612
    0.083868 0.095738 0.098536 0.08666
    0.072696 0.066214 0.075214 0.081697
    0.041549 0.030444 0.03492 0.046025
    0.036079 0.0089147 0.013787 0.040951
    0.061738 0.046845 0.047594 0.062487];

Z=[0.050461 0.049951 0.00043675 0.00094618
    0.050416 0.050713 0.0021967 0.0018999
    0.049938 0.049571 -0.0010824 -0.0007154
    0.049866 0.05076 0.0001538 -0.00074082
    0.049152 0.053339 -0.00095205 -0.005139
    0.05 0.05 0 0
    0.050865 0.049426 -0.00093744 0.00050112
    0.054991 0.05253 -0.0028496 -0.00038885
    0.050461 0.049951 0.00043675 0.00094618];

Weightages=[ 1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1
             1 1 1 1 ];

Size = size(X) ;
% Number of raws
n=Size(1)-1;
% Number of Columns
m=Size(2)-1;
```

```
% input order in u direction
k=3;
% input order in w direction
l=3;

K = k - 1 ;
L = l - 1;
USpacing = 100;
% Knot vector in u direction
for j=1:n+k+1
    if ((j-1)<k)
        t(j) = 0;
    elseif ((j-1)>=k && (j-1)<=n)
        t(j) = (j-1) - k + 1;
    else
        t(j) = n - k + 2;
    end
end

% Adjusting Spacing in u and w directions
while 1
        if rem(USpacing,t(end-k+1))==0
            break
        end
          USpacing=USpacing+1;
end

% Knot vector in w direction
T = linspace(0 , 1 , m + l + 1 )  ;
WSpacing=(m+l).*(USpacing./t(end-k+1));
% Parameter value for w
w = linspace(0,1,WSpacing) ;
% Parameter value for u
u = linspace(t(k),t(end-k+1),USpacing);

for i=0 :n-k+1
    Colume=1;
    L=l-1;

    for j= 0  : m
    % Defining a constraint in w direction  to repeat the control
points
    if ( j == m - l + 2 )
     L = l - 1 ;

    end

    % Calculating the size of column in a patch
    Patch_u_dimension=(USpacing./(n-k+2));
    % initializing zero vector in a u direction segment

Xu_Vector_Segment=zeros(1,Patch_u_dimension);Yu_Vector_Segment_Rationa
```

```
l=zeros(1,Patch_u_dimension);Zu_Vector_Segment=zeros(1,Patch_u_dimensi
on);
    % Converting vector into matrix form

[Xu_Vector_Segment]=meshgrid(Xu_Vector_Segment);[Yu_Vector_Segment_Rat
ional]=meshgrid(Yu_Vector_Segment_Rational);[Zu_Vector_Segment]=meshgr
id(Zu_Vector_Segment);
    Weight1=zeros(1,Patch_u_dimension);[Weight1]=meshgrid(Weight1);

    for ii = i :i + k -1
    Patch_w_dimension=(WSpacing./(m+l));

Xw_Vector_Segment=zeros(1,Patch_w_dimension);Yw_Vector_Segment=zeros(1
,Patch_w_dimension);Zw_Vector_Segment=zeros(1,Patch_w_dimension);

[Xw_Vector_Segment]=meshgrid(Xw_Vector_Segment);[Yw_Vector_Segment]=me
shgrid(Yw_Vector_Segment);[Zw_Vector_Segment]=meshgrid(Zw_Vector_Segme
nt);
    Weight=zeros(1,Patch_w_dimension);[Weight]=meshgrid(Weight);

    for jj = j:j+l-1
    % constraint  for repeating the control point in w direction
         if ( j <= m - l + 1 )
    zj = jj ;
         else
    zj = jj - ( m - l + 2 ) ;
         end

         if jj >= m + 1
             jj = jj + 1 ;
         end

    % Calculating values of basisfunctions in u direction
    BasisFunc_u_direction=BasisFunction(ii,t,k,u,n);
    % Extracting Curve segments in u direction
    Ua=(BasisFunc_u_direction(ii+1,Primary_basis(K,t,u,1)));
    Ub=(BasisFunc_u_direction(ii+1,Primary_basis(K,t,u,1)));
    Uc=(BasisFunc_u_direction(ii+1,Primary_basis(K,t,u,1)));
    Uw=(BasisFunc_u_direction(ii+1,Primary_basis(K,t,u,1)));

    % Calculating values of basisfunctions in w direction
    BasisFunc_w_direction=BasisFunction(zj,T,l,w,m);

    Wx=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*X(mod( ii
+ 1 , n + 2 ), mod( jj + 1 , m + 2 ));
    Wy=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*Y(mod( ii
+ 1 , n + 2 ), mod( jj + 1 , m + 2 ));
    Wz=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*Z(mod( ii
+ 1 , n + 2 ), mod( jj + 1 , m + 2 ));
    W
=(BasisFunc_w_direction(zj+1,Primary_basis(L,T,w,1))).*Weightages(mod(
ii + 1 , n + 2 ), mod( jj + 1 , m + 2 ));
```

```
    [Ua,Wx]=meshgrid(Ua,Wx);
    [Ub,Wy]=meshgrid(Ub,Wy);
    [Uc,Wz]=meshgrid(Uc,Wz);
    [Uw,W]=meshgrid(Uw,W);

Xw_Vector_Segment=Xw_Vector_Segment+Wx;Yw_Vector_Segment=Yw_Vector_Seg
ment+Wy;Zw_Vector_Segment=Zw_Vector_Segment+Wz;
    Weight=Weight+W;

    end
    Weight1=Weight1+Weight.*Uw;
    % Forming a patch

Xu_Vector_Segment=(Xu_Vector_Segment+Ua.*Xw_Vector_Segment);Yu_Vector_
Segment_Rational=(Yu_Vector_Segment_Rational+Ub.*Yw_Vector_Segment);Zu
_Vector_Segment=(Zu_Vector_Segment+Uc.*Zw_Vector_Segment);
    end
    % Rationalizing the X,Y, and Z vectors

Xu_Vector_Segment_Rational=Xu_Vector_Segment./Weight1;Yu_Vector_Segmen
t_Rational=Yu_Vector_Segment_Rational./Weight1;Zu_Vector_Segment_Ratio
nal=Zu_Vector_Segment./Weight1;
    % extracting and Storing the patch values

    for I=1:length(Xu_Vector_Segment)
        if i==0
            Raw=1;
        else
            Raw=i*length(Xu_Vector_Segment)+1;
        end
    for kk=1:length(Xu_Vector_Segment)
        Xu_Vector(Raw,Colume)=Xu_Vector_Segment_Rational(I,kk);
        Yu_Vector(Raw,Colume)=Yu_Vector_Segment_Rational(I,kk);
        Zu_Vector(Raw,Colume)=Zu_Vector_Segment_Rational(I,kk);
        Raw=Raw+1;

    end
      Colume=Colume+1;
    end
    L=L+1;
    end
    K=K+1;
end

hold all
X1=X';
Y1=Y';
Z1=Z';
% Plotting the surface
surf(Xu_Vector,Yu_Vector,Zu_Vector)
```

```
colormap summer
% Plotting the control net
plot3(X,Y,Z)
plot3(X1,Y1,Z1)
grid on
xlabel('X-Axis')
ylabel('Y-Axis')
zlabel('Z-Axis')
```

Appendix I Curve STEP File code

STEP_Curve.m

```
clear;clc;
% Input Control points
X=[ 1 4 6 8 10 4 ];
Y=[ 1 2 5 2 10 5 ];
Z=[ 0 0 0 0 0 0] ;
% order
k = 3;
% Name Of the file
Filename ='Spline.stp' ;
StepConversion(X,Y,Z,k,Filename) ;
```

StepConversion.m

```
function StepConversion(X,Y,Z,k,Filename)

% opening a file
fid = fopen(Filename,'w+t');
if fid < 0
    fprintf('Error \n');
    return;
end

format long
% Number of control points
n = length(X)-1 ;
kk=k;
% knot vector
for i= 0 : n + k
    if ( i < k )
        Knot_Vec( i + 1 ) = 0;
    elseif ( i >= k && i <= n )
        Knot_Vec( i + 1 ) = i - k + 1;
    else
        Knot_Vec( i + 1 ) = n - k + 2;
    end
end

T = zeros ( 1 , numel(Knot_Vec));
```

```
ii = 1 ;
% onverting Knot vector in zero to one span
for i = 1 : n + k + 1
   if i <= k
       T(i) = 0;
   elseif i > k && i <= n + k + 1 - k
     Space  =   linspace ( 0 , 1 , n - k + 3  ) ;
     T( i ) = Space (ii+1);
     ii = ii + 1 ;
   else
       T(i) = 1 ;
   end
end

Knot = zeros(1,n-k+3) ;

for i = 1 : numel(Knot)

    if i == 1
            Knot(i) = k ;
    elseif i == numel(Knot)
            Knot(i) = k ;
    else
            Knot(i) = 1 ;
    end
end

Knot1 = zeros(1,n-k+3) ;

for i = 1 : numel(Knot1)

   if i == 1
            Knot1(i) = 0 ;
    elseif i == numel(Knot1)
            Knot1(i) = 1 ;
    else
            Knot1(i) = T(kk+1) ;
            kk = kk+1 ;
    end
end

T = Knot ;
Temp_3 = 0 ;
for i = 1 : length(T)
    Temp_1 = num2str(T(i)) ;
    Temp_2 = length(Temp_1) ;
    Temp_3 = Temp_3 + Temp_2 ;
end

Knot_Vec=zeros(1,Temp_3+length(T)-1+2);
ii= 1;i=1;jj=1;
% Converting knot vector from double to character class
```

```
[Knot_Vec] = Charconversion(Knot_Vec,ii,i,jj,T) ;
%
txt_Knot  = [ char(Knot_Vec) ] ;
  T_1 = Knot1 ;
  Temp_3 = 0 ;
for i = 1 : length(T_1)
    Temp_1 = num2str(T_1(i)) ;
    Temp_2 = length(Temp_1) ;
    Temp_3 = Temp_3 + Temp_2 ;
end
t=zeros(1,Temp_3+length(T_1)-1+2);
ii= 1;i=1;jj=1;
[t] = Charconversion(t,ii,i,jj,T_1) ;
txt1_Knot  = char(t) ;

% Calculating number of instances
Number = 28 ;
Entities = zeros(1,length(X) );
for i = 1 : length(X)
Entities(i) = [Number];
Number = Number + 1;
end
Intance = Entities;

Temp_3 = 0 ;
for i = 1 : length(Intance)
    Temp_1 = num2str(Intance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1)  ;
end
T_Intance=zeros(1,Temp_3+length(Intance)-1+2);
ii= 1;i=1;jj=1;
[T_Intance] = Charconversion(T_Intance,ii,i,jj,Intance);

% Converting instnces into character class
txt__Intance  = char(T_Intance) ;
% Header Section
fprintf(fid,'ISO-10303-21;\n');
fprintf(fid,'HEADER;\n');
fprintf(fid,'FILE_DESCRIPTION (( ''STEP AP214'' ),''1'' );\n');
fprintf(fid,'FILE_NAME (''%s'',''2016-12-06T03:29:56'',( '''' ),( ''''
),''SwSTEP 2.0'',''SolidWorks 2015'','''' );\n',Filename); % change
the name
fprintf(fid,'FILE_SCHEMA (( ''AUTOMOTIVE_DESIGN'' ));\n');
fprintf(fid,'ENDSEC;\n');
% Data Section
fprintf(fid,'DATA;\n');
fprintf(fid,'#22 = PRODUCT_RELATED_PRODUCT_CATEGORY ( ''part'', '''',
( #4 ) ) ;\n');
fprintf(fid,'#23 = APPLICATION_PROTOCOL_DEFINITION ( ''draft
international standard'', ''automotive_design'', 1998, #8 ) ;\n');
fprintf(fid,'#24 = APPLICATION_PROTOCOL_DEFINITION ( ''draft
international standard'', ''automotive_design'', 1998, #6 ) ;\n');
```

```
fprintf(fid,'\n');
fprintf(fid,'#25 = SHAPE_DEFINITION_REPRESENTATION ( #1, #26 ) ;\n');
fprintf(fid,'\n');
fprintf(fid,'#1 = PRODUCT_DEFINITION_SHAPE ( ''NONE'', ''NONE'',  #2 )
;\n');
fprintf(fid,'#2 = PRODUCT_DEFINITION ( ''UNKNOWN'', '''', #3, #7 )
;\n');
fprintf(fid,'#3 = PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE (
''ANY'', '''', #4, .NOT_KNOWN. ) ;\n');
fprintf(fid,'#4 = PRODUCT ( ''%s'', ''%s'', '''', ( #5 ) )
;\n',Filename,Filename);
fprintf(fid,'#5 = PRODUCT_CONTEXT ( ''NONE'', #6, ''mechanical'' )
;\n');
fprintf(fid,'#6 = APPLICATION_CONTEXT ( ''automotive_design'' ) ;\n');
fprintf(fid,'#7 = PRODUCT_DEFINITION_CONTEXT ( ''detailed design'',
#8, ''design'' ) ;\n');
fprintf(fid,'#8 = APPLICATION_CONTEXT ( ''automotive_design'' ) ;\n');
fprintf(fid,'\n');
fprintf(fid,'#26 =
GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION ( ''%s'', ( #21,
#9 ), #13 ) ;\n',Filename);
% B-spline Curve Entity
Number = 28;
fprintf(fid,'#27 = B_SPLINE_CURVE_WITH_KNOTS ( ''NONE'', %d
,%s,.UNSPECIFIED., .F., .F.,%s,%s,.UNSPECIFIED. ) ; \n',k-
1,txt__Intance,txt_Knot,txt1_Knot);
for i = 1 : length(X)
fprintf(fid,'#%d = CARTESIAN_POINT ( ''NONE'',( %1.19f, %1.19f,
%1.19f) ) ;\n',Number,X(i),Y(i),Z(i));
Number = Number + 1 ;
end

Number = 28+length(X) ;
Entities = zeros(1,length(X)-1 );

for i = 1 : length(X) -1
Entities(i) = Number;
Number = Number + 7;
end
Intance = Entities;
for i = 1 : length(Intance)
    Temp_1 = num2str(Intance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1)  ;
end
ii= 1;i=1;jj=1;

while 1
     if i == 1
         T_Intance_Curveset(jj) = ',' ;
     elseif i == (2*numel(Intance) + 1)
         T_Intance_Curveset(jj) = ')';
         break
```

```
      else
         if rem(i,2) == 0
             if jj < i
                 jj = i ;
             end
             if Intance(i/2) > 9 || (length(num2str(Intance(i/2)))) > 1
                 Temp = num2str(Intance(ii)) ;
                 T_Intance_Curveset(jj) = '#';
                 jj = jj+ 1 ;
                 for j = 1 : length(Temp)
                 T_Intance_Curveset(jj) = Temp(j) ;
                      jj = jj + 1;
                 end
                 ii = ii + 1 ;
             else
         T_Intance_Curveset(jj) = '#';
         jj = jj+ 1 ;
         T_Intance_Curveset(jj)= num2str(Intance(ii)) ;
          ii = ii + 1 ;
          jj = jj + 1 ;
             end
         else
          T_Intance_Curveset(jj) = ',' ;
          jj = jj + 1 ;
         end
      end
 i = i + 1 ;
end

txt__Intance  = [ char(T_Intance_Curveset) ] ;
fprintf(fid,'#21 = GEOMETRIC_CURVE_SET ( ''NONE'', ( #27%s
);\n',txt__Intance);
Intance_trimmed = 28+length(X) ;
I = 1;
% Control polygon (Optional)
for i = 1 : length(X) - 1
Magnitude = sqrt((X(i)-X(i+1))^2+(Y(i)-Y(i+1))^2+(Z(i)-Z(i+1))^2) ;
fprintf(fid,'#%d = TRIMMED_CURVE ( ''NONE'', #%d, ( PARAMETER_VALUE (
0.0000000000000000000 ), #%d ), ( PARAMETER_VALUE (
1.0000000000000000000 ), #%d ), .T., .PARAMETER. )
;\n',Intance_trimmed,Intance_trimmed+3*I,Intance_trimmed+I,Intance_tri
mmed+2*I);
fprintf(fid,'#%d = CARTESIAN_POINT ( ''NONE'',  ( %f, %f, %f ) )
;\n',Intance_trimmed+I,X(i),Y(i),Z(i));
fprintf(fid,'#%d = CARTESIAN_POINT ( ''NONE'',  ( %f, %f, %f ) )
;\n',Intance_trimmed+2*I,X(i+1),Y(i+1),Z(i+1));
fprintf(fid,'#%d  = LINE ( ''NONE'', #%d, #%d);
\n',Intance_trimmed+3*I,Intance_trimmed+5*I,Intance_trimmed+4*I);
fprintf(fid,'#%d  = VECTOR ( ''NONE'', #%d, %f )
;\n',Intance_trimmed+4*I,Intance_trimmed+6*I,Magnitude);
```

```
fprintf(fid,'#%d  = DIRECTION ( ''NONE'',  ( %f, %f, %f ) )
;\n',Intance_trimmed+6*I,(X(i+1)-X(i))/Magnitude,(Y(i+1)-
Y(i))/Magnitude,(Z(i+1)-Z(i))/Magnitude);
fprintf(fid,'#%d  = CARTESIAN_POINT ( ''NONE'',  (%f , %f, %f ) )
;\n',Intance_trimmed+5*I,X(i),Y(i),Z(i));
Intance_trimmed = Intance_trimmed + 7;
end
fprintf(fid,'#9 = AXIS2_PLACEMENT_3D ( ''NONE'', #10, #11, #12 )
;\n');
fprintf(fid,'#10 = CARTESIAN_POINT ( ''NONE'',  (
0.000000000000000000, 0.000000000000000000, 0.000000000000000000 )
) ;\n');
fprintf(fid,'#11 = DIRECTION ( ''NONE'',  ( 0.000000000000000000,
0.000000000000000000, 1.000000000000000000 ) ) ;\n');
fprintf(fid,'#12 = DIRECTION ( ''NONE'',  ( 1.000000000000000000,
0.000000000000000000, 0.000000000000000000 ) ) ;\n');
fprintf(fid,'\n');
fprintf(fid,'#13 =( GEOMETRIC_REPRESENTATION_CONTEXT ( 3 )
GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT ( ( #14 ) )
GLOBAL_UNIT_ASSIGNED_CONTEXT ( ( #15, #16, #17 ) )
REPRESENTATION_CONTEXT ( ''NONE'', ''WORKASPACE'' ) );\n');
fprintf(fid,'#14 = UNCERTAINTY_MEASURE_WITH_UNIT (LENGTH_MEASURE(
1.00000000000000100E-005 ), #15, ''distance_accuracy_value'',
''NONE'');\n');
fprintf(fid,'#15 =( CONVERSION_BASED_UNIT ( ''INCH'', #19 )
LENGTH_UNIT ( ) NAMED_UNIT ( #18) );\n');
fprintf(fid,'#18 = DIMENSIONAL_EXPONENTS ( 1.000000000000000000,
0.000000000000000000, 0.000000000000000000, 0.000000000000000000,
0.000000000000000000, 0.000000000000000000, 0.000000000000000000 )
;\n');
fprintf(fid,'#19 = LENGTH_MEASURE_WITH_UNIT ( LENGTH_MEASURE(
0.02539999999999999900 ), #20 );\n');
fprintf(fid,'#20 =( LENGTH_UNIT ( ) NAMED_UNIT ( * ) SI_UNIT ( $,
.METRE. ) );\n');
fprintf(fid,'#17=( NAMED_UNIT ( * ) SI_UNIT ( $, .STERADIAN. )
SOLID_ANGLE_UNIT ( ) );\n');
fprintf(fid,'#16 =( NAMED_UNIT ( * ) PLANE_ANGLE_UNIT ( ) SI_UNIT ( $,
.RADIAN. ) );\n');
fprintf(fid,'ENDSEC;\n');
fprintf(fid,'END-ISO-10303-21;\n');
fclose(fid) ;
end
```

Charconversion.m

```
function [T1] = Charconversion(T1,ii,i,jj,T)

while 1
     if i == 1
         T1(jj) = '(' ;
```

```
      elseif i == (2*numel(T) + 1)
          T1(jj) = ')';
          break
      else
         if rem(i,2) == 0
             if jj < i
                 jj = i ;
             end
             if T(i/2) > 9 || (length(num2str(T(i/2)))) > 1
                 Temp = num2str(T(ii)) ;
                 for j = 1 : length(Temp)
                 T1(jj) = Temp(j) ;
                      jj = jj + 1;
                 end
                 ii = ii + 1 ;
             else
          T1(jj)= num2str(T(ii)) ;
           ii = ii + 1 ;
           jj = jj + 1 ;
             end
          else
           T1(jj) = ',' ;
           jj = jj + 1 ;
          end
      end
 i = i + 1 ;
 end

end
```

Appendix J Surface STEP File code

STEP_surface.m

```
clear;
clc;
% Open File and input name of the fine
fid = fopen('StepSurface.stp','w+t');
if fid < 0
    fprintf('Error \n');
    return;
end
% Input X,Y,and Z Coordinates
X=[31 25.861 4.419 0
    28.34 20.503 0.272 0
    32.683 29.25 7.042 0
    31 25.861 4.419 -2.156];
Y=[0 12.399 14.220 0
```

```
    3.11 12.744 10.589 -4.093
    -1.968 12.18 16.517 4.356
    0 12.399 14.220 5.280];
Z=[13 13 13 13
    8.666 8.666 8.666 8.666
    4.356 4.356 4.35 4.356
    0 0 0 0];
%Input orders in U and W Directions respectively
k=3;l=3;
Size = size(X) ;
%Number of raws
n=Size(1)-1;
%Number of Columns
m=Size(2)-1;

K = k - 1 ;
L = l - 1;

kk=k;
ll=l;
% Knot vector in u direction
for i= 0 : n + k

    if ( i < k )

        T_U( i + 1 ) = 0;

    elseif ( i >= k && i <= n )

        T_U( i + 1 ) = i - k + 1;

    else

        T_U( i + 1 ) = n - k + 2;

    end

end
T_U1 = zeros ( 1 , numel(T_U));
ii = 1 ;

% Converting a knot vector in a zero to one span
for i = 1 : n + k + 1
   if i <= k
       T_U1(i) = 0;
   elseif i > k && i <= n + k + 1 - k
     Space  =   linspace ( 0 , 1 , n - k + 3  ) ;
     T_U1( i ) = Space (ii+1);
     ii = ii + 1 ;
   else
       T_U1(i) = 1 ;
```

```
   end

end

% Knot vector in w direction
for i= 0 : m + l
    if ( i < l )
        T_w( i + 1 ) = 0;
    elseif ( i >= l && i <= m )
        T_w( i + 1 ) = i - l + 1;
    else
        T_w( i + 1 ) = m - l + 2;
    end
end
T_w1 = zeros ( 1 , numel(T_w));
ii = 1 ;

% Converting a knot vector in a zero to one span
for i = 1 : m + l + 1
   if i <= l
       T_w1(i) = 0;
   elseif i > l && i <= m + l + 1 - l
     Space  =   linspace ( 0 , 1 , m - l + 3  ) ;
     T_w1( i ) = Space (ii+1);
     ii = ii + 1 ;
   else
       T_w1(i) = 1 ;
   end
end

% Converting knot vector in a character class
[TU,TU1] = Knotvector_Conversion(k,kk,n,T_U1) ;
[TV,TV1] = Knotvector_Conversion(l,ll,m,T_w1) ;

% Control Points Arrangments %
Number = 74 ;
Entities = zeros(1,(n+1)*(m+1) );
for i = 1 : (n+1)*(m+1)
Entities(i) = Number;
Number = Number + 1;
end

Instance = Entities;
Temp_3 = 0 ;
for i = 1 : length(Instance)
    Temp_1 = num2str(Instance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1)  ;
end

Surface_Instance=zeros(1,Temp_3+length(Instance)-
1+2+(2*(n+1))+(n+1)*(m+1));
j = 1;
```

```
Index = 0 ;
% Converting a control point matrix in a character class
while 1

    if j == 1
        Surface_Instance(j) = '(' ;
        j = j + 1;

    elseif j == length(Surface_Instance)
         Surface_Instance(j) = ')' ;
         break

    else

        Instance1 = Instance(Instance(Index+1)<= Instance & Instance
<= Instance (Index+ m +1 ) ) ;
        Index = Index + m +1 ;
        Temp_3 = 0 ;

          for i = 1 : length(Instance1)

              Temp_1 = num2str(Instance1(i)) ;
              Temp_3 = Temp_3 + length(Temp_1)  ;

          end

         Temp_Instance=zeros(1,Temp_3+length(Instance1)-1+2+m+1);
         ii= 1;i=1;jj=1;
         Temp_Instance =
Charconversion_Controlpoints(Temp_Instance,ii,i,jj,Instance1) ;

         for i = 1 : length(Temp_Instance)
          Surface_Instance(j) = Temp_Instance(i) ;
          j= j + 1;
         end

          if j < length(Surface_Instance )-2
             Surface_Instance(j) = ',';
             j = j + 1 ;

          end
    end
end

Controlpoint_List = char(Surface_Instance) ;

% Header Section
fprintf(fid,'ISO-10303-21;\n');
fprintf(fid,'HEADER;\n');
fprintf(fid,'FILE_DESCRIPTION (( ''STEP AP214'' ),''1'' );\n');
```

```
fprintf(fid,'FILE_NAME (''%s'',''2016-12-06T03:29:56'',( '''' ),( ''''
),''SwSTEP 2.0'',''SolidWorks 2015'','''' );\n','StepSurface_Matlab');
% change the name
fprintf(fid,'FILE_SCHEMA (( ''AUTOMOTIVE_DESIGN'' ));\n');
fprintf(fid,'ENDSEC;\n');
% Data Section
fprintf(fid,'DATA;\n');
fprintf(fid,'\n');
fprintf(fid,'#10=SHAPE_REPRESENTATION_RELATIONSHIP('''','''',#62,#22);
\n');
fprintf(fid,'#11=COLOUR_RGB('''',0.,0.,0.);\n');
fprintf(fid,'#12=FILL_AREA_STYLE_COLOUR('''',#11);\n');
fprintf(fid,'#13=FILL_AREA_STYLE('''',(#12));\n');
fprintf(fid,'#14=SURFACE_STYLE_FILL_AREA(#13);\n');
fprintf(fid,'#15=SURFACE_SIDE_STYLE('''',(#14));\n');
fprintf(fid,'#16=SURFACE_STYLE_USAGE(.BOTH.,#15);\n');
fprintf(fid,'#17=MECHANICAL_DESIGN_GEOMETRIC_PRESENTATION_REPRESENTATI
ON('''',(#19),#61);\n');
fprintf(fid,'#18=PRESENTATION_STYLE_ASSIGNMENT((#16));\n');
fprintf(fid,'#19=STYLED_ITEM('''',(#18),#21);\n');
fprintf(fid,'#20=PRESENTATION_LAYER_ASSIGNMENT(''Default'','''',(#21))
;\n');
fprintf(fid,'#21=SHELL_BASED_SURFACE_MODEL(''shell_1'',(#23));\n');
fprintf(fid,'\n');
fprintf(fid,'#22=MANIFOLD_SURFACE_SHAPE_REPRESENTATION(''shell_rep_0''
,(#21,#64),#61);\n');
fprintf(fid,'#23=OPEN_SHELL('''',(#24));\n');
fprintf(fid,'#24=ADVANCED_FACE('''',(#25),#43,.T.);\n');
fprintf(fid,'#25=FACE_OUTER_BOUND('''',#26,.T.);\n');
fprintf(fid,'#26=EDGE_LOOP('''',(#27,#28,#29,#30));\n');
fprintf(fid,'#27=ORIENTED_EDGE('''',*,*,#31,.T.);\n');
fprintf(fid,'#28=ORIENTED_EDGE('''',*,*,#32,.T.);\n');
fprintf(fid,'#29=ORIENTED_EDGE('''',*,*,#33,.T.);\n');
fprintf(fid,'#30=ORIENTED_EDGE('''',*,*,#34,.T.);\n');
fprintf(fid,'#31=EDGE_CURVE('''',#39,#40,#35,.T.);\n');
fprintf(fid,'#32=EDGE_CURVE('''',#40,#41,#36,.T.);\n');
fprintf(fid,'#33=EDGE_CURVE('''',#41,#42,#37,.T.);\n');
fprintf(fid,'#34=EDGE_CURVE('''',#42,#39,#38,.T.);\n');
fprintf(fid,'#39=VERTEX_POINT('''',#70);\n');
fprintf(fid,'#70=CARTESIAN_POINT('''',(%f,%f,%f));\n',X(1,1),Y(1,1),Z(
1,1));
fprintf(fid,'#40=VERTEX_POINT('''',#71);\n');
fprintf(fid,'#71=CARTESIAN_POINT('''',(%f,%f,%f));\n',X(end,1),Y(end,1
),Z(end,1));
fprintf(fid,'#41=VERTEX_POINT('''',#72);\n');
fprintf(fid,'#72=CARTESIAN_POINT('''',(%f,%f,%f));\n',X(end,end),Y(end
,end),Z(end,end));
fprintf(fid,'#42=VERTEX_POINT('''',#73);\n');
fprintf(fid,'#73=CARTESIAN_POINT('''',(%f,%f,%f));\n',X(1,end),Y(1,end
),Z(1,end));
fprintf(fid,'#44=SHAPE_DEFINITION_REPRESENTATION(#45,#62);\n');
fprintf(fid,'#45=PRODUCT_DEFINITION_SHAPE(''Document'','''',#47);\n');
```

```
fprintf(fid,'#46=PRODUCT_DEFINITION_CONTEXT(''3D Mechanical
Parts'',#51,''design'');\n');
fprintf(fid,'#47=PRODUCT_DEFINITION(''A'',''First
version'',#48,#46);\n');
fprintf(fid,'#48=PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE(''
A'',''First version'',#53,.MADE.);\n');
fprintf(fid,'#49=PRODUCT_RELATED_PRODUCT_CATEGORY(''tool'',''tool'',(#
53));\n');
fprintf(fid,'#50=APPLICATION_PROTOCOL_DEFINITION(''Draft International
Standard'',''automotive_design'',1999,#51);\n');
fprintf(fid,'#51=APPLICATION_CONTEXT(''data for automotive mechanical
design processes'');\n');
fprintf(fid,'#52=PRODUCT_CONTEXT(''3D Mechanical
Parts'',#51,''mechanical'');\n');
fprintf(fid,'#53=PRODUCT(''Document'',''Document'',''Rhino converted
to STEP'',(#52));\n');
fprintf(fid,'#54=((LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT(.MILLI.,.METRE.))
;\n');
fprintf(fid,'#55=(NAMED_UNIT(*)PLANE_ANGLE_UNIT());\n');
fprintf(fid,'#56=DIMENSIONAL_EXPONENTS(0.,0.,0.,0.,0.,0.,0.);\n');
fprintf(fid,'#57=PLANE_ANGLE_MEASURE_WITH_UNIT(PLANE_ANGLE_MEASURE(0.0
1745329252),#55);\n');
fprintf(fid,'#58=(CONVERSION_BASED_UNIT(''DEGREES'',#57)NAMED_UNIT(#56
)PLANE_ANGLE_UNIT());\n');
fprintf(fid,'#59=(NAMED_UNIT(*)SI_UNIT($,.STERADIAN.)SOLID_ANGLE_UNIT(
));\n');
fprintf(fid,'#60=UNCERTAINTY_MEASURE_WITH_UNIT(LENGTH_MEASURE(0.001),#
54,''DISTANCE_ACCURACY_VALUE'',''Maximum model space distance between
geometric entities at asserted connectivities'');\n');
fprintf(fid,'#61=(GEOMETRIC_REPRESENTATION_CONTEXT(3)GLOBAL_UNCERTAINT
Y_ASSIGNED_CONTEXT((#60))GLOBAL_UNIT_ASSIGNED_CONTEXT((#59,#58,#54))RE
PRESENTATION_CONTEXT(''ID1'',''3D''));\n');
fprintf(fid,'\n');
fprintf(fid,'#62=SHAPE_REPRESENTATION(''Document'',(#63,#64),#61);\n')
;
fprintf(fid,'#63=AXIS2_PLACEMENT_3D('''',#69,#65,#66);\n');
fprintf(fid,'#64=AXIS2_PLACEMENT_3D('''',#690,#67,#68);\n');
fprintf(fid,'#65=DIRECTION('''',(0.,0.,1.));\n');
fprintf(fid,'#66=DIRECTION('''',(1.,0.,0.));\n');
fprintf(fid,'#67=DIRECTION('''',(0.,0.,1.));\n');
fprintf(fid,'#68=DIRECTION('''',(1.,0.,0.));\n');
fprintf(fid,'#69=CARTESIAN_POINT('''',(0.,0.,0.));\n');
fprintf(fid,'#690=CARTESIAN_POINT('''',(0.,0.,0.));\n');
fprintf(fid,'#43=B_SPLINE_SURFACE_WITH_KNOTS('''',%d,%d,%s,.UNSPECIFIE
D.,.F.,.F.,.F.,%s,%s,%s,%s,.UNSPECIFIED.);\n',k-1,l-
1,Controlpoint_List,TU,TV,TU1,TV1);
% Calculating instance numbers for four B-spline curves
 Number = 74 ;
 Number1 = 74 ;
for i = 1 : n+1
    for j = 1 : m+1
```

```
fprintf(fid,'#%d=CARTESIAN_POINT('''',(%f,%f,%f));\n',Number,X(i,j),Y(
i,j),Z(i,j));
 Number  = Number + 1 ;
 Number1 = Number1 +1 ;
    end
end

Number_Curve_U = Number ;
Entities = zeros(1,n+1 );
for i = 1 : n +1
Entities(i) = Number1;
Number1 = Number1 + 1;
end

Intance = Entities;
Temp_3 = 0 ;
for i = 1 : length(Intance)
    Temp_1 = num2str(Intance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1);

end
Temp_Instance=zeros(1,Temp_3+length(Intance)-1+2);
ii= 1;i=1;jj=1;
Temp_Instance =
Charconversion_Controlpoints(Temp_Instance,ii,i,jj,Intance) ;
Instance_U = char(Temp_Instance) ;
fprintf(fid,'#35=B_SPLINE_CURVE_WITH_KNOTS('''',%d,%s,.UNSPECIFIED.,.F
.,.F.,%s,%s,.UNSPECIFIED.);\n',k-1,Instance_U,TU,TU1);

for i = 1 : n+1

fprintf(fid,'#%d=CARTESIAN_POINT('''',(%f,%f,%f));\n',Number_Curve_U,X
(i,1),Y(i,1),Z(i,1));
Number_Curve_U  = Number_Curve_U + 1 ;


end

Number_Curve_Vend = Number_Curve_U ;
Entities = zeros(1,m );
for i = 1 : m +1
Entities(i) = Number1;
Number1 = Number1 + 1;
end

Intance = Entities;
Temp_3 = 0 ;
for i = 1 : length(Intance)
    Temp_1 = num2str(Intance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1)  ;
end
```

```
Temp_Instance=zeros(1,Temp_3+length(Intance)-1+2);
ii= 1;i=1;jj=1;
Temp_Instance =
Charconversion_Controlpoints(Temp_Instance,ii,i,jj,Intance) ;
Instance_Vend = char(Temp_Instance) ;
fprintf(fid,'#36=B_SPLINE_CURVE_WITH_KNOTS('''',%d,%s,.UNSPECIFIED.,.F
.,.F.,%s,%s,.UNSPECIFIED.);\n',l-1,Instance_Vend,TV,TV1);

for i = 1 : m+1

fprintf(fid,'#%d=CARTESIAN_POINT('''',(%f,%f,%f));\n',Number_Curve_Ven
d,X(end,i),Y(end,i),Z(end,i));
 Number_Curve_Vend  = Number_Curve_Vend + 1 ;

end

Number_Curve_Uend =  Number_Curve_Vend ;
Entities = zeros(1,n+1 );
for i = 1 : n +1
Entities(i) = Number1;
Number1 = Number1 + 1;
end

Intance = Entities;
Temp_3 = 0 ;
for i = 1 : length(Intance)
    Temp_1 = num2str(Intance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1)  ;
end
Temp_Instance=zeros(1,Temp_3+length(Intance)-1+2);
ii= 1;i=1;jj=1;
Temp_Instance =
Charconversion_Controlpoints(Temp_Instance,ii,i,jj,Intance) ;
Instance_Uend = char(Temp_Instance) ;
fprintf(fid,'#37=B_SPLINE_CURVE_WITH_KNOTS('''',%d,%s,.UNSPECIFIED.,.F
.,.F.,%s,%s,.UNSPECIFIED.);\n',k-1,Instance_Uend,TU,TU1);

for i = n+1 : -1 : 1

fprintf(fid,'#%d=CARTESIAN_POINT('''',(%f,%f,%f));\n',Number_Curve_Uen
d,X(i,end),Y(i,end),Z(i,end));
 Number_Curve_Uend  = Number_Curve_Uend + 1 ;

end

Number_Curve_V = Number_Curve_Uend ;
Entities = zeros(1,m+1 );
for i = 1 : m+1
Entities(i) = Number1;
Number1 = Number1 + 1;
end
```

```
Intance = Entities;
Temp_3 = 0 ;
for i = 1 : length(Intance)
    Temp_1 = num2str(Intance(i)) ;
    Temp_3 = Temp_3 + length(Temp_1)  ;
end
Temp_Instance=zeros(1,Temp_3+length(Intance)-1+2);
ii= 1;i=1;jj=1;
Temp_Instance =
Charconversion_Controlpoints(Temp_Instance,ii,i,jj,Intance) ;
Instance_V = char(Temp_Instance) ;
fprintf(fid,'#38=B_SPLINE_CURVE_WITH_KNOTS('''',%d,%s,.UNSPECIFIED.,.F
.,.F.,%s,%s,.UNSPECIFIED.);\n',l-1,Instance_V,TV,TV1);

for i = m+1 : -1 : 1

fprintf(fid,'#%d=CARTESIAN_POINT('''',(%f,%f,%f));\n',Number_Curve_V,X
(1,i),Y(1,i),Z(1,i));
 Number_Curve_V  = Number_Curve_V + 1 ;

end
fprintf(fid,'ENDSEC;\n');
fprintf(fid,'END-ISO-10303-21;\n');
% Closing the file
fid(close);
```

Knotvector_Conversion.m

```
function [T,T1] = Knotvector_Conversion(k,kk,n,T_U1)
Knot = zeros(1,n-k+3) ;
for i = 1 : numel(Knot)

    if i == 1
            Knot(i) = k ;
    elseif i == numel(Knot)
            Knot(i) = k ;
    else
            Knot(i) = 1 ;
    end
end

Knot1 = zeros(1,n-k+3) ;

for i = 1 : numel(Knot1)

   if i == 1
            Knot1(i) = 0 ;
    elseif i == numel(Knot1)
            Knot1(i) = 1 ;
    else
```

```
            Knot1(i) = T_U1(kk+1) ;
            kk = kk+1 ;
    end
end

T_U1 = Knot ;
Temp_3 = 0 ;
for i = 1 : length(T_U1)
    Temp_1 = num2str(T_U1(i)) ;
    Temp_2 = length(Temp_1) ;
    Temp_3 = Temp_3 + Temp_2 ;
end
T=zeros(1,Temp_3+length(T_U1)-1+2);
ii= 1;i=1;jj=1;
[T] = Charconversion(T,ii,i,jj,T_U1) ;
 T = char(T) ;

 Temp = Knot1 ;
  Temp_3 = 0 ;
for i = 1 : length(Temp)
    Temp_1 = num2str(Temp(i)) ;
    Temp_2 = length(Temp_1) ;
    Temp_3 = Temp_3 + Temp_2 ;
end
T1=zeros(1,Temp_3+length(Temp)-1+2);
ii= 1;i=1;jj=1;
[T1] = Charconversion(T1,ii,i,jj,Temp) ;
T1 = char(T1) ;
end
```

Charconversion_Controlpoints.m

```
function [T_Instance] =
Charconversion_Controlpoints(T_Instance,ii,i,jj,Instance1)

while 1
     if i == 1
         T_Instance(jj) = '(' ;
     elseif i == (2*numel(Instance1) + 1)
         T_Instance(jj) = ')';
         break
     else
        if rem(i,2) == 0
            if jj < i
                jj = i ;
            end
            if Instance1(i/2) > 9 || (length(num2str(Instance1(i/2))))
> 1
                Temp = num2str(Instance1(ii)) ;
                T_Instance(jj) = '#';
```

```
                jj = jj+ 1 ;
                for j = 1 : length(Temp)
                T_Instance(jj) = Temp(j) ;
                     jj = jj + 1;
                end
                ii = ii + 1 ;
            else
        T_Instance(jj) = '#';
        jj = jj+ 1 ;
        T_Instance(jj)= num2str(Instance1(ii)) ;
         ii = ii + 1 ;
         jj = jj + 1 ;
            end
        else
         T_Instance(jj) = ',' ;
         jj = jj + 1 ;
        end
    end
 i = i + 1 ;

end
```

References

[1] Achille Messac, "*Optimization in Practice with MATLAB®: For Engineering Students and Professionals",* March 18, 2015

[2] IanGibson· DavidRosen, BrentStucker "Additive Manufacturing Technologies: 3D Printing, Rapid Prototyping, and Direct Digital Manufacturing", 2015

[3] Michael Mortenson, "Geometric Modeling", April 15, 2006

[4] David Salomon," Curves and Surfaces for Computer Graphics", September 8, 2005

[5] D.Brackett, I. Ashcroft, R. Hague," TOPOLOGY OPTIMIZATION FOR ADDITIVE MANUFACTURING", August 17 2011

[6] Robert W. Schuler, "The Application of ISO 10303-11 (the EXPRESSLanguage) in Defining Data Models for Software Designand Implementation", April 2001

[7] Les Piegl, University of South Florida , "On NURBS: A Survey", January 1991

[8] David Loffredo," Fundamentals of STEP Implementation", STEP Tools, Inc., Rensselaer Technology Park, Troy, New York 12180

[9] Ap214, Steptools.com, Available online at: "http://www.steptools.com/stds/stp_aim/html/schema.html#step_merged_ap_schema" [Accessed: 15- July 2016]

[10] ISO 10303-21, Wikipedia, [Online]. Available at : https://en.wikipedia.org/wiki/ISO_10303-21 [Accessed: 22 May 2016]

[11] Vince, John A, "Mathematics for Computer Graphics",2013.

[12] Martin Philip Bendsoe, Ole Sigmund," Topology Optimization: Theory, Methods, and Applications", December 1, 2003

[13] Richard H. Bartels, John C. Beatty, " An introduction to the use of splines in computer graphics" May 1995

Biographical Information

Harshit Jani has completed his bachelors in Automotive engineering from Gujarat Technological University, Ahmedabad, India in the year 2013. During his undergraduate period, he found keen interest in CAD/CAE and programming. Therefore, he joined the university of Texas Arlington in August 2014 for pursuing his dream.