FRANKFURT UNIVERSITY OF APPLIED SCIENCES

MASTERS THESIS

Discovering the Potential of Surrogate Modeling For Predicting Multiple Moving Resonances With Complex-Valued Frequency Domain Data

---

Author:
Milan DABHI

Supervisor:
Prof. Dr. Peter THOMA
Prof. Dr. Egbert FALKENBERG

A thesis submitted in fulfillment of the requirements for the degree of Msc. High Integrity Systems

in the department of

Computer Science and Engineering(Fb2)

December 29, 2023

Declaration of Authorship

I, Milan DABHI, declare that this thesis titled, " Discovering the Potential of Surrogate Modeling For Predicting Multiple Moving Resonances With Complex-Valued Frequency Domain Data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while I was a candidate for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my work.
- I have acknowledged all primary sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 29.12.2023

"Knowledge is not simply another commodity. On the contrary, knowledge is never used up. It increases by diffusion and grows by dispersion. "

- Daniel J. Boorstin

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Abstract

Fb2
Computer Science and Engineering

Msc. High Integrity Systems

Discovering the Potential of Surrogate Modeling For Predicting Multiple Moving Resonances With Complex-Valued Frequency Domain Data

by Milan DABHI

Building upon the foundation laid by previous research, this thesis ventures further into surrogate modeling, emphasizing complex-valued Frequency Domain (FD) data characteristic of electromagnetic systems. Predicting moving resonance with high fidelity remains a crucial obstacle in surrogate model development. While previous efforts successfully employed circle fitting to model such data, yielding smoother parameter curves, limitations were encountered in achieving the desired smoothness of the curve, particularly when addressing the intricate behaviors of resonance. To advance this endeavor, this study explores alternative geometrical fitting techniques, including ellipse and spline fitting, to refine the surrogate modeling process. Despite rigorous attempts with these methods, the quest for an optimally smooth curve representation persisted. This thesis proposes a continuation of this investigation, seeking novel methodologies and adaptations in the surrogate modeling framework that can embrace the complexity of the data. It was found that none of the techniques used to fit the data locally yielded the desired outcome, so another perspective needs to be implemented to find a better approach to resolving the problem.

Keywords: Surrogate Modelling, Frequency Domain, Circle Extraction, Ellipse Extraction, Quadratic Spline Fitting, Cubic Spline Fitting, Quartic Spline Fitting, Polar plots, Smooth Curves, Curvature.

Acknowledgements

I hereby extend my deep appreciation and sincere gratitude to those who have played an instrumental role in the fruition of this thesis. Foremost, I express my profound thanks to my primary advisor, Prof. Dr. Peter Thoma. His expert guidance, unwavering support, and invaluable insights have been pivotal in shaping the ideas and methodologies encapsulated within this work. His belief in my potential and the opportunity to engage in this approach have been fundamental to my academic growth and the successful completion of this thesis. Additionally, I am indebted to my secondary advisor, Prof. Dr. Egbert Falkenberg, whose continuous encouragement and support have been invaluable during my research. His perspectives and feedback have significantly contributed to the depth and quality of this study. My tenure at the Frankfurt University of Applied Sciences has been a journey of immense learning and opportunity, for which I am immensely grateful. The knowledge and experiences gained here have been integral to my professional and personal development. Lastly, my deepest gratitude goes to my parents. Their unconditional love, unwavering support, and constant encouragement have been the cornerstone of my strength and resilience. Their sacrifice and belief in my abilities are the driving force behind my endeavors and accomplishments during my Master's education.

Contents

List of Figures

List of Algorithms

Listings

List of Abbreviations

FD	**F**requency **D**omain
PRM	**P**olynomial **R**egression **M**ethod
LSCF	**L**east **S**quares **C**ircle **F**it
SLSQP	**S**equential **L**east **S**quares **P**rogramming
LSEF	**L**east **S**quares **E**llipse **F**it
CI	**C**ontinuous **I**ntegration
CD	**C**ontinuous **D**evelopment
S-Parameters	**S**cattering Parmeters

Dedicated to my family and my teachers . . .

Chapter 1

Introduction

Surrogate modeling is a technique used to increase the speed of simulations by training a model that mimics the underlying physics of a process. It is commonly used in computationally intensive or time-consuming simulations or when there are practical limitations (Easum et al., 2017). For example, Frequency Domain (FD) Simulation of a dipole antenna. Fig.1.1 shows the numerically simulated data within a spectrum of 25-35 GHz for various values of a geometry parameter "Lambda". Fig.1.2 shows a clear representation with fewer lambda parameter points.

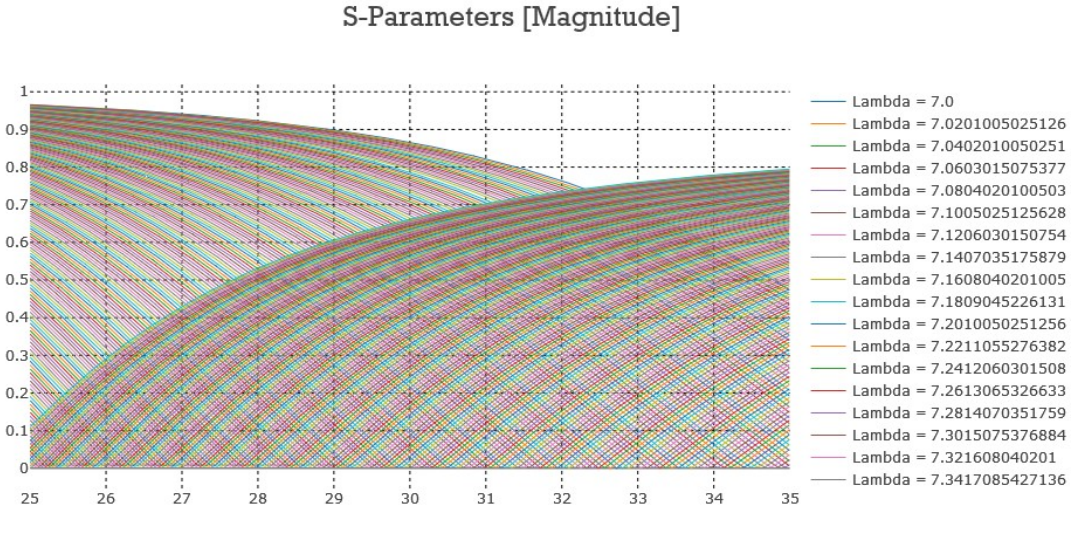


FIGURE 1.1: Simulation Data From Dipole Antenna

The simple dipole antenna exhibits the behavior of a moving resonance. A surrogate model based on the Polynomial Regression Method (PRM) was utilized by Dasi, 2023 to interpolate unknown frequency domain simulation data, specifically complex-valued S-parameters. Dasi, 2023 used Least Square Circle Fit (LSCF) to find the best-fitting circle and extract circle parameters to create a polynomial regression model. The figures 1.3, 1.4, 1.5, 1.6 demonstrate the model fitting via a polynomial regression method for the corner frequency of 25.0 GHz.

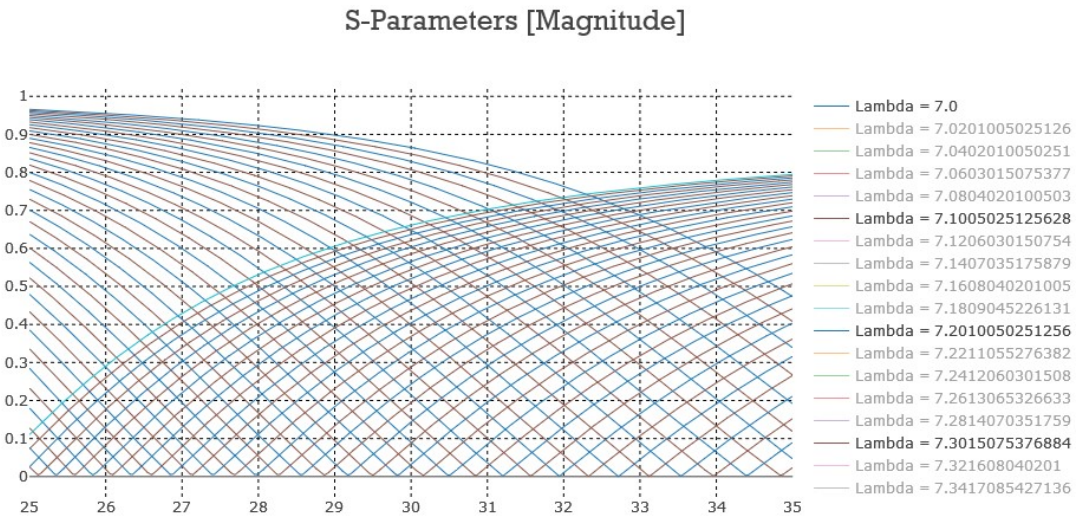


FIGURE 1.2: Moving Resonances in a Dipole Antenna

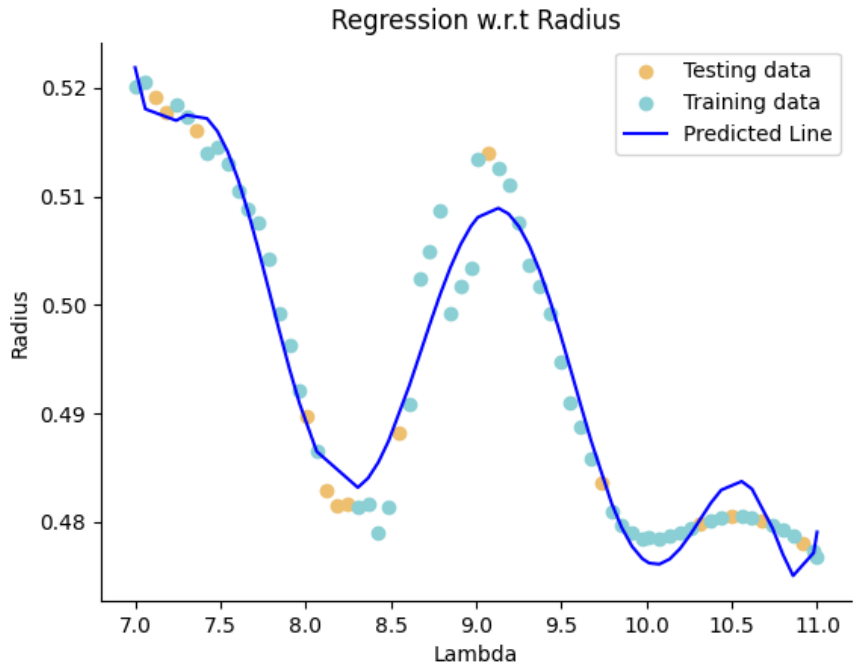


FIGURE 1.3: Polynomial Regression Model for Radius

1.1 Problem Description

The figures 1.3, 1.4, 1.5, 1.6 represent models of circle parameters and their respective extracted circle parameters are illustrated as a function of lambda (dipole length) for a corner frequency of 25 GHz in figures 1.7, 1.8, 1.9, 1.10. In the previous comprehensive study, which was carried out by Dasi, 2023, the goal was to transform the data obtained from the simulation into smoother curves. However, the desired smoothness with the help of LSCF is not achieved for the corner frequencies.

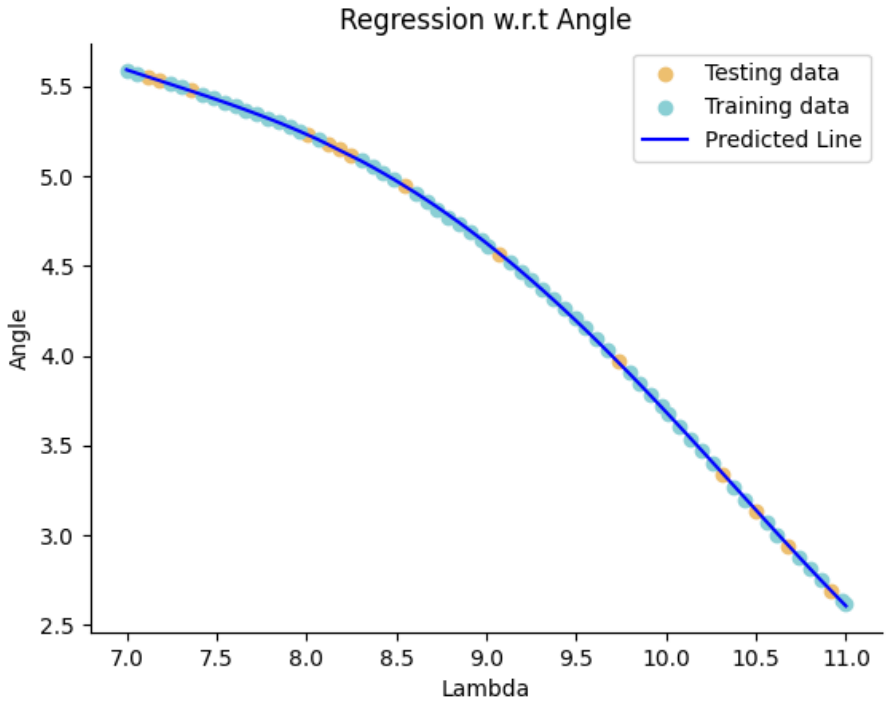


FIGURE 1.4: Polynomial Regression Model for Phase

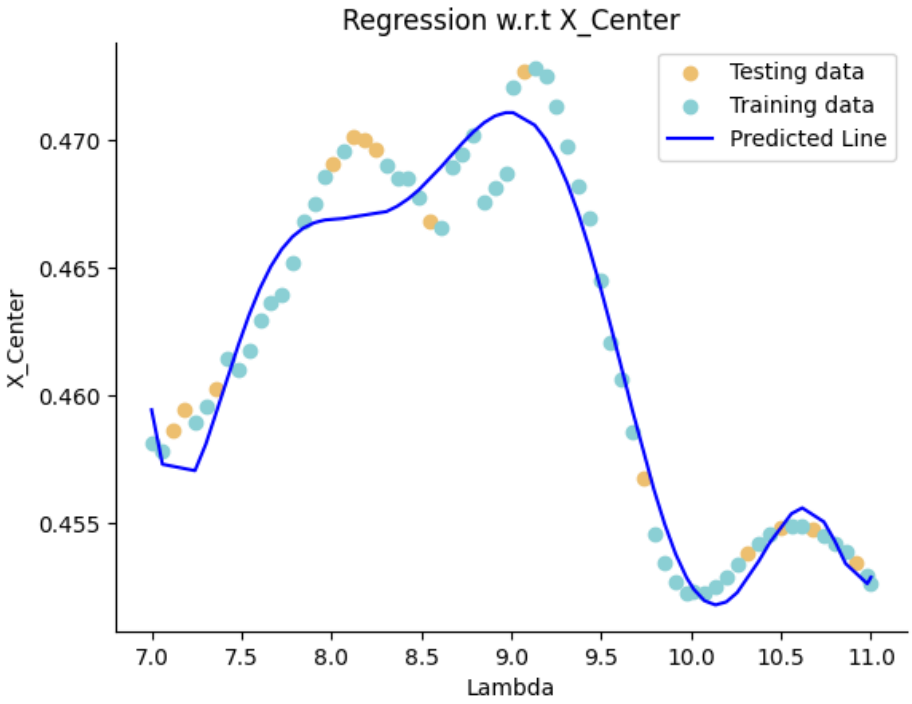


FIGURE 1.5: Polynomial Regression Model for X-center coordinate

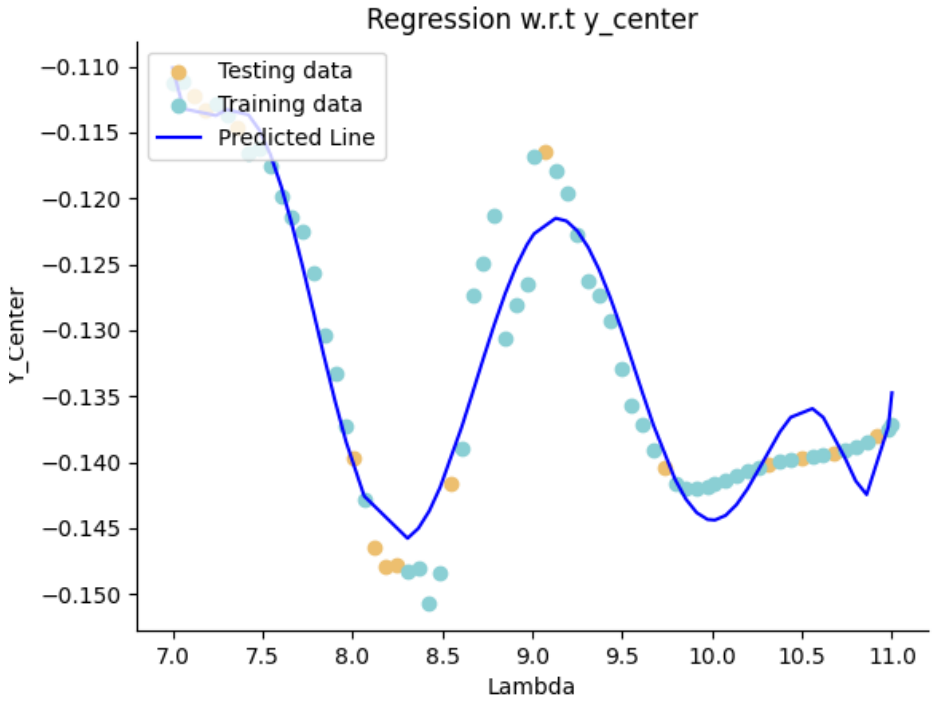


FIGURE 1.6: Polynomial Regression Model for Y-center coordinate

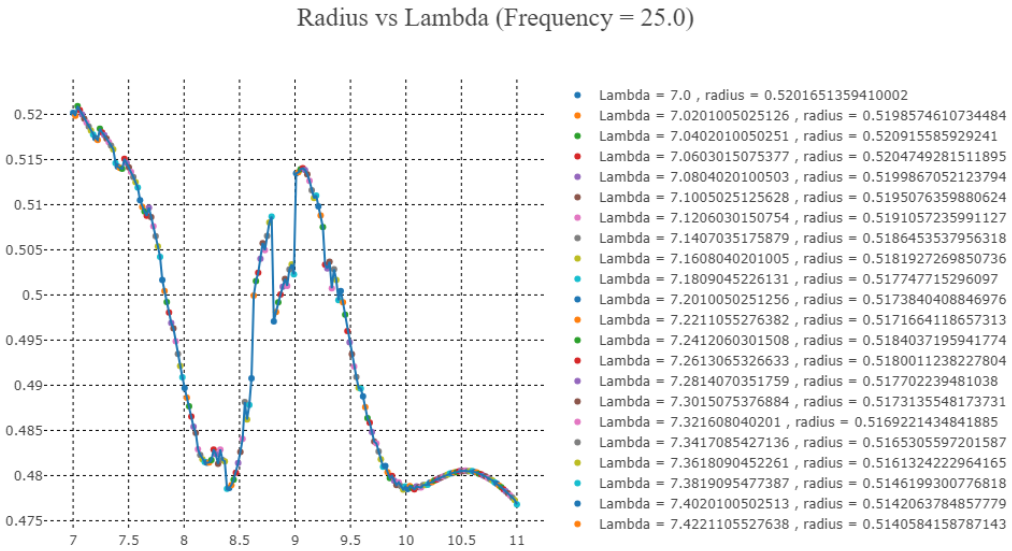


FIGURE 1.7: Radius Vs Lambda (Frequency = 25.0)

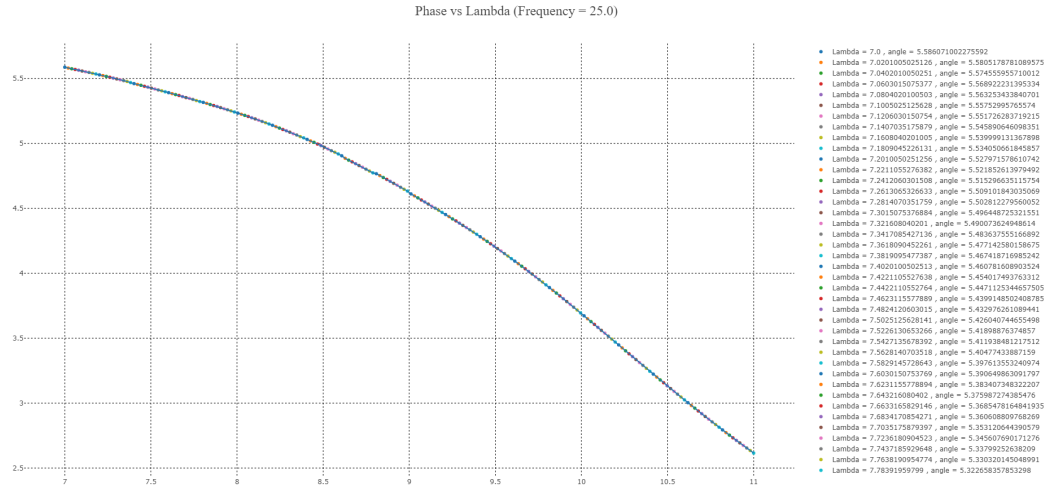


FIGURE 1.8: Phase Vs Lambda (Frequency = 25.0)

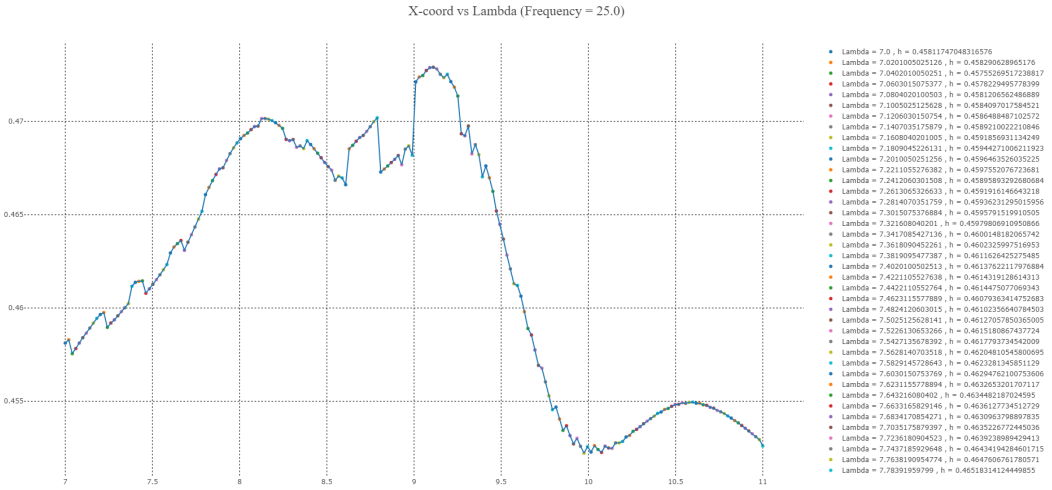


FIGURE 1.9: X-Center Vs Lambda (Frequency = 25.0)

1.2 Scope of Work

Geometrical fitting techniques are mathematical methods used to fit a set of data points with geometric shapes like ellipses and splines, which is particularly useful in applications where data exhibit cyclical or rotational patterns. This thesis aims to research such methods in order to overcome the limitations of the previous circle-fitting approach. The choice of a suitable fitting method depends on the nature of the data and the specific requirements of the analysis.

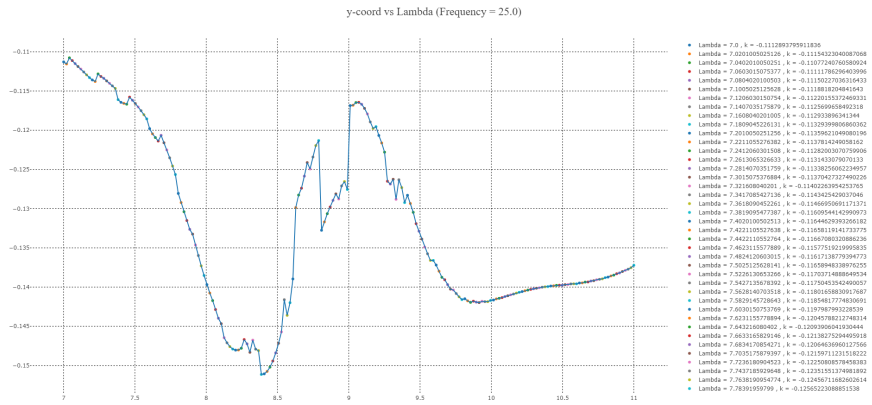


FIGURE 1.10: Y-Center Vs Lambda (Frequency = 25.0)

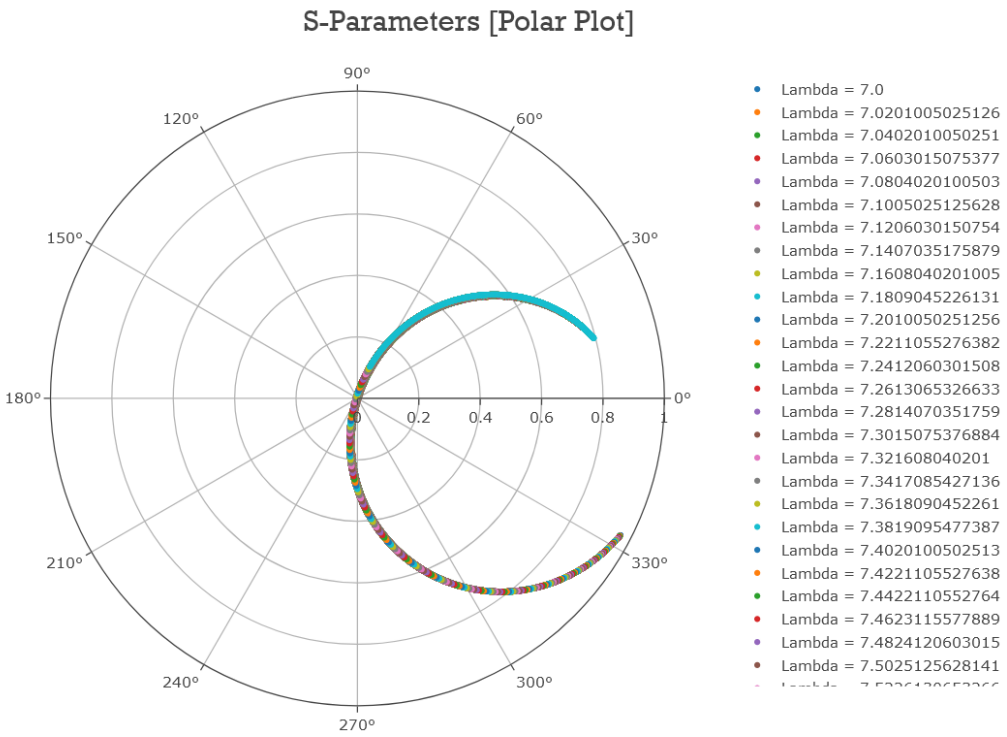


FIGURE 1.11: FRFs Visualization in Polar Chart

From fig. 1.11, it is clear that our data exhibits cyclical behavior, which is a polar representation of Frequency Response Functions (FRFs). By using geometrical fitting, the surrogate model can more effectively capture the underlying patterns and relationships in the data, leading to improved predictions and better representation of the actual system behavior. By focusing on this area, the research aims to achieve the desired smoothness of curves that can represent the simulation data accurately enough to create a highly accurate surrogate model.

1.3 Research Objective

The main objectives of this research are as follows:

- To study and implement different geometric shapes and curves to fit the simulation data with the desired smoothness of the curves of the corner frequencies.
- To evaluate results from previous and current studies and to identify if the data gathered for the study has any potential problem.
- To understand the challenges encountered during the fitting process for local neighboring frequencies.

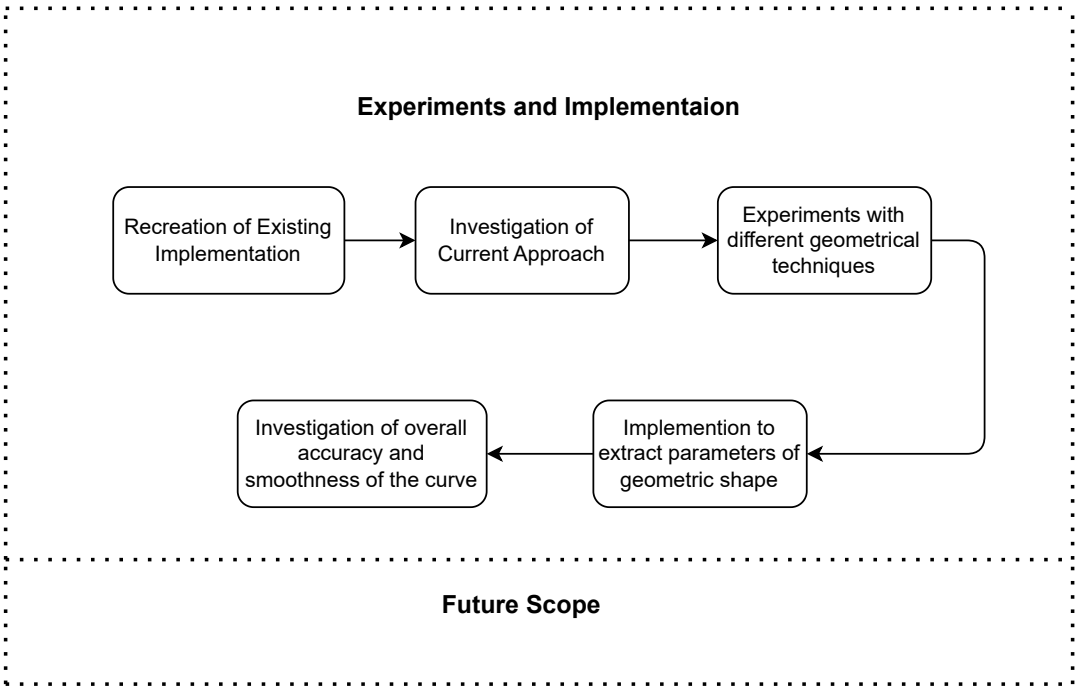


FIGURE 1.12: Overall Objective of Thesis

1.4 Report Outline

To provide a clear and organized overview of the approaches and frameworks utilized in this thesis and to make sure they are in accordance with the main goal, the chapters are organized as follows:

- **Chapter 1 - Introduction:** Introduction of the problem domain tailored to the previous research and objectives with a clear report outline.
- **Chapter 2 - Fundamentals And Literature Review:** Mathematical fundaments of geometrical fitting techniques and literature review of methods.

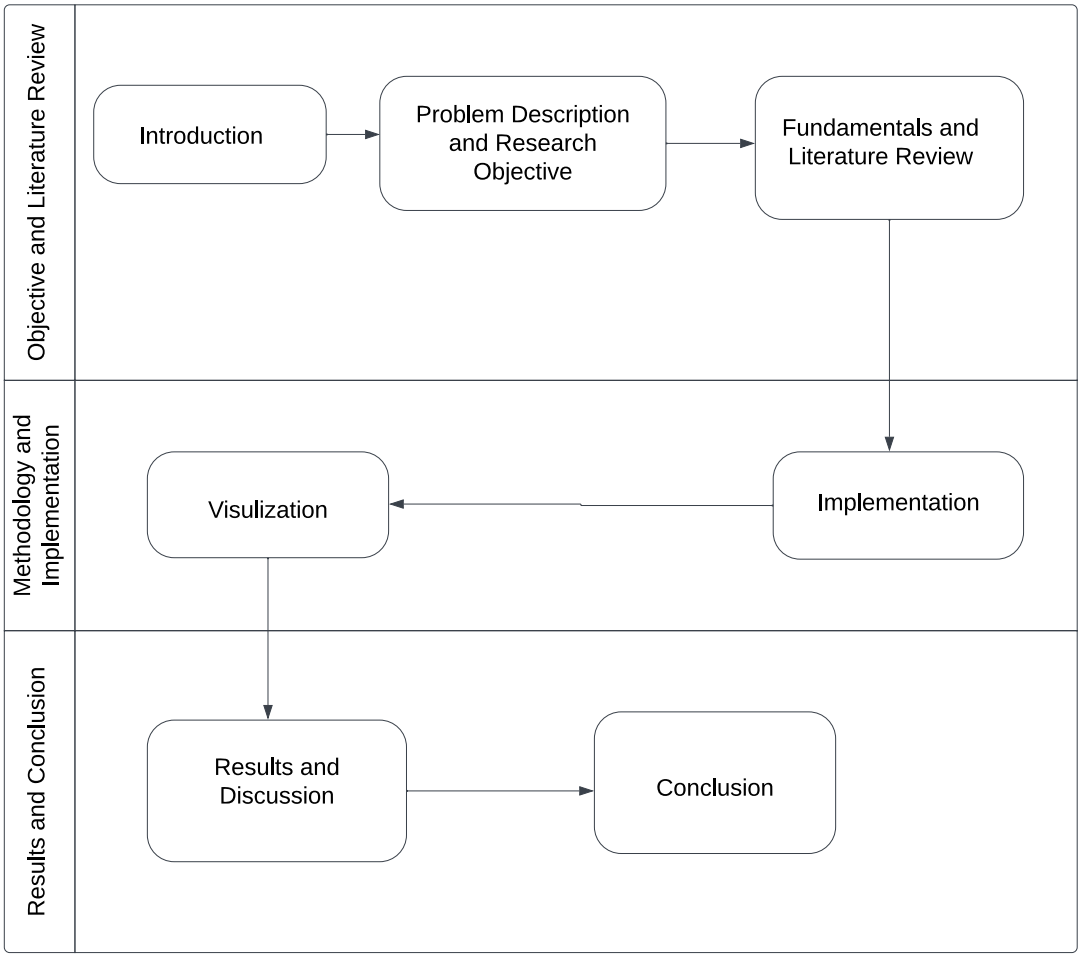


FIGURE 1.13: Thesis Report Outline

- **Chapter 3 - Methodology and Implementation:** Practical implementation of the methods and visualization of the parameters.
- **Chapter 4 - Results Discussion:** Discussion of the results from various methods employed throughout the thesis and comparison to previous studies.
- **Chapter 5 - Conclusion and Future Work:** A conclusive outcome of the research findings, their implications, and potential areas for future improvements.

Chapter 2

Fundamentals And Literature Review

The least squares approach is fundamental for creating and validating models by fitting theoretical curves to data (Širca, 2016). The nonlinear Least Squares Circle Fit offered by Scipy (Virtanen et al., 2020) was discussed in the previous study. Least squares fitting is a mathematical procedure that finds the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets of the points from the curve (S. J. Ahn et al., 2001).

2.1 Mathematical Approach to Ellipse Fitting

To fit an ellipse to a set of given points in parametric form, the methodology involves minimizing the sum of squares of the distances from these points to the 'best' ellipse. The optimization problem is formulated as a nonlinear least-squares problem that aims to find the parameters that minimize the residuals between the fitted ellipse and the data points. As the Gander et al., 1994 outlines, this process results in a system of second-order nonlinear equations for five unknowns, encompassing both the ellipse's positions and shape parameters.

2.1.1 Least Squares Ellipse Fit

SciPy python library provides 'optimize' functions for minimizing objective functions, possibly subject to constraints. 'minimize' is such a function that provides local optimization of the scaler function (ellipse equation). The method used here is Sequential Least Squares Programming (SLSQP), which was proposed by Kraft, 1988. SLSQP minimizes a function of many variables under any set of equality and inequality and limits constraints. SLSQP also handles infinite values by converting them into large floating point values.
Given a series of n data points of the form (x_i, y_i), the aim is to find the best fitting ellipse expressed by the equation 2.1,

$$\frac{(x_i - h)^2}{a^2} - \frac{(y_i - k)^2}{b^2} = 1 \tag{2.1}$$

The steps involved in implementing Least Squares Ellipse Fit (LSEF) are discussed in the algorithm (1). Fig. 2.1 depicts an example of the result obtained from the LSEF of the data points. The given data points are in the shape of an arc, which are located close to each other but not spread across the ellipse's circumference. This method best approximates these data points by minimizing the residual error between these points and the estimate.

Algorithm 1 Least Squares Ellipse Fit

Require: Series of n data points (x_i, y_i),
Ensure: Ellipse center (h, k) and semi-major axis a and semi-minor axis b

Pass the x and y data points with $initailParams$ to ***optimize.minimize*** method.
function INITIALIZEPARAMS(x_i, y_i)
    $h_0 = mean(x_i)$
    $k_0 = mean(x_i)$
    $a_0 = \sqrt{mean(\sum_{i=1}^{n}(x_i - h_0)^2)}$
    $b_0 = \sqrt{mean(\sum_{i=1}^{n}(x_i - k_0)^2)}$
    return $[a_0, b_0, h_0, k_0]$
end function
function LEASTSQUAREELLPSE(x_i, y_i)
    $initalParams = InitialParams(x_i, y_i)$
    $results = optimize.minimize(ellipseEquation, initalParams, (x_i, y_i))$
    return $results.x$ ▷ Returns the center and axis
end function

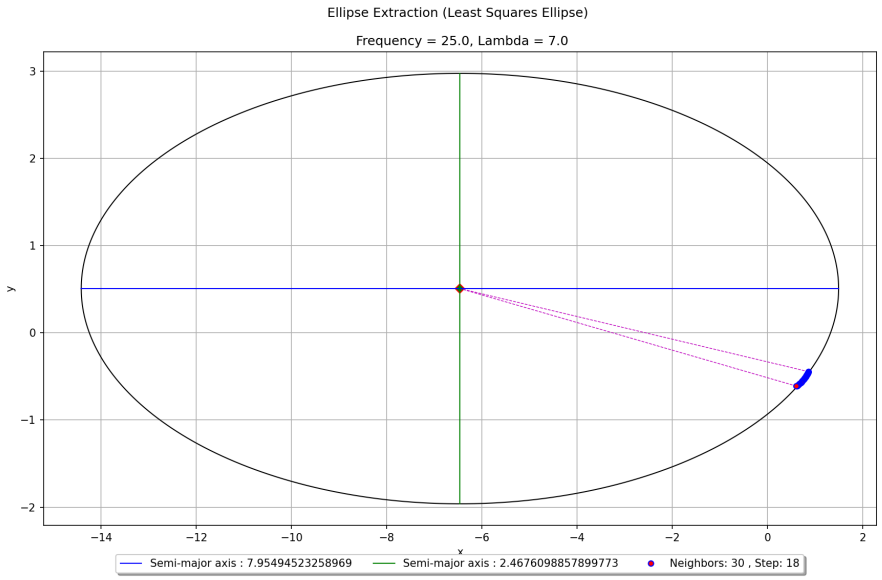


FIGURE 2.1: Least Squares Ellipse Fit

2.2 Mathematical Approaches to Spline Fitting

The spline approximation method is viable for representing large sets of points, providing accurate and smooth representations while preserving important shape characteristics, and the method based on lease squares minimization of the distance of the points from the spline function ensures smoothness of the representation (Halír and Kostková, 2000). This section describes different spline fitting approaches, such as quadratic, cubic, and quartic. All these methods use the least square approximation for fitting the spline function. SciPy python library offers a *curve_fit* function, which uses non-linear squares to fit the spline function to data.

2.2.1 Quadratic Spline

The essence of quadratic spline fitting can be encapsulated by a specific mathematical formulation. This spline, characterized by its piecewise quadratic segments, adheres to the condition of continuity up to the first derivative (Guo, 2009). The fundamental equation governing the quadratic spline can be articulated as follows:

$$S(x) = ax^2 + bx + c \tag{2.2}$$

Here a, b, and c are coefficients that are determined to ensure the smoothness and continuity of the spline. Equation 2.1 serves as the cornerstone for deriving the spline that best fits a given dataset, according to the least squares criterion.

Algorithm 2 Least Squares Quadratic Spline Fit

Require: Series of n data points (x_i, y_i),
Ensure: Quadratic Spline Coefficients a, b and c
1: Pass the x and y data points to ***optimize.curve_fit*** method.
2: **function** QUADRATICSPLINEEXTRACTION(x_i, y_i)
3: $params, covariance = optimize.curve\_fit(quadratic\_spline\_equation, x_i, y_i)$
4: **return** *params* ▷ Returns the Optimized coefficients a, b and c
5: **end function**

The algorithm 2 delineates the process for executing the Quadratic Spline Fit, while Fig. 2.2 illustrates the outcome derived from applying the *curve_fit* to the dataset.

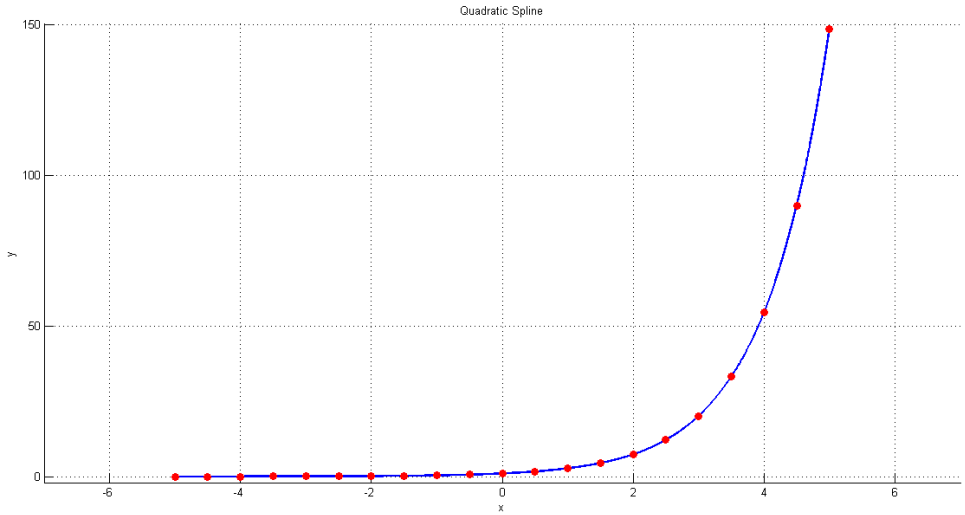


FIGURE 2.2: Quadratic Spline Fit (Mekkawy, 2023)

2.2.2 Cubic Spline

Cubic spline interpolation, a fundamental concept in numerical analysis and data fitting, can be succinctly represented by a mathematical equation. For each subinterval between data points, the cubic spline is defined by a third-degree polynomial.

$$S(x) = ax^3 + bx^2 + cx + d \tag{2.3}$$

The spline function takes the form of equation 2.3, where a, b, c, and d are coefficients that ensure smoothness and continuity at each data point. The 2.3 equation forms the backbone of cubic spline interpolation, enabling the creation of a smooth curve that not only passes through each data point but also maintains continuous first and second derivatives across the entire range (Plass and Stone, 1983).

Algorithm 3 Least Squares Cubic Spline Fit

Require: Series of n data points (x_i, y_i),
Ensure: Cubic Spline Coefficients a, b, c and d
1: Pass the x and y data points to ***optimize.curve_fit*** method.
2: $initial\_params = (0, 0, 0, 0)$
3: **function** CUBICSPLINEEXTRACTION(x_i, y_i)
4: $params, covariance = optimize.curve\_fit(cubic\_spline\_equation, x_i, y_i, initial\_params)$
5: **return** *params* ▷ Returns the Optimized coefficients a, b, c and d
6: **end function**

A step-by-step procedure to fit the cubic spline is defined in algorithm 3 and fig. 2.3 represents the cubic spline fit for a given data.

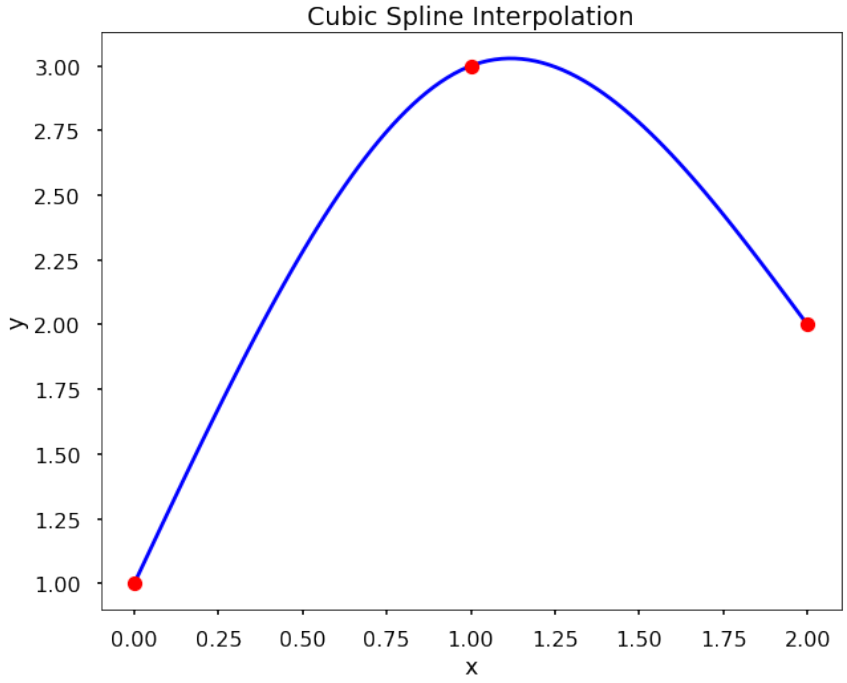


FIGURE 2.3: Cubic Spline Fit (Kong et al., 2020)

2.2.3 Circle Extraction From Cubic Spline

Differentiating individual splines for the x and y coordinates becomes necessary to determine the tangent at any point on the curve. Since the curve is defined parametrically with separate splines for x and y coordinates, the tangent vector at any point can be obtained by differentiating these splines with respect to the parameter. This approach is crucial for the precise calculation of tangents, especially in complex curves where direction changes are significant. The differentiation of these separate splines provides the necessary components of the tangent vector in the x and y directions (Ghys et al., 2013). Let $\gamma(s)$ be a regular parametric curve, where s is a vector.

$$s = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.4}$$

The tangent $T(s)$ at point t is the derivative of $\gamma(t)$ w.r.t to t,

$$T(s) = \begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} \tag{2.5}$$

Using the tangent $T(s)$ we can obtain normal unit vector $N(t)$ at point t as,

$$N(t) = \frac{1}{\|\gamma'(t)\|} \begin{bmatrix} -y'(t) \\ x'(t) \end{bmatrix} \tag{2.6}$$

Now, by taking the derivative of tangent and using equation 2.6, the following relation is obtained,

$$T'(s) = k(s)N(s), R(s) = \frac{1}{|k(s)|} \tag{2.7}$$

where $k(s)$ is the *signed curvature* and $R(s)$ is the *radius of curvature* at each point for which s is composed. By simplifying the equation 2.7 and using the equation 2.6 the *radius of curvature* and center $Q(t)$ at point t is defined as,

$$R(t) = \left| \frac{(x'(t)^2 + y'(t)^2)^{3/2}}{x'(t)y''(t) - x''(t)y'(t)} \right| \tag{2.8}$$

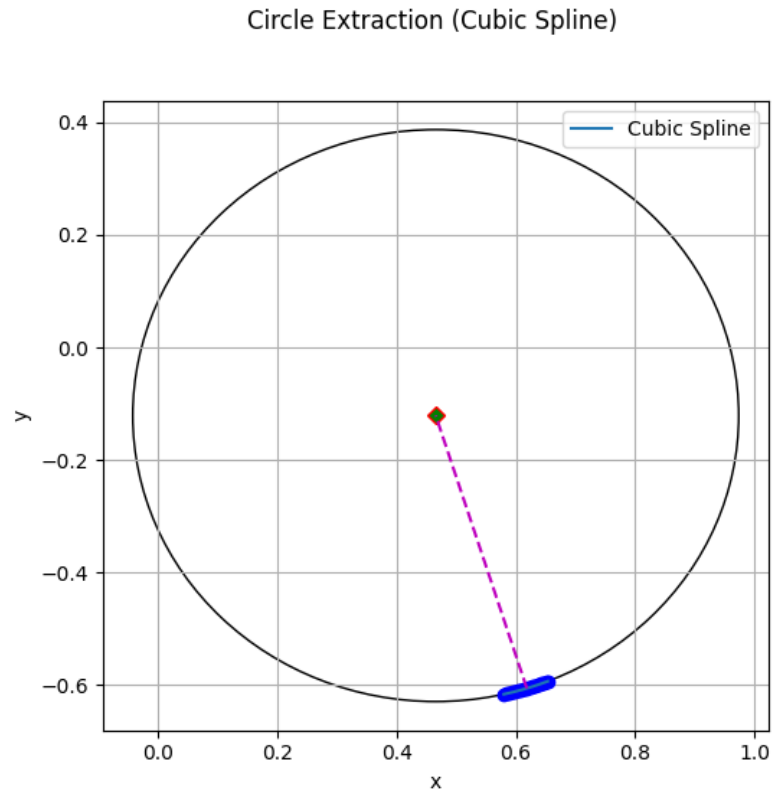
$$Q(t) = \gamma(t) + \frac{1}{k(t)}N(t) \tag{2.9}$$

The algorithm 4 depicts the analytical approach to finding the radius and center of

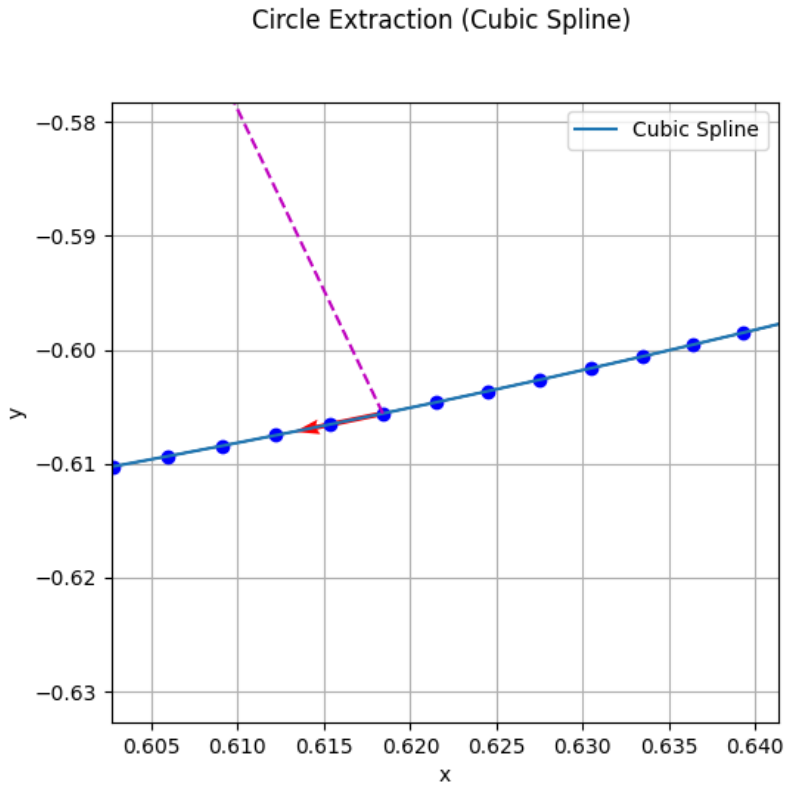
Algorithm 4 Circle Extraction using Cubic Spline

Require: Series of n data points (x_i, y_i),
Ensure: Circle center (h, k) and radius r
Pass the x and y data points to ***interpolation.CubicSpline*** method.
function GETCUBICSPLINE(x_i, y_i)
    $x(t) = CubicSpline(t, x_i)$
    $y(t) = CubicSpline(t, y_i)$
    return x_t, y_t
end function
function EXTRACTCIRCLEPARAMETERS(x_i, y_i)
    $x(t), y(t) = getCubicSpline(x_i, y_i)$
    $x'(t) = x(t).derivative()$
    $y'(t) = y(t).derivative()$
    $T(t) = \frac{x'(t)}{y'(t)}$
    $N(t) = \frac{T(t)}{norm(T(t))}$
    $x''(t) = x'(t).derivative()$
    $y''(t) = y'(t).derivative()$
    $curvature = \frac{x'(t)y''(t) - x''(t)y'(t)}{(x'(t)^2 + y'(t)^2)^{1}.5}$
    $raduis = \frac{1}{abs(curvature)}$
    $center = [x(t), z(t)] + \frac{1}{curvature}N(t)$
    return $center, radius$
end function

the local circle anywhere in the Cartesian plane at a given point. Fig. 2.4 shows tangent directions and center direction along with the Circle passing through the spline curve.



(A) Original View



(B) Zoomed View

FIGURE 2.4: Circle Extraction Using Spline

2.2.4 Quartic Spline

The quartic spline is lauded for its efficiency in approximation, particularly due to its higher-degree nature, which circumvents the creation of corners at the joints, thus requiring fewer data points to achieve a given level of accuracy compared to higher-order methods (Dubey and Paroha, 2014).
The quartic spline is a fourth-degree polynomial, and equation 2.10 represents the spline equation with five unknown coefficients a, b, c, d, and e. The algorithm 5 shows the least square quartic spline fit approach to solve five unknown coefficients. Fig. 2.5 shows the quartic spline and its behavior.

$$S(x) = ax^4 + bx^3 + cx^2 + dx + e \tag{2.10}$$

Algorithm 5 Least Squares Quartic Spline Fit

Require: Series of n data points (x_i, y_i),
Ensure: Quartic Spline Coefficients a, b, c, d and e
1: Pass the x and y data points to ***optimize.curve_fit*** method.
2: $initial\_params = (0, 0, 0, 0, 0)$
3: **function** QUARTICSPLINEEXTRACTION(x_i, y_i)
4: $params, covariance = optimize.curve\_fit(quartic\_spline\_equation, x_i, y_i, initial\_params)$
5: **return** *params* ▷ Returns the Optimized coefficients a, b, c, d and e
6: **end function**

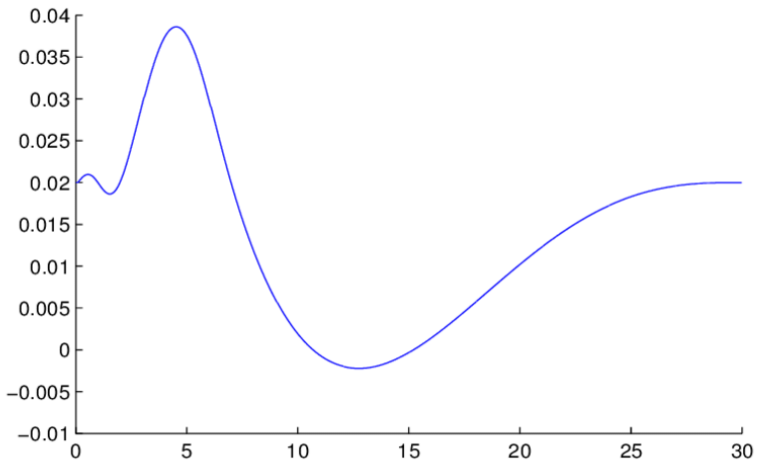


FIGURE 2.5: Quartic Spline (Hagan, 2005)

2.3 Summary

This chapter delves deeply into the least squares fitting techniques with a keen emphasis on ellipse and spline fitting, a cornerstone in the realm of data modeling. It commences with an overview of the nonlinear Least Squares Circle Fit from Scipy, highlighting its indispensable role in aligning theoretical curves with empirical data. The narrative then transitions to a comprehensive discussion on ellipse fitting methodologies, specifically focusing on the minimization of squared distances, a technique pivotal in deducing the optimal parameters that characterize the best-fit ellipse.

The exposition extends to spline fitting, exploring quadratic, cubic, and quartic methods, each underpinned by the least squares approximation principle to ensure a smooth and accurate representation of data. In particular, the cubic spline segment illuminates the intricate process of computing tangents and curvatures by differentiating the splines associated with x and y coordinates, underscoring the significance of this technique in rendering precise geometric interpretations of complex data sets.

A section on quartic splines is especially noteworthy, demonstrating their proficiency in approximation due to their higher polynomial degree, which adeptly mitigates the formation of corners at data joints. This attribute significantly reduces the number of data points needed to maintain a specified level of accuracy, showcasing the efficiency of quartic splines over other higher-order spline methods.

Concluding the chapter, this chapter presents a suite of algorithms designed for fitting these splines and for the meticulous extraction of circle parameters from cubic splines, providing a clear algorithmic pathway from data points to geometric fit. Accompanying illustrations vividly depict the results of these fitting processes, offering a visual understanding of the practical applications and effectiveness of these advanced mathematical tools in data modeling and curve fitting.

Chapter 3

Methodology and Implementation

3.1 Introduction to Methodology

The Methodology chapter commences with a comprehensive exploration of the techniques and processes harnessed to refine the precision of geometric shape fitting to data points. This research builds upon the foundational work of Dasi, 2023, propelling the application of least squares fitting methods to enhance the smoothness and accuracy of curve approximations. Delving into the intricacies of spline fitting, the methodology juxtaposes quadratic, cubic, and quartic splines, elucidating their unique contributions to the field. With a blend of mathematical rigor and computational finesse, the chapter delineates the algorithmic pathways and optimization strategies employed while highlighting the utilization of sophisticated software tools like SciPy. The narrative reveals the systematic implementation of these approaches, shedding light on the precise visualization techniques adopted to capture the cyclical patterns inherent in the data. This exploration is not merely a technical recount; it is an insightful journey into the challenges encountered, the innovative solutions devised, and the thorough evaluation processes that underscore the quest for a superior model of geometric data representation. The culmination of this chapter sets a precedent for future scholarly endeavors, aiming to inspire continued advancements in data modeling and curve fitting.

3.2 Geometrical Fitting Techniques

The basic principle of geometric fitting involves finding the best geometric representation (such as a curve, circle, or other shape) that fits a given set of data points. Geometric fitting seeks to minimize the orthogonal distance to the curve, providing a more aesthetic and geometrically accurate result (S.-J. Ahn, 2008). Geometric fitting methods are designed to handle and fit geometric shapes to scattered data points, and they often involve constrained numerical optimization to find the best-fitting geometric representation (Arlinghuas, 1994).

This part of the thesis unveils the theoretical underpinnings of fitting data points to geometric shapes such as ellipses and various types of splines. It delves into the

nuances of ellipse fitting through the Least Squares Ellipse Fit (LSEF) method and explores spline fitting modalities encompassing quadratic, cubic, and quartic forms. Each fitting approach is meticulously unpacked, revealing its mathematical foundations and the computational algorithms that bring these theories to life. Special emphasis is placed on the use of the SciPy library, a pivotal tool that enhances the robustness and efficiency of these fitting techniques.

3.2.1 Tools And Technology

- **Python:** Python programming is highly advantageous for implementing geometric fitting techniques due to its simplicity and power. Python's syntax is clear and readable, which makes the coding process more intuitive, especially for mathematical operations. It boasts a rich ecosystem of libraries such as NumPy for numerical computation, SciPy for optimization, and Matplotlib for visualization, streamlining complex tasks like spline fitting and curve optimization. Furthermore, Python's extensive community support and open-source nature mean that researchers can leverage shared knowledge and resources, facilitating collaborative development and innovative solutions. This robust support and the language's flexibility make Python ideal for implementing and experimenting with geometric fitting techniques in data analysis and computational research.

- **Matplotlib:** Matplotlib stands out for its comprehensive and detailed approach to creating static, animated, and interactive visualizations in Python. It is a complete toolkit for creating various 2-dimensional and 3-dimensional plots. This library is used for plotting ellipses and spline curves in this thesis.

- **Plotly:** Plotly, in contrast, excels in crafting dynamic and interactive visualizations. Its strength lies in its capacity to create engaging, web-friendly graphics that invite user interaction. This interactivity is pivotal for exploratory data analysis, enabling users to delve deeper into datasets and uncover insights that static graphs might not reveal. Plotly's intuitive design and advanced features make it particularly suitable for modern data visualization tasks where user engagement and clarity of complex data patterns are key, and this is the reason why this library is widely used throughout the thesis to plot the results.

- **SciPy:** SciPy is highly suited for least squares fitting, particularly in the context of its *curve_fit* and *minimize* functions, as well as its *CubicSpline* class. The *curve_fit* function simplifies the process of fitting a curve to a dataset, making it ideal for various types of spline fitting. The *minimize* function provides a versatile approach to optimization, crucial for refining least squares models. Such an approach, SLQSP, is used in this thesis; besides, there are more such approaches provided by the SciPy library. Additionally, the *CubicSpline* class in SciPy is tailored for creating smooth cubic splines, making it a powerful tool

for precise curve fitting. These features make SciPy a robust and reliable choice for complex data-fitting tasks in scientific computing. These functions are used in this thesis to facilitate the geometric fitting techniques discussed in chapter 2.

3.3 Implementation of Algorithms

In this section, I will detail the application of least squares methods for curve fitting, focusing on ellipse fitting and spline techniques, including quadratic, cubic, and quartic splines. This section will thoroughly discuss the algorithmic execution of these methods, elaborating on the specific processes and computational steps. Emphasis will be placed on the mathematical formulation and practical application, highlighting how these algorithms are tailored to extract and optimize geometric shape parameters from the given datasets. This exploration will provide insights into the technical aspects of these fitting techniques and their role in enhancing the accuracy and smoothness of the modeled curves.

3.3.1 Extraction of Neighboring Frequencies

The methodology adopted in this research for neighbor frequency extraction closely aligns with the process described in Dasi, 2023's thesis. Utilizing the *extract_neighbours* function, the methodology involves an initial duplication of the data frame list to preserve the original data integrity. The process then identifies a specific data frame using a predefined index and extracts its x and y coordinates. A key aspect of this approach is implementing a refined filtering system, which adeptly adjusts for potential edge cases. This ensures the selection of a precise number of neighboring frequencies, optimizing the accuracy of the data selection process. This technique plays a pivotal role in the research, facilitating meticulous data management and extracting pertinent frequency-related information, thereby bolstering the overall analytical rigor of the study. The snippet 3.1 shows general code that takes any number of neighbors into consideration.

```
def extract_neighbours(self, idx):
    orgDFs = self.orgDF_list[:]  # copy list into a new variable
    _pick = orgDFs[idx]
    extract_xy = _pick[-2:]  # type list
    xy_tuple = tuple(extract_xy)
    # copy the list into a new variable so we don't change it
    main_list = orgDFs[:]
    _res_list = self.filter_freq(main_list, idx)

    # update the index with the new working list
    _new_idx = _res_list.index(_pick)
    log.debug(f"extract_neighbours: Neigh Count --> {self._neigh}")
    left_slice = _new_idx - self._neigh//2  # start of the range
```

```
    # Account for edges for the left slice
    # max((0, left_sclice), (len(list) - <req.num.of.neightbours>)),
    # min(<idx>, <idx-1>)
    left_slice = min(max(0, left_slice), len(_res_list) - self._neigh)

    # extract the right slice range
    right_slice = left_slice + self._neigh  # end of range
    return _res_list[left_slice:right_slice], xy_tuple
```

LISTING 3.1: Extraction of Neighboring Frequencies

3.3.2 Least Square Ellipse Fit

As discussed in the algorithm 1, a least square method to fit the ellipse for a given set of complex coordinate points is utilized. This approach minimizes residual errors and takes the best approximation of the ellipse fit for the given data points. SLSQP optimizer is used to handle infinite values. In addition, the method calculates the center of the ellipse, semi-major axis, and semi-minor axis. The section demonstrates a step-by-step approach with detailed code implementation for the Least Squares Ellipse Fit. This methodical exposition begins with the objective of an Ellipse, followed by a systematic breakdown of the code. It covers the initialization of parameters, the application of the optimization function, and the nuances of employing the Sequential Least Squares Programming (SLSQP) method. Each segment of the code is elucidated to provide a clear understanding of its role and functionality in achieving the ellipse fitting,

- **Objective of Ellipse:** The code snippet 3.2 defines the *ellipse_objective* function, crucial for the Least Squares Ellipse Fit algorithm. This function calculates the sum of squared residuals between the data points (x, y) and an ellipse defined by parameters a, b, h, and k. Parameters a and b represent the semi-major and semi-minor axes, while h and k denote the ellipse's center coordinates.

```
        def ellipse_objective(self, params, *args):
            x, y = args
            a, b, h, k = params
            return np.sum((((x - h) / a)**2 +
                    ((y - k) / b)**2 - 1)**2)

```

LISTING 3.2: Objective of Ellipse

- **Initialize Parameters:** The *initialize_params* function in the code snippet is designed to set initial values for the parameters of an ellipse being fitted to a set of data points. It calculates the initial estimates for the center of the ellipse ($h_initial$, $k_initial$) as the mean of the x and y data points, respectively. The initial estimates for the semi-major and semi-minor axes ($a_initial$, $b_initial$) are

computed as the square roots of the mean squared distances of the x and y data points from their respective means. This approach provides a starting point for the ellipse fitting process, ensuring that the initial parameters are grounded in the data's distribution.

```
        def ellipse_objective(self, params, *args):
            h_initial = np.mean(x_data)
            k_initial = np.mean(y_data)
            a_initial = np.sqrt(np.mean((x_data - h_initial)**2))
            b_initial = np.sqrt(np.mean((y_data - k_initial)**2))

```

LISTING 3.3: Objective of Ellipse

- **Ellipse Fitting:**
  - LOC(1-15) from code block 3.4 is the data preparation phase. It involves copying and restructuring the dataset from *DataParser*, focusing on relevant columns such as frequency, lambda, and S-parameters.
  - LOC(16-34) Conducts a detailed extraction and filtering process. It removes duplicates and sorts relevant data points, ensuring a clean dataset for analysis. We have 1001 frequencies in the spectrum of 25 - 35 GHz.
  - LOC(48-62) Iterates through the filtered dataset, extracting neighboring frequencies and coordinates. This step is crucial for gathering adjacent data points relevant to each frequency. We do this for 200 lambda parameters from 7 until 200.
  - LOC(64-83) Implements the ellipse fitting process. It involves initializing parameters and then applying the minimize function with the ellipse_objective method to determine the ellipse's parameters, such as center(h, k), semi-major(a), and semi-minor(b) axes.
  - LOC(84-87) Compiles the results into a structured format. It organizes the calculated ellipse parameters alongside corresponding frequency and lambda values into a data frame, setting the stage for regression modeling and further analysis.

```
    def process_ellipse_extraction(self):
        # create a copy of the dataset
        frf_df_cp = DataParser().get_freq_data()
        frf_df6 = frf_df_cp.copy(deep=True)

        # Pick each row of these columns containing frequency,
            lambda, S1 real, S1 imaginary
        # and form a list of each row containing these
            column values.

        self.orgDF_list = frf_df6.apply(lambda row:
                                        [row["Frequency"]]
```

```
                                    + [row["Lambda"]]
                                    + [row["S11_Real[RE]"]]
                                    + [row["S11_Imaginary[Im]"]],
                                    axis=1).to_list()
        # extract the freq. to a list
        frq_list = df_elements["Frequency"].to_list()
        # remove the duplicate freq.
        _frqs = list(dict.fromkeys(frq_list))

        for i, frq in enumerate(_frqs, start=1):
            # For the picked frequency, extract all
                the matching rows
            if not frf_df6.empty:
                result_df = frf_df6[frf_df6['Frequency'].isin([frq
    ])]
            # working dict after the data extraction-based
                on the freq.
            wrk_dict = result_df.apply(lambda row:
                                    [row["Frequency"]]
                                    + [row["Lambda"]]
                                    + [row["S11_Real[RE]"]]
                                    + [row["S11_Imaginary[Im]"]],
                                    axis=1).to_dict()
            """
            - For each "lambda" value in the working data frame,
                pick the index and extract the adjacent data
                points for the picked index, which in turn
                contains the adjacent neighboring frequency
                data.
            - Pass the extracted real and imaginary values as
                tuples to plot the ellipse that passes through
                these XY points.
            - With the ellipse fit, we obtain the ellipse
                "Center", "Semi Major Axis," and
                "Semi Minor Axis" information.
            """
            for k, item in wrk_dict.items():
                # extract the neighbors
                info, xy_tuple = self.extract_neighbours(k)
                frequency_value = item[0]
                lambda_value = item[1]

                # extract the coordinate tuples from the info
                for item in info:
                    # 0:Freq and 1 Lambda from 2:Coordinates
                    coord = tuple(item[2:])
                    # extract 1st element of the tuple into list
                    self.x.append(coord[0])
                    # extract 2nd element of the tuple into a list
                    self.y.append(coord[1])
                    freq_pts.append(item[0])

```

```
64                  initial_params =
65                      self.initialize_params(self.x, self.y)
66
67                  # Now pass the extracted x-y points and initial
68                      params and calculate the
69                  # center, semi-major axis (a), and semi-minor axis
    (b)
70                  result = minimize(self.ellipse_objective,
71                                    initial_params,
72                                    args=(self.x, self.y),
73                                    method = 'SLSQP')
74                  a, b, h, k = result.x
75
76                  lambda_list.append(lambda_value)
77                  semi_major_axis_list.append(a)
78                  semi_minor_axis_list.append(b)
79                  h_list.append(h)
80                  k_list.append(k)
81                  # contains list of xy tuples
82                  xy_list.append(xy_tuple)
83                  _freqs.append(frq)
84              # create dataframe with the desired attributes
85              _df = pd.DataFrame(list(zip(_freqs, lambda_list,
86                          semi_major_axis_list, semi_minor_axis_list
    ,
87                          h_list, k_list, xy_list)),
88                          columns=["Frequency", "Lambda",
89                          "Semi_Major_Axis", "Semi_Minor_Axis",
90                          "x_center", "y_center",
91                          "coordinates"])
92
```

LISTING 3.4: Least Square Ellipse Extraction

Now, if the variation in the ellipse parameters versus lambda parameter range is smooth enough, it can be concluded that the least square ellipse fit is best suited for our data. The figures 3.1, 3.2, 3.3, 3.4 illustrates how extracted ellipse parameters vary as a function of lambda for a given frequency of 25 MHz. The figures clearly show that the curves could be smoother. It is unclear as to why this data is so noisy, but it is clear that initial data has quite an influence on the results. So, we need another approach, which will be discussed in the next section of this chapter.

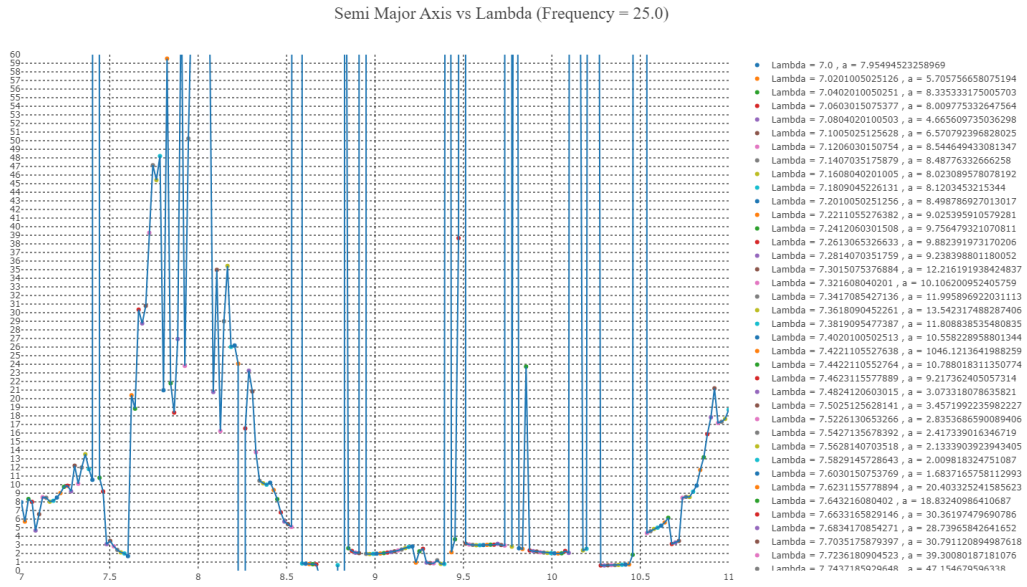


FIGURE 3.1: Semi-Major Axis Vs Lambda (Frequency = 25.0)

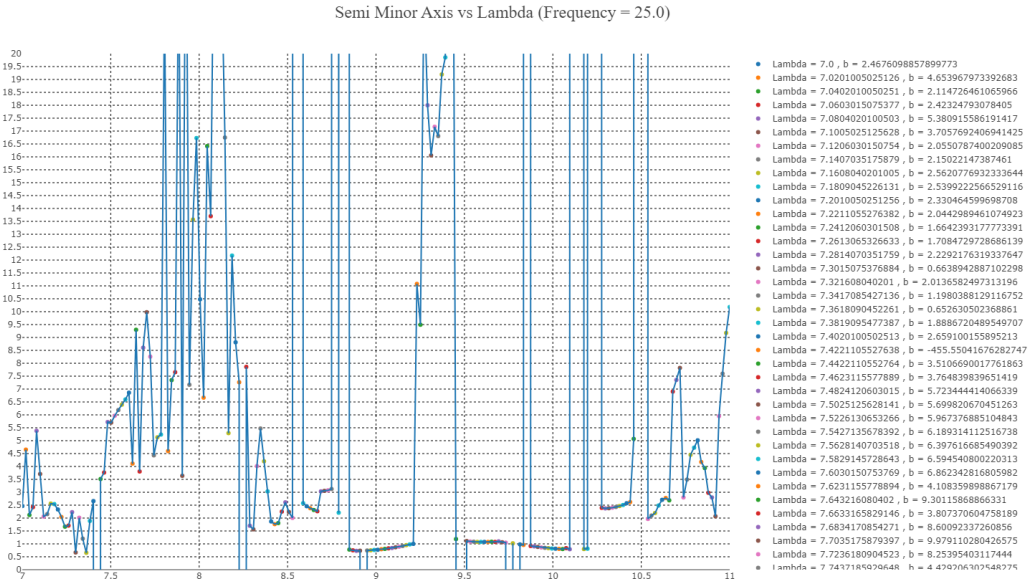


FIGURE 3.2: Semi-Minor Axis Vs Lambda (Frequency = 25.0)

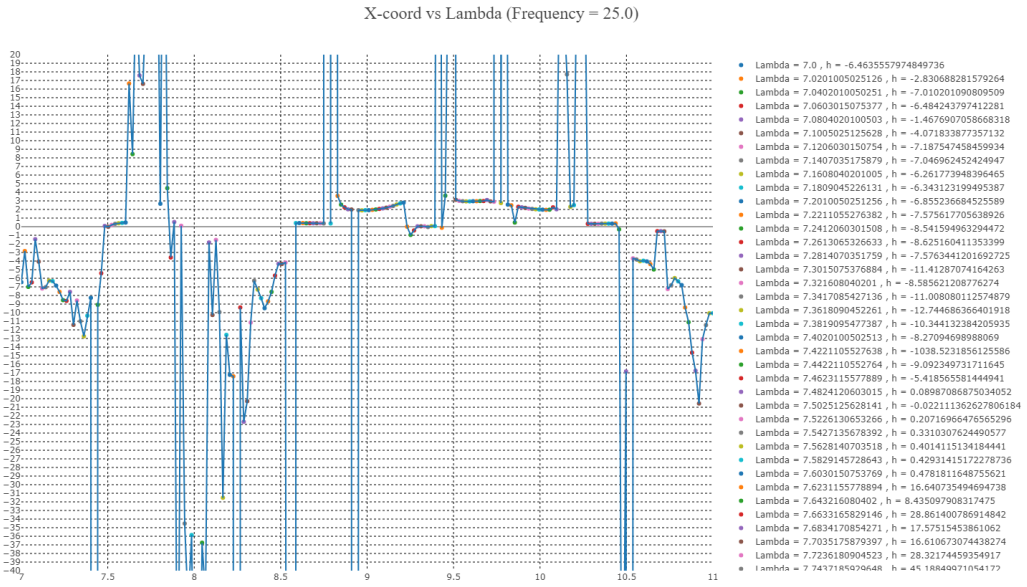


FIGURE 3.3: X-Center Vs Lambda (Frequency = 25.0)

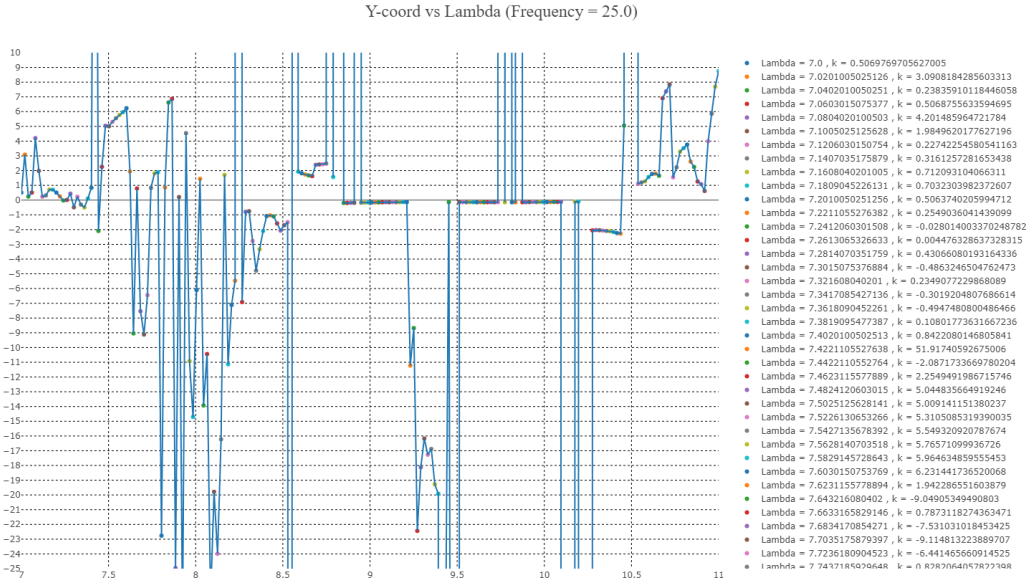


FIGURE 3.4: Y-Center Vs Lambda (Frequency = 25.0)

3.3.3 Quadratic Spline Fit

This section elaborates on the implementation of the least-square quadratic spline fit approach as outlined in algorithm 2, utilizing the *curve_fit* function from the SciPy library. It includes a detailed explanation of the algorithm's procedure, starting with the initial data preparation and progressing through the intricacies of applying the

curve_fit method. This comprehensive overview provides insights into the technical aspects of the quadratic spline fitting process and highlights the application of SciPy's capabilities in optimizing the spline fit to the data.

- **Quadratic Spline:** The code snippet defines a function quadratic_spline, representing the equation for a quadratic spline. The function takes an independent variable x and three coefficients a, b, and c. It calculates the value of the spline at x by evaluating the quadratic equation $ax^2 + bx + c$. This function forms the core of the quadratic spline fitting process, allowing for the determination of a curve that best fits a given set of data points based on these coefficients.

```
        def quadratic_spline(self, x, a, b, c):
            """Quadratic spline equation."""
            return a * x**2 + b * x + c

```

LISTING 3.5: Quadratic Spline Equation

- **Quadratic Spline Extraction:** The general part of data preparation, extraction, and filtering is the same as in code snippet 3.4, with only the core part of data fitting changed based on the method being implemented.
  - LOC(18-29) from code block 3.6 shows that we use an initial guess for the quadratic spline coefficients set as [0, 0, 0]. The curve_fit function from SciPy is then utilized to fit a quadratic spline to the extracted x and y coordinate data. This fitting process involves the function quadratic_spline, which defines the quadratic equation from snippet 3.5. The curve fitting yields optimized parameters (coefficients) for the quadratic spline, represented by a_opt, b_opt, and c_opt. These coefficients are the result of minimizing the squared differences between the spline and the data points.

```
        def process_spline_extraction(self):
            for k, item in wrk_dict.items():
                # extract the neighbors
                info, xy_tuple = self.extract_neighbours(k)
                frequency_value = item[0]
                lambda_value = item[1]

                # extract the coordinate tuples from the info
                for item in info:
                    # 0:Freq and 1 Lambda from 2:Coordinates
                    coord = tuple(item[2:])
                    # extract 1st element of the tuple into list
                    self.x.append(coord[0])
                    # extract 2nd element of the tuple into a list
                    self.y.append(coord[1])
                    freq_pts.append(item[0])

```

```
                 # Initial guess for parameters
                initial_guess = [0, 0, 0]
                """Fit a quadratic spline to the given
                data points (x, y)."""

                # Perform the curve fitting
                params, covariance = curve_fit(
                    self.quadratic_spline,
                    self.x, self.y, p0=initial_guess)

                # Extract the optimized coefficients
                a_opt, b_opt, c_opt = params

                lambda_list.append(lambda_value)
                coeff_a_list.append(a)
                coeff_b_list.append(b)
                coeff_c_list.append(c)
                # contains list of xy tuples
                xy_list.append(xy_tuple)
                _freqs.append(frq)

            # create dataframe with the desired attributes
            _df = pd.DataFrame(list(zip(_freqs, lambda_list,
                        coeff_a_list, coeff_b_list, coeff_c_list,
                        xy_list)),
                        columns=["Frequency", "Lambda",
                        "Coeff_a", "Coeff_b", "Coeff_c",
                        "coordinates"])

```

LISTING 3.6: Least Square Quadratic Spline Extraction

The analysis of the figures 3.5, 3.6, 3.7 indicates that the least square quadratic spline fit may not be adequately suited for the intended purpose, as evidenced by significant fluctuations and spikes in the data relative to adjacent lambda values. This observation suggests a potential limitation of the quadratic spline in capturing the nuances of the dataset. Consequently, the exploration will progress to consider a higher-order polynomial, specifically a third-order (cubic) spline, which may offer improved fitting characteristics. The following section will consider the implementation and efficacy of this cubic spline approach in addressing the identified discrepancies.

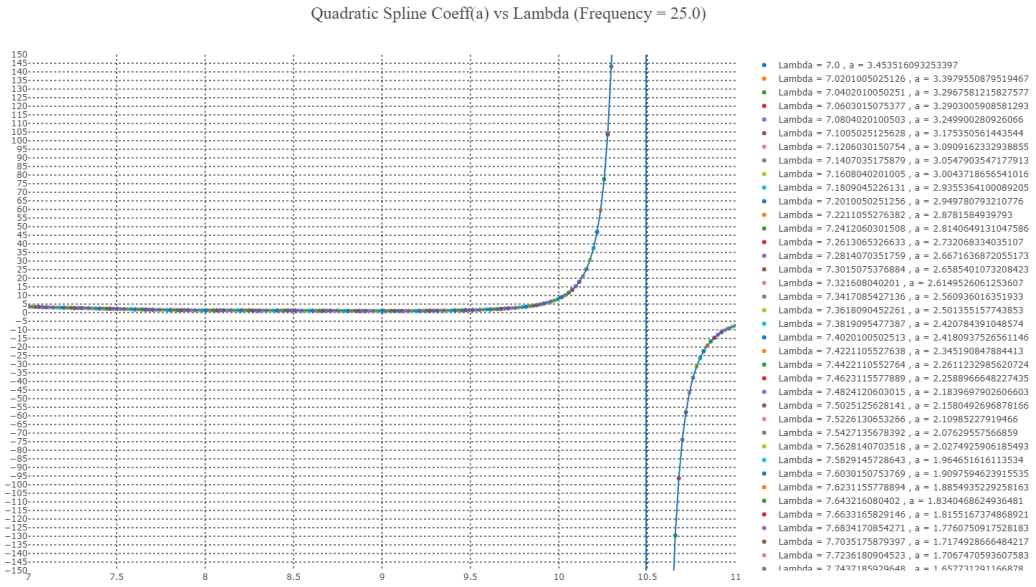


FIGURE 3.5: Quadratic Spline Coefficient(a) Vs Lambda (Frequency = 25.0)

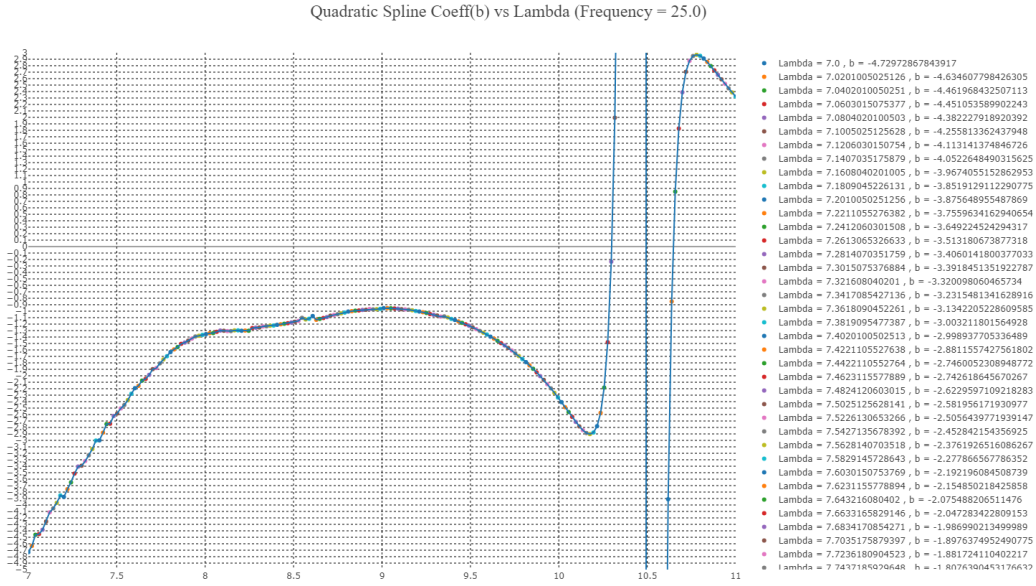


FIGURE 3.6: Quadratic Spline Coefficient(b) Vs Lambda (Frequency = 25.0)

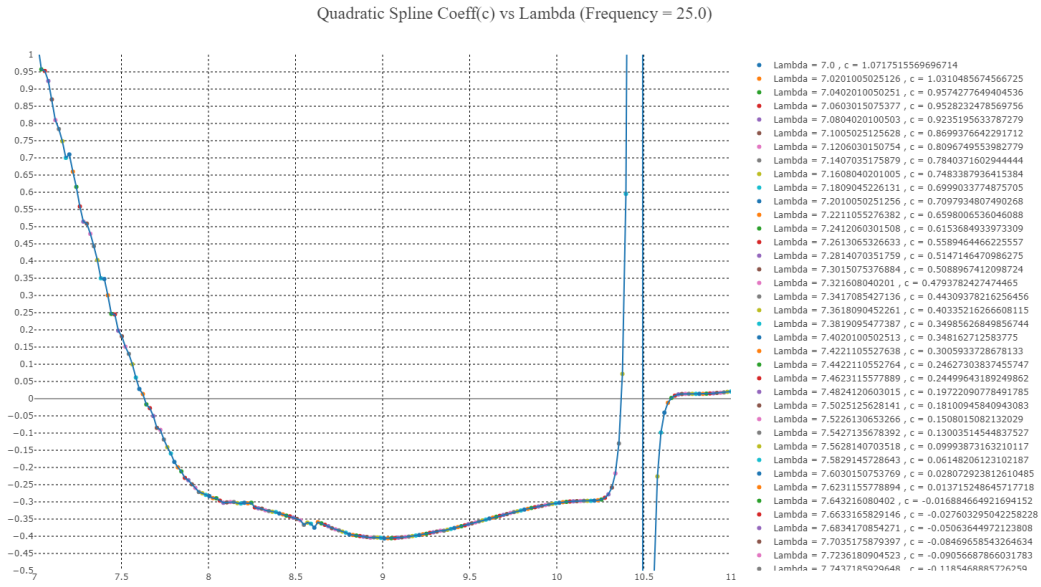


FIGURE 3.7: Quadratic Spline Coefficient(c) Vs Lambda (Frequency = 25.0)

3.3.4 Cubic Spline Fit

The following section outlines a structured approach with code implementation for the Least Squares Cubic Spline Fit, as mentioned in algorithm 3. The explanation delves into the intricacies of applying this higher-order polynomial fitting, using specific coding examples to illustrate the methodology. This approach not only enhances the clarity of the implementation but also showcases the progression from theory to practical application in spline fitting. Some coding aspects are the same as **LOC(1-62)** from snippet 3.4; only the core is different.

- **Cubic Spline:** The cubic_spline function in the code 3.7 is designed to compute the value of a cubic spline at given points. It first converts the input x into a NumPy array to facilitate element-wise operations. The function then evaluates the cubic spline equation $a + bx + cx^2 + dx^3$, where a, b, c, and d are the spline coefficients. This equation represents a cubic polynomial, the form of which is essential for fitting a cubic spline to a set of data points in curve-fitting processes.

```
1         def cubic_spline(self, x, a, b, c, d):
2             x = np.array(x)
3             """ Cubic Spline Equation. """
4             return return a + b * x + c * x**2 + d * x**3
5
```

LISTING 3.7: Cubic Spline Equation

- **Cubic Spline Extraction: LOC (2-30)** section of code 3.8 involves fitting a cubic spline to a set of data points. It starts by defining an initial guess for the spline coefficients as zeros. The curve_fit function then fits the cubic spline, defined in the cubic_spline method, to the data points stored in $self.x$ and $self.y$. The optimized coefficients $(a\_opt, b\_opt, c\_opt, d\_opt)$ are extracted from this fitting process. The function appends these coefficients and corresponding lambda values to respective lists for later analysis, storing the list of xy coordinate tuples and frequencies for further processing.

```
        def process_spline_extraction(self):
                # Initial guess for parameters
                initial_guess = (0, 0, 0, 0)
                """Fit a Cubic spline to the given
                data points (x, y)."""

                # Perform the curve fitting
                params, covariance = curve_fit(
                    self.cubic_spline,
                    self.x, self.y, p0=initial_guess)

                # Extract the optimized coefficients
                a_opt, b_opt, c_opt, d_opt = params

                lambda_list.append(lambda_value)
                coeff_a_list.append(a)
                coeff_b_list.append(b)
                coeff_c_list.append(c)
                coeff_d_list.append(d)
                # contains list of xy tuples
                xy_list.append(xy_tuple)
                _freqs.append(frq)

            # create dataframe with the desired attributes
            _df = pd.DataFrame(list(zip(_freqs, lambda_list,
                        coeff_a_list, coeff_b_list, coeff_c_list,
                        coeff_d_list, xy_list)),
                        columns=["Frequency", "Lambda",
                        "Coeff_a", "Coeff_b", "Coeff_c",
                        "Coeff_d", coordinates"])

```

LISTING 3.8: Least Square Cubic Spline Extraction

After finding the optimized cubic spline coefficients when they are plotted as a function of lambda for a given corner frequency of 25 GHz, from figures 3.8, 3.9, 3.10, 3.11 it is clear that up until lambda value ten, the curve is relatively smooth, and then there is a considerable spike, so this method does not give us the desired solution for our problem. We will try to find the solution in the next section by adding one more polynomial.

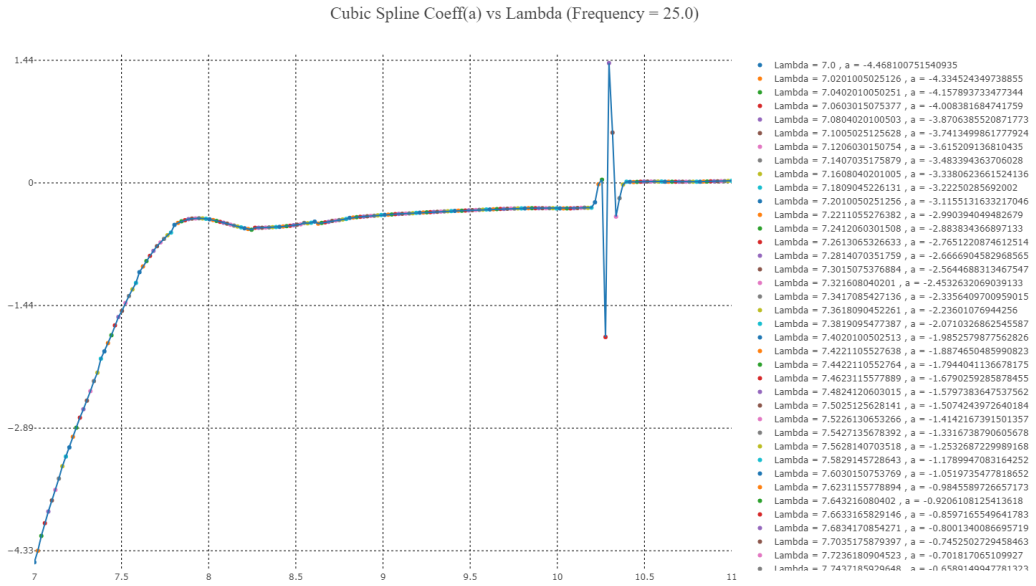


FIGURE 3.8: Cubic Spline Coefficient(a) Vs Lambda (Frequency = 25.0)

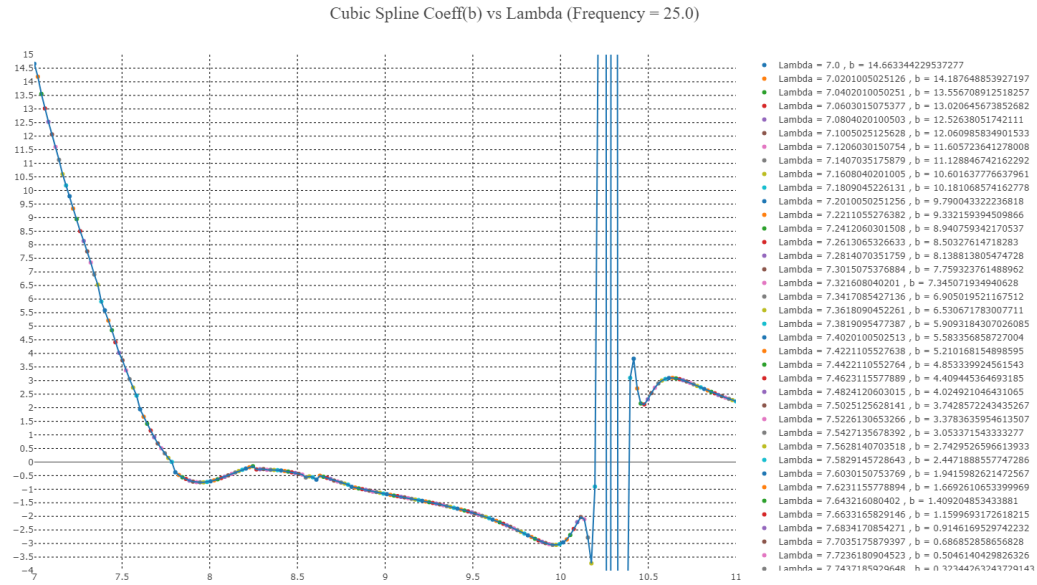


FIGURE 3.9: Cubic Spline Coefficient(b) Vs Lambda (Frequency = 25.0)

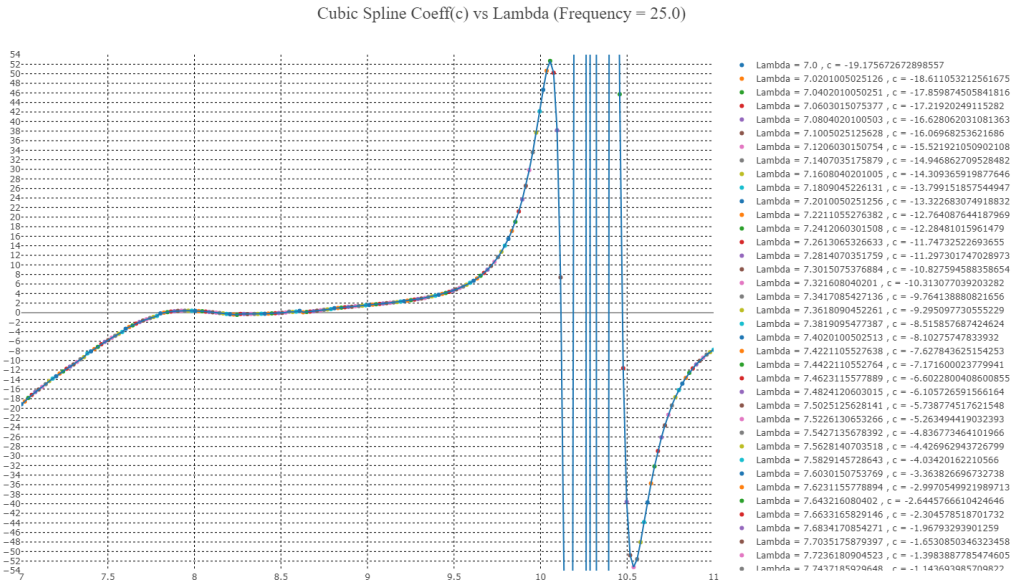


FIGURE 3.10: Cubic Spline Coefficient(c) Vs Lambda (Frequency = 25.0)

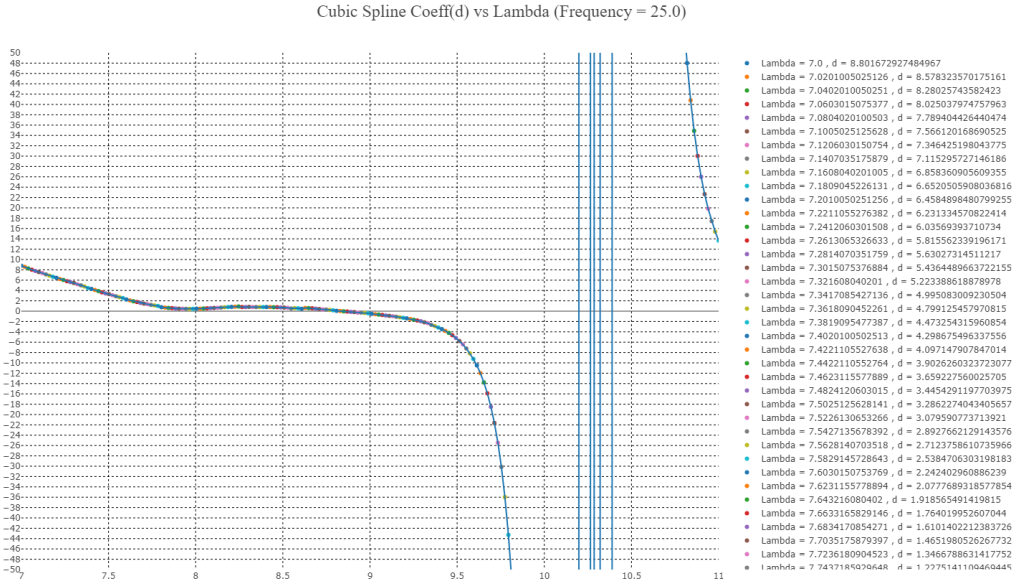


FIGURE 3.11: Cubic Spline Coefficient(d) Vs Lambda (Frequency = 25.0)

3.3.5 Quartic Spline

This thesis section delineates the methodical approach and code implementation for the Least Squares Quartic Spline Fit. It involves a sequential presentation detailing each step of the process and the associated Python code. This comprehensive approach begins with the theoretical framework of quartic spline fitting, followed by a practical demonstration through code, highlighting the application of specific functions and methods. The focus is on ensuring clarity in the step-by-step development and execution of the quartic spline fitting technique.

- **Quartic Spline:** The quartic_spline function in this code snippet 3.9 is designed to compute the value of a quartic spline at given data points. It first ensures that the input variable x is in the form of a NumPy array for efficient computation. Then, it evaluates the quartic spline equation $a + bx + cx^2 + dx^3 + ex^4$. Here, a, b, c, d, and e are coefficients that define the shape of the spline. This equation represents a fourth-degree polynomial, which is crucial for fitting a quartic spline to a set of data points in curve-fitting processes.

```
1         def quartic_spline(self, x, a, b, c, d, e):
2             x = np.array(x)
3             """ Quartic Spline Equation. """
4             return a + b * x  + c * x**2 + d * x**3 + e * x**4
5
```

LISTING 3.9: Quartic Spline Equation

- **Quartic Spline Extraction: LOC(2-13)** code segment outlines fitting a quartic spline to a set of data points. It begins with setting initial guesses for the spline's coefficients to zeros. The curve_fit function from the SciPy library is then utilized to perform the curve fitting, applying the quartic_spline function to the data points stored in $self.x$ and $self.y$. The quartic spline's optimized coefficients ($a\_opt, b\_opt, c\_opt, d\_opt, e\_opt$) are extracted from the fitting.

```
1         def process_spline_extraction(self):
2                 # Initial guess for parameters
3                 initial_guess = (0, 0, 0, 0, 0)
4                 """Fit a Quartic spline to the given
5                 data points (x, y)."""
6
7                 # Perform the curve fitting
8                 params, covariance = curve_fit(
9                     self.quartic_spline,
10                    self.x, self.y, p0=initial_guess)
11
12                # Extract the optimized coefficients
13                a_opt, b_opt, c_opt, d_opt, e_opt = params
14
15                lambda_list.append(lambda_value)
16                coeff_a_list.append(a)
```

```
                coeff_b_list.append(b)
                coeff_c_list.append(c)
                coeff_d_list.append(d)
                coeff_e_list.append(e)
                # contains list of xy tuples
                xy_list.append(xy_tuple)
                _freqs.append(frq)

            # create dataframe with the desired attributes
            _df = pd.DataFrame(list(zip(_freqs, lambda_list,
                        coeff_a_list, coeff_b_list, coeff_c_list,
                        coeff_d_list, coeff_d_list, xy_list)),
                        columns=["Frequency", "Lambda",
                        "Coeff_a", "Coeff_b", "Coeff_c",
                        "Coeff_d",v"Coeff_e", coordinates"])

```

LISTING 3.10: Least Square Quartic Spline Extraction

These captured coefficients are then visualized to check for the smoothness of the curve. Fig. 3.12, 3.13, 3.14, 3.15, and 3.16 illustrate the plots for coefficients as a function of lambda. From the figures, it is clear that increasing the polynomial degree does not help with smoothen curves. So, in the upcoming section, we will use a different approach to finding the circle using cubic spline because, so far, only cubic spline somewhat gives smooth behavior.

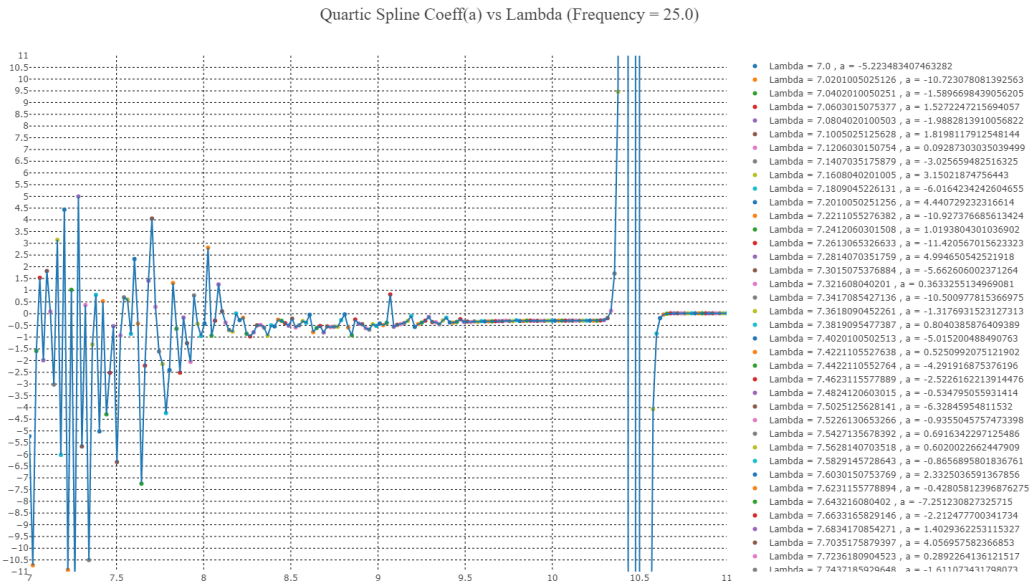


FIGURE 3.12: Quartic Spline Coefficient(a) Vs Lambda (Frequency = 25.0)

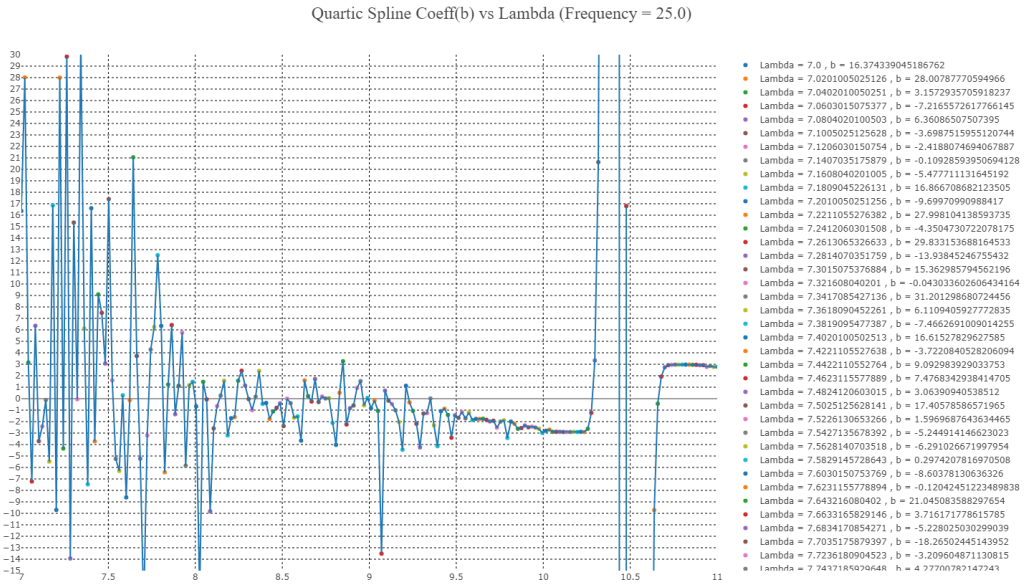


FIGURE 3.13: Quartic Spline Coefficient(b) Vs Lambda (Frequency = 25.0)

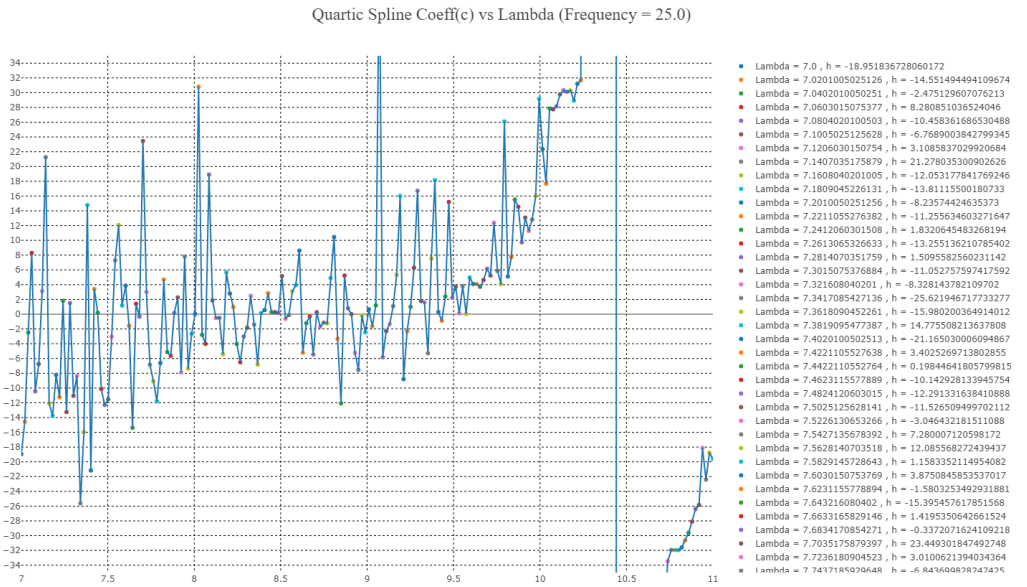


FIGURE 3.14: Quartic Spline Coefficient(c) Vs Lambda (Frequency = 25.0)

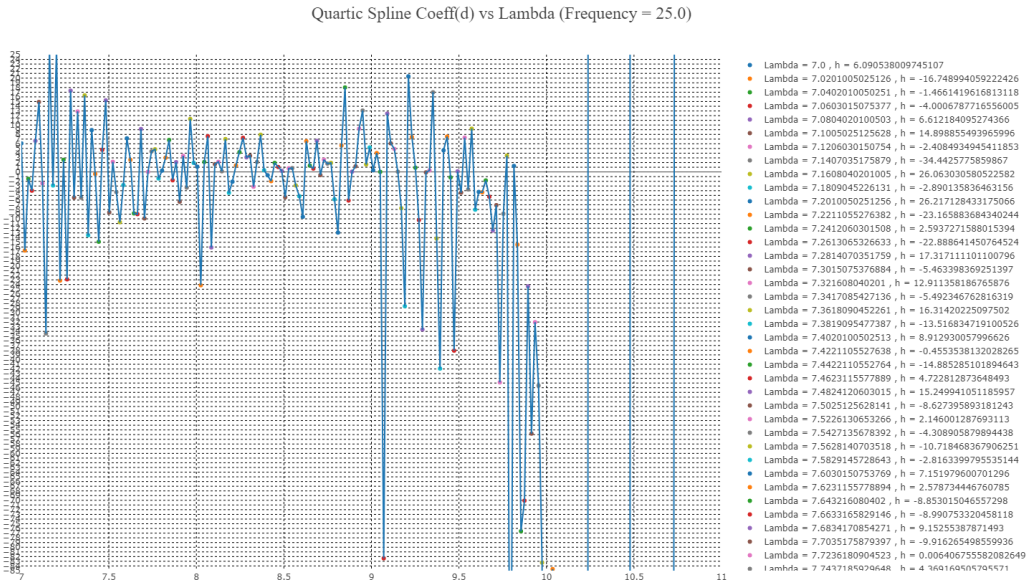


FIGURE 3.15: Quartic Spline Coefficient(d) Vs Lambda (Frequency = 25.0)

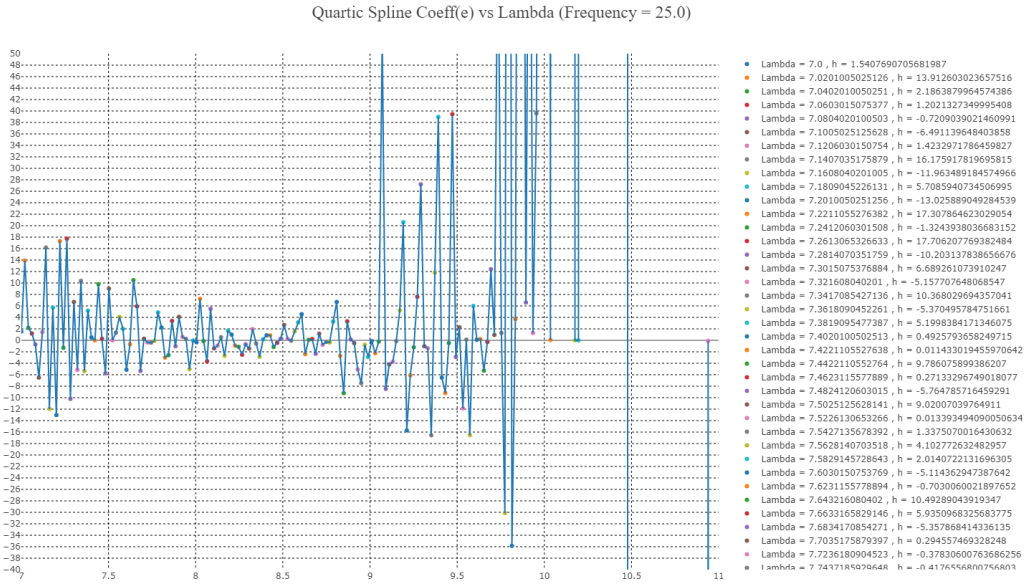


FIGURE 3.16: Quartic Spline Coefficient(e) Vs Lambda (Frequency = 25.0)

All the approaches that were discussed above are based on the function $y = f(x)$, i.e., we tried to find the best-fitting curve or shape as a function of x, which did not yield the desired results. From Figures 3.12, 3.13, 3.14, 3.15, and 3.16, we can see that higher-order polynomials can capture the complex patterns in the data, resulting in over-fitting, which might be the cause of the waves in the plots. Another cause

might be an outlier in the range of 10-10.5 lambda values because fitting a spline at that point has a sudden curve, causing huge spikes in the graphs. In the next section, we will implement a completely different approach for the circle fitting.

3.4 Circle Parameter Extraction

In our case, circle extraction from $y = f(x)$ is possible. In this case, we have two branches (upper and lower halves of the circle), and finding tangent and curvature at a point where the upper and lower halves of the circle meet becomes more complicated. So, using vector calculus and expressing the function $y = f(x)$ as a vector function, as mentioned in equation 2.4, makes it easier to express the tangent and curvature vectors and derive their formulas at any point on the circle. In order to express the function $y = f(x)$ as a vector function, we will derive two separate equations for x and y coordinates.

This section on circle extraction using cubic splines explores the idea of computing separate cubic splines for x and y coordinates to find tangents in the Cartesian plane accurately. This approach enhances the precision in determining the tangent vector at each point on the curve. The step-by-step process, supported by code implementation, not only demonstrates the extraction of circle parameters but also delves into the computation of tangents and curvature, showcasing the effectiveness of handling x and y coordinates through individual cubic splines.
The code snippet 3.11 demonstrates the process of extracting circle parameters. The core part can be explained as follows:

- **LOC(5-8):** First, it identifies the index of the selected frequency and determines the range for cubic spline fitting. The range will always be line [-i, i], where i is an Integer. It calculates the t values in such a way that at a given frequency, the value of t will always be 0. For example, let's say we have a frequency of 25, and the associated value of the x-coordinate is 2.566778; then the function value will become 2.566778 at $t = 0$, i.e., $x(t = 0) = 2.566778$
- **LOC(14-26):** The code generates a range for cubic spline interpolation and computes cubic splines for x and y coordinates using this range. It then calculates the first and second derivatives of these splines. The first derivatives (dx_dt, dy_dt) are essential for finding the tangent vector at a given point, while the second derivatives (ddx_dt, ddy_dt) are used to determine the curvature at the same point. This code segment is crucial for analyzing the curve's properties at a specific frequency.
- **LOC(29-38):** Calculates the tangent vector and curvature at a specific frequency point using derivatives of cubic splines. It computes the tangent using the first derivatives of x and y splines evaluated at a given frequency point. Curvature is then derived from these derivatives, providing a measure of how sharply the

curve bends. Additionally, the radius of curvature is calculated, which gives the radius of the approximated circle at that point.

- **LOC(40-50):** Computes the curve's unit tangent and normal vectors at a specific frequency point. It then uses these vectors, along with the calculated curvature and radius, to determine the center of the corresponding circle. The unit tangent vector is normalized to ensure it has a length of one, and the normal unit vector is obtained by rotating the tangent vector. Finally, the circle's center is calculated, incorporating the radius and the direction of the normal unit vector, providing essential parameters for the circle associated with the spline at the given frequency.

```
1       def create_tangent_for_freq(self, neigh_freq, freq, x_coord,
2               y_coord, y_spline, lambda_val):
3
4           # get the index of the selected frequency
5           freq_idx = neigh_freq.index(freq)
6           total_neigh = len(neigh_freq)
7           start = len(neigh_freq[:freq_idx])
8           stop = len(neigh_freq[freq_idx+1:])
9
10          # generate range as a function parameter for
11          #    x and y data points
12          # for a given "freq" value the t value will be 0
13          # if total_neigh = 5 than t range = [-2, 2]
14          t = np.linspace(-start, stop, total_neigh)
15
16          # find cubic splines
17          x_t = cs(t, x_coord)
18          y_t = cs(t, y_coord)
19
20          # get derivative
21          dx_dt = x_t.derivative()
22          dy_dt = y_t.derivative()
23
24          # get second order derivative
25          ddx_dt = dx_dt.derivative()
26          ddy_dt = dy_dt.derivative()
27
28          # tangent of at a given "freq" point
29          tangent = np.array([dx_dt(0), dy_dt(0)])
30
31          # curvature at the tangent point
32          numerator = dx_dt(0) * ddy_dt(0) - dy_dt(0) * ddx_dt(0)
33          denominator = (dx_dt(0)**2 + dy_dt(0)**2)**1.5
34          curvature = numerator / denominator
35
36          # radius of the curve (circle)
37          radius = np.abs(((dx_dt(0)**2 + dy_dt(0)**2)**1.5)/
38                  (dx_dt(0)*ddy_dt(0) - dy_dt(0)*ddx_dt(0)))
```

```
39
40              # unit tangent
41              T_unit = tangent / np.linalg.norm(tangent)
42
43              # normal unit tangent
44              N = np.array([-T_unit[1], T_unit[0]])
45
46              # center of the circle
47              C = np.array([x_t(0), y_t(0)]) +
48                  np.sign(curvature) * radius * N
49
50              return C[0], C[1], radius # circle parameters
```

LISTING 3.11: Circle Parameter Extraction

Fig 3.17, 3.18, 3.19, and 3.20 depict plots of Phase, Radius, X-Center, and Y-Center vs Lambda, respectively. From the figures, it is evident that only the circle phase has smooth behavior. Still, other curves are not smooth, and extracting the circle parameters using a spline approach does not help us achieve the desired results. The concrete reason for this problem is discussed in the results section of this thesis.

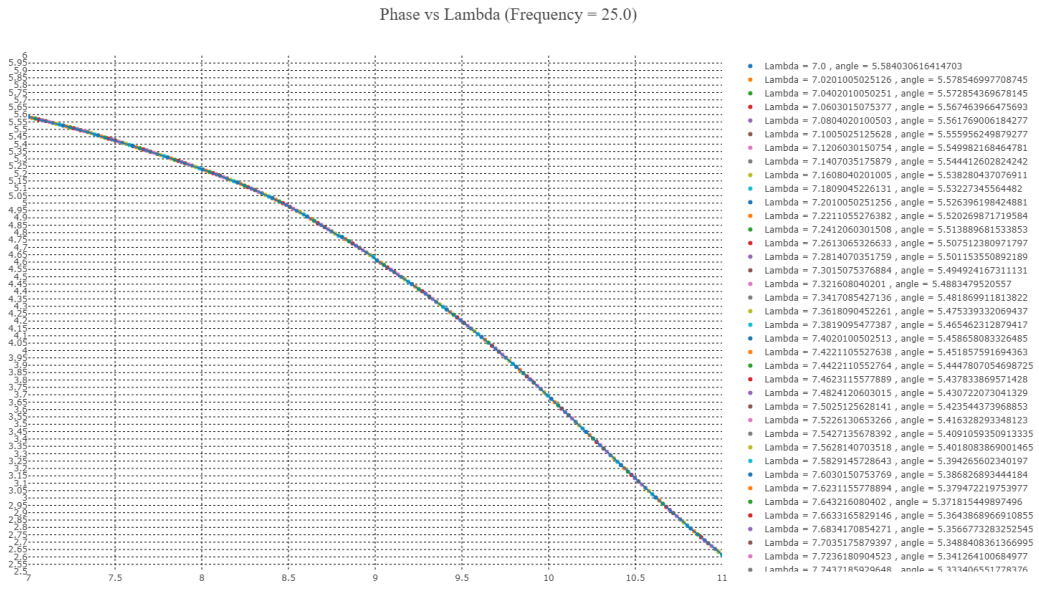


FIGURE 3.17: Phase Vs Lambda (Frequency = 25.0)

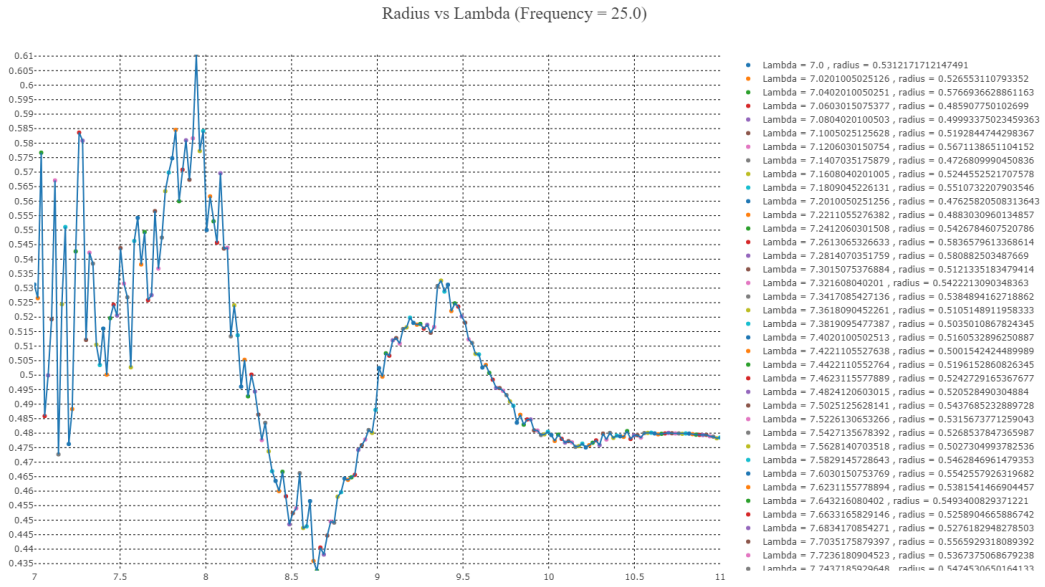


FIGURE 3.18: Radius Vs Lambda (Frequency = 25.0)

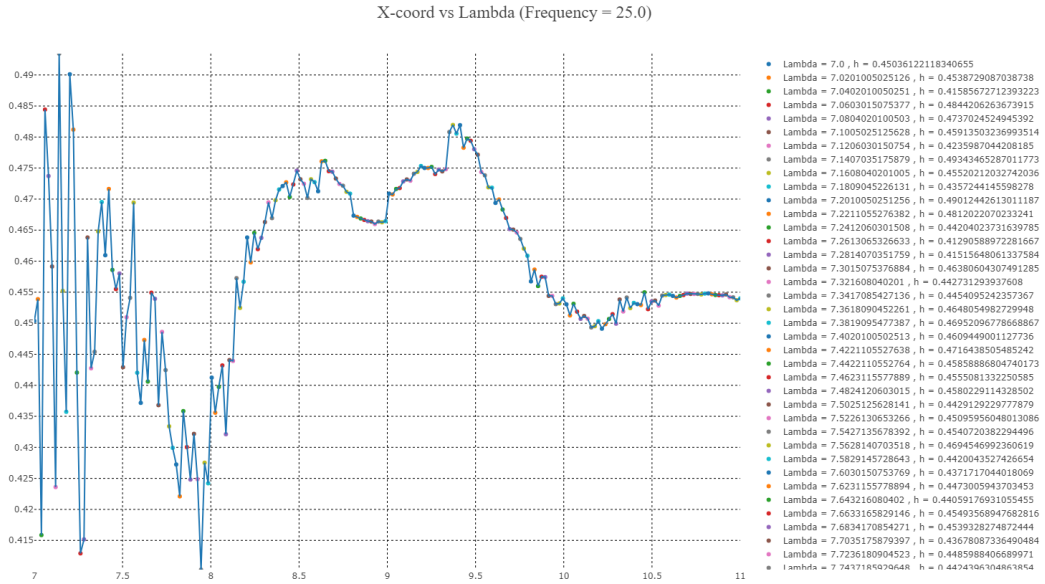


FIGURE 3.19: X-Center Vs Lambda (Frequency = 25.0)

In the initial part of the circle parameters vs. Lambda plots (see Fig., 3.18, 3.19, and 3.20) we can see that the behavior is chaotic, which is due to the selection of neighbors in the starting frequency range. In the starting frequency range, the neighbor selection is one-directional, so tangent and curvature at that point significantly affect the circle radius.

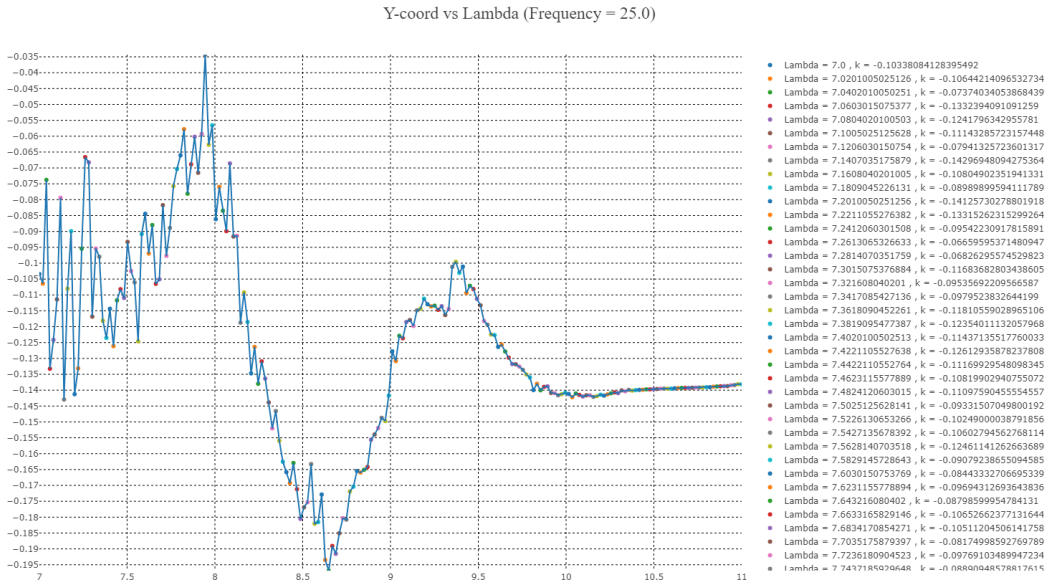


FIGURE 3.20: Y-Center Vs Lambda (Frequency = 25.0)

3.5 Code Deployment

Launched in 2008, GitHub has become a renowned desktop and web-based hosting platform, pivotal for Continuous Integration (CI) and Continuous Deployment (CD) in software development (github, 2008). The code in this thesis was crafted using Visual Studio Code IDE. This entire code base and the thesis results have been uploaded to GitHub. For detailed viewing and exploration, the repository is accessible at the following URL: `https://github.com/M-Dabhi/surrogate_modelling_v2`

3.6 Summary

The chapter 3 on methodology in the thesis delves into advanced geometric shape-fitting techniques, building upon Dasi, 2023's foundational work. It focuses on enhancing curve approximation accuracy using least squares fitting methods, comparing quadratic, cubic, and quartic splines. The chapter highlights algorithmic strategies and the use of SciPy for optimization, with a detailed discussion of the mathematical and computational aspects of spline fitting. It also covers the implementation of various algorithms for ellipse and spline fitting, emphasizing the precision in data representation. The methodologies for neighbor frequency extraction and the use of Python, Matplotlib, Plotly, and SciPy are discussed, illustrating their roles in data analysis and visualization. The chapter concludes with an exploration of circle parameter extraction using cubic splines. The challenges encountered in achieving the desired smoothness in curve fitting will be discussed in the Results and Discussion chapter 4.

Chapter 4

Results and Discussion

In this chapter, the focus is on the critical analysis and interpretation of the outcomes derived from the application of geometric fitting techniques to the simulation data. This chapter synthesizes the findings from implementing various spline fitting methods, including quadratic, cubic, and quartic splines, and the ellipse fitting process. The effectiveness of these methods in capturing the inherent patterns of the data is rigorously examined, along with any challenges and discrepancies encountered. The chapter also delves into the nuances of extracting circle parameters using cubic splines, assessing the precision and effectiveness of this technique. The insights gained from this analysis are contextualized within the broader scope of the research objectives.

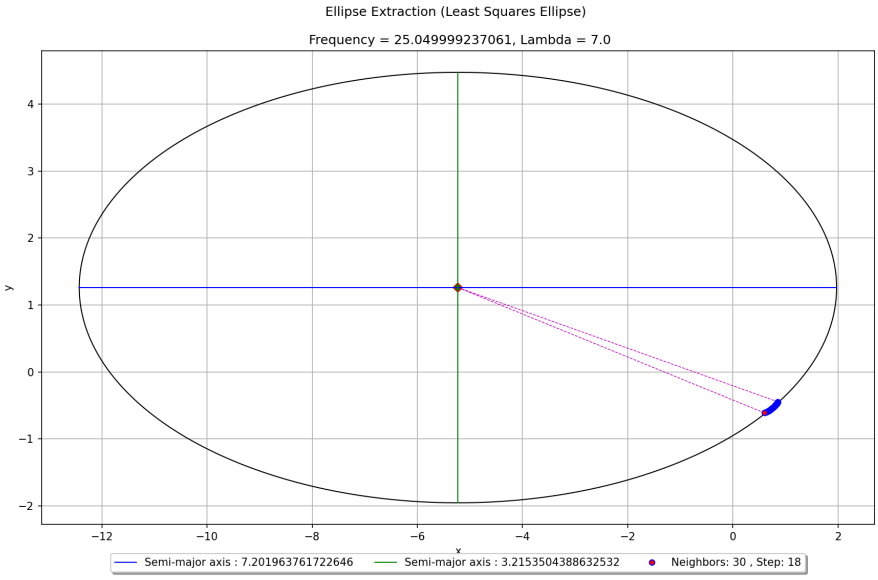
4.1 Results

4.1.1 Evaluation of Ellipse Extraction:

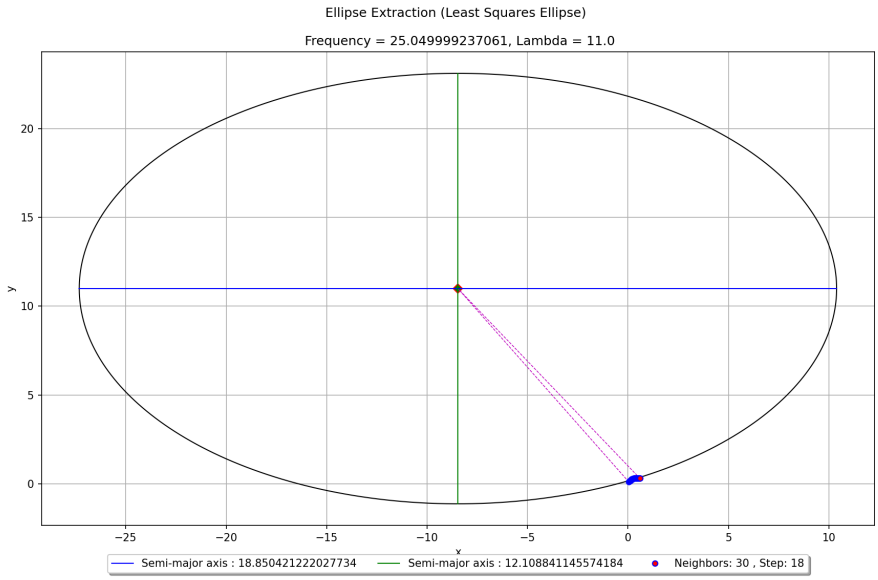
Figure 4.1 illustrates the ellipse extraction process for frequencies around 25 GHz, focusing on 30 neighboring frequencies at lambda values 7 and 11. This comparison reveals distinct patterns in how the data points are distributed around the ellipse for different lambda values. For lambda value 11, the points appear closely clustered, forming a compact curve, as shown in Fig.4.1b. In contrast, the ellipse corresponding to lambda value 7, as depicted in Fig.4.1a, demonstrates a broader spread of data points, resulting in a more expansive curve. This variation in point distribution for the same number of neighboring frequencies highlights the sensitivity of the ellipse shape to the chosen lambda value, reflecting the intricate relationship between frequency, lambda, and the resulting geometric representation.

From Figure 4.1, we have observed that the distribution of the points is not uniform; henceforth, it will impact the ellipse radius and center. At specific ranges of lambda for certain frequencies, the positioning of these points is not good enough to form an arc. It is tricky to fit an ellipse with such close points with the least square fit methods, which usually perform the data fitting of larger arcs or the points that

are spread across the circumference of the ellipse. The main purpose behind the discussion here is to describe the least square ellipse fit and its impact on representing the simulation data.



(A) Ellipse Extraction for Lambda 7.0



(B) Ellipse Extraction for Lambda 11.0

FIGURE 4.1: Ellipse Extraction with 30 Frequencies

Ellipse Parameter Plots as a function of Lambda:

The preceding analysis focused on extracting ellipse patterns and assessing the data points for a set of 30 frequency neighbors, particularly in relation to specific lambda values, with a special emphasis on the frequency 25.049999237061. This exploration

was instrumental in modeling the ellipse parameters extracted across the frequency spectrum. Visual representations were plotted as functions of lambda at specific frequencies to gauge the continuity and smoothness in the variation of these parameters. A deeper dive into the behavior of these parameters at crucial corner frequencies of the spectrum was initiated. Specifically, for the frequency 25.049999237061, the analysis delved into the ellipse extractions at lambda values 7 and 11, representing a small subset of the 200 lambda values in the simulation data. This comprehensive approach enabled a meticulous extraction and analysis of ellipse parameters across these extensive lambda values, providing a thorough insight into the behavior and characteristics of these parameters at the selected frequency.

The analysis of the results from the plots for the frequency 25.049999237061, particularly in relation to the lambda parameter, is presented in Figures 4.2, 4.3, 4.4, and 4.5. These figures illustrate the variation of each ellipse parameter – including the semi-major axis, semi-minor axis, and center coordinates – as influenced by lambda values within the 7-11 range. It is observed that these curves exhibit unexpected noise and abrupt changes. The speculation at this stage is that minor variations in data might be causing significant alterations in the ellipse parameters.

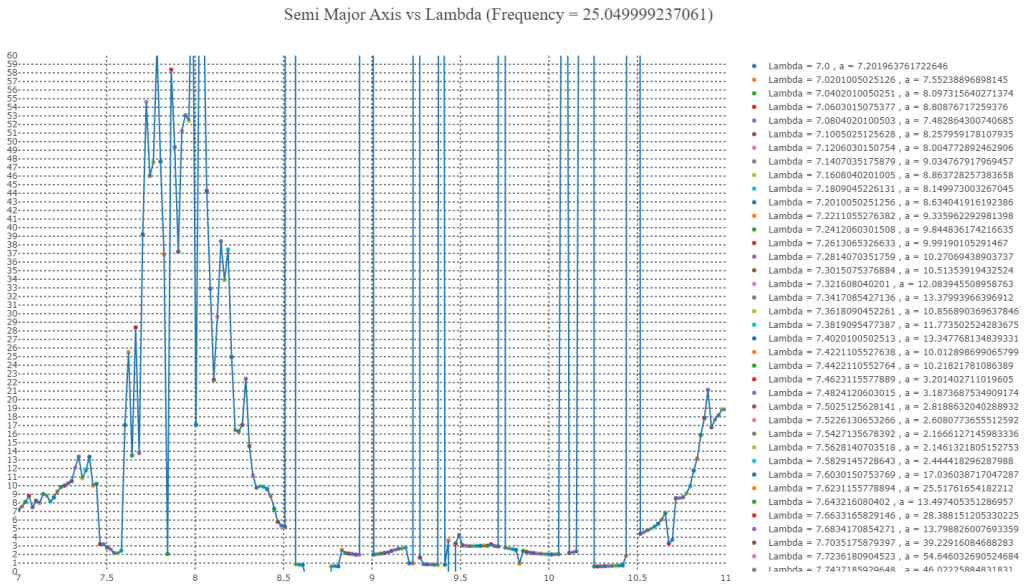


FIGURE 4.2: Semi-Major Axis Vs Lambda (Frequency = 25.049999237061)

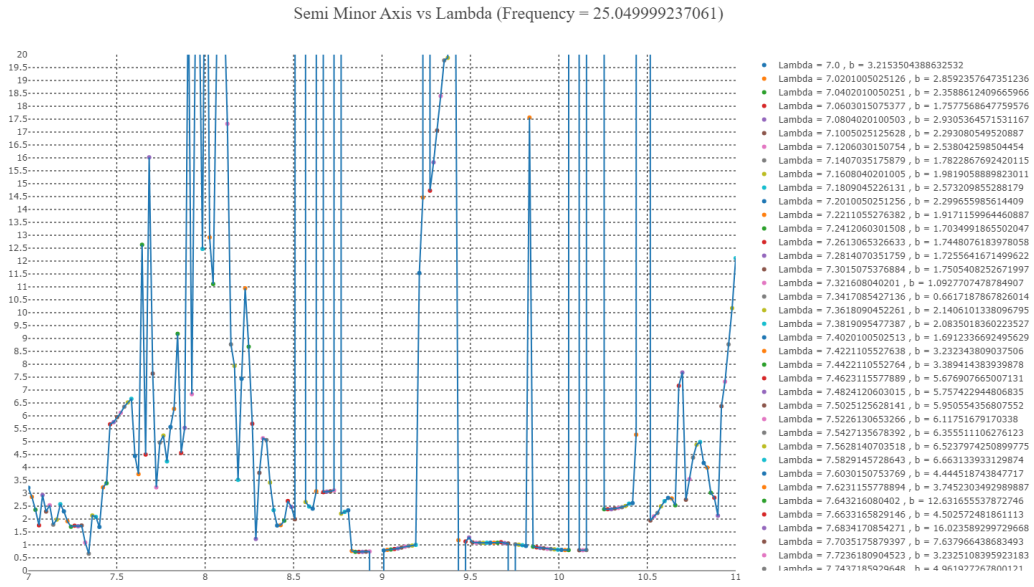


FIGURE 4.3: Semi-Minor Axis Vs Lambda (Frequency = 25.049999237061)

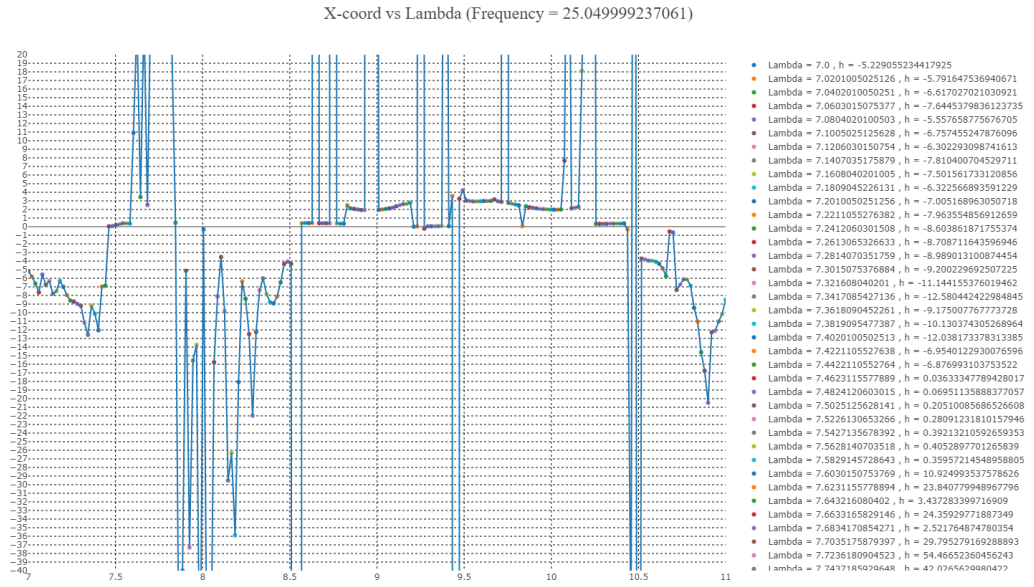


FIGURE 4.4: X-Center Vs Lambda (Frequency = 25.049999237061)

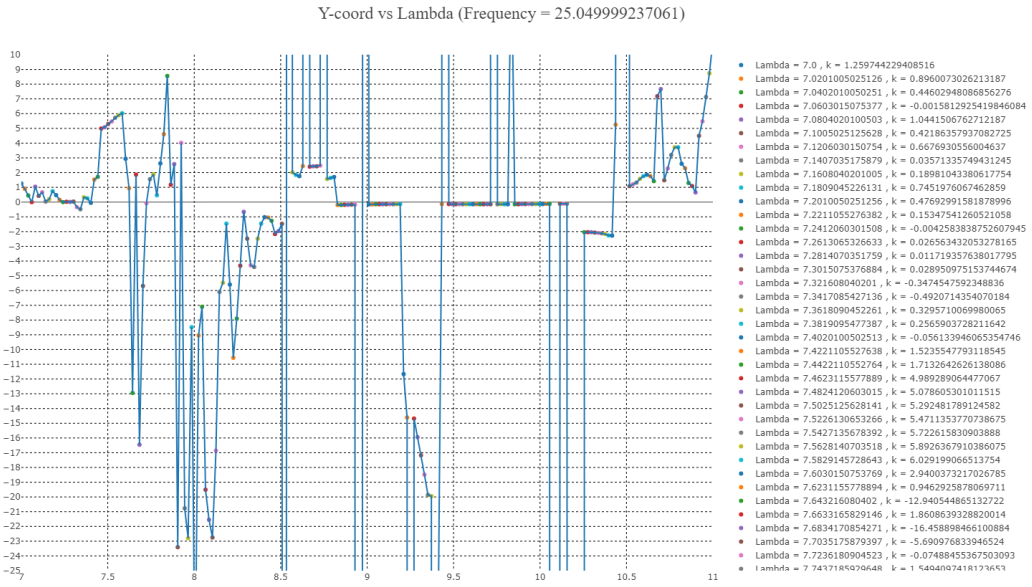


FIGURE 4.5: Y-Center Vs Lambda (Frequency = 25.049999237061)

Let us now examine the results of the ellipse parameter for the frequency 30.0. From the figures in 4.6, 4.7, 4.8, 4.9, the resulting curves are as the previous frequency results (see: Figures 4.2, 4.3, 4.4, 4.5), which belongs to the corner of frequency spectrum). The results should be somewhat similar to those from the previous study of LSCF, but they are a lot worse. As discussed in the previous section, this behavior is due to a huge change in ellipse parameters because ellipse fit is more sensitive to changes in data compared to circles. So, the overall behavior of the least square ellipse yields unexpected changes in the curve. This challenge underscores the need to explore alternative least square fitting methods to address these issues and potentially achieve more stable results.

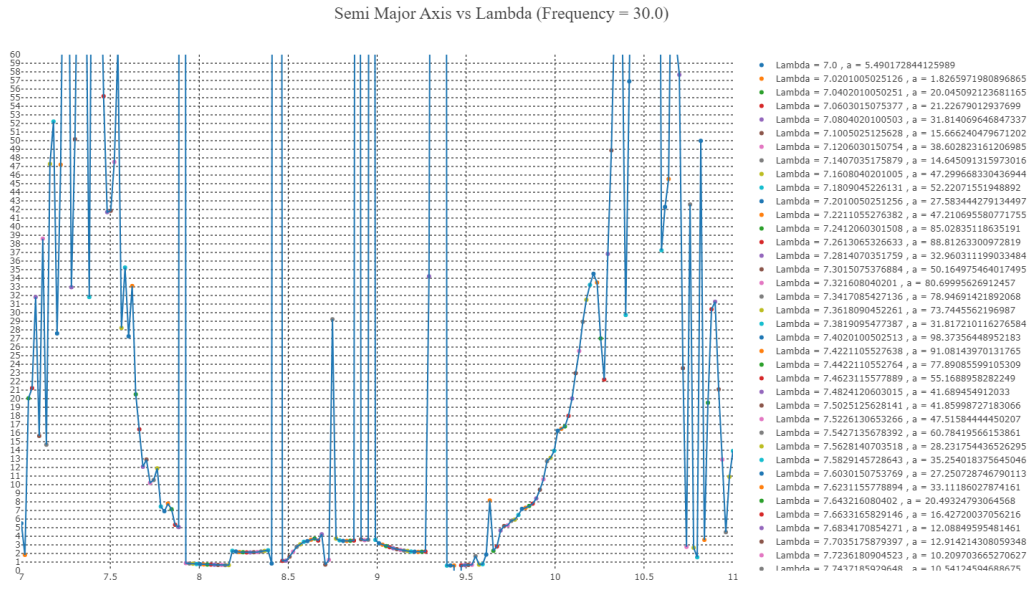


FIGURE 4.6: Semi-Major Axis Vs Lambda (Frequency = 30.0)

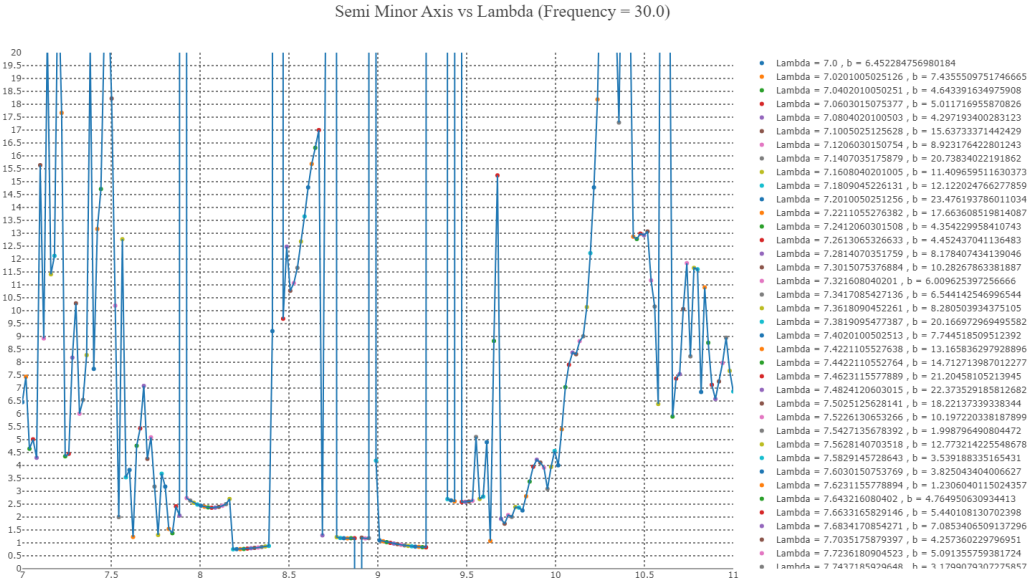


FIGURE 4.7: Semi-Minor Axis Vs Lambda (Frequency = 30.0)

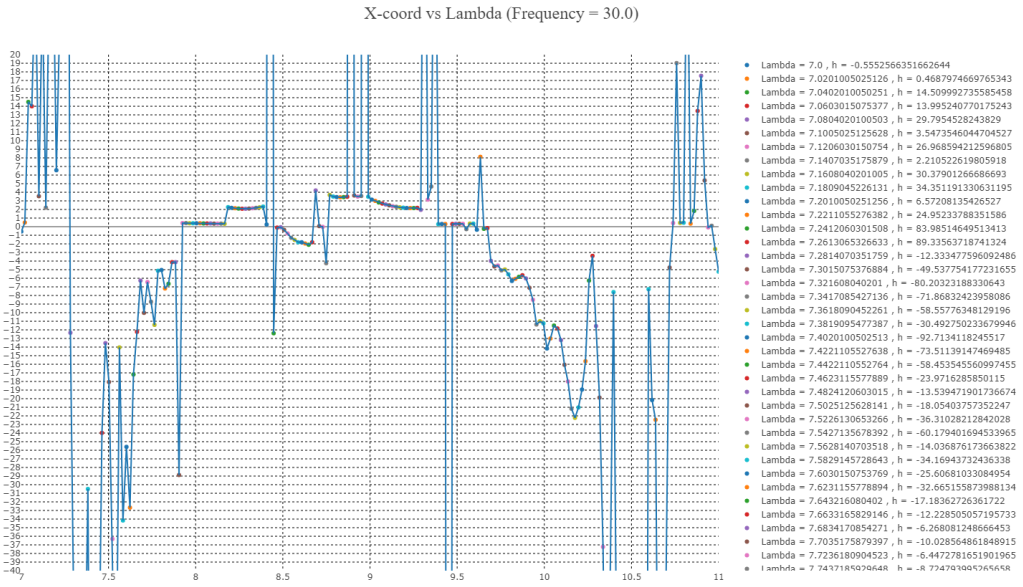


FIGURE 4.8: X-Center Vs Lambda (Frequency = 30.0)

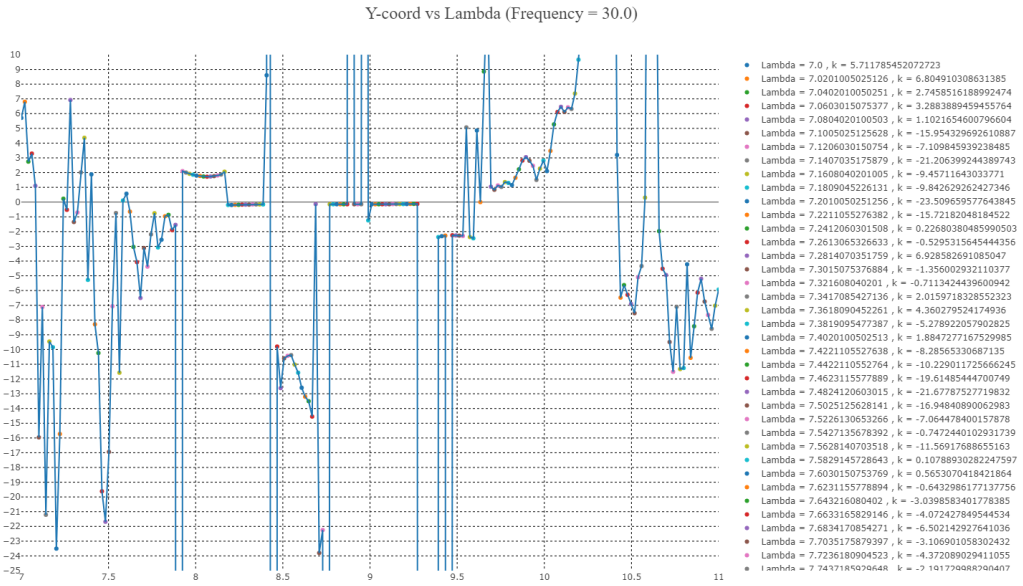


FIGURE 4.9: Y-Center Vs Lambda (Frequency = 30.0)

4.1.2 Evaluation of Quadratic Spline

Spline Coefficient as a function of Lambda:

The figures 4.10, 4.11, and 4.12 represent the behavior of quadratic spline coefficients a, b, and c as a function of lambda for a fixed corner frequency of 25.049999237061 GHz. They plot the variability of each coefficient across a range of lambda values. Such plots are useful to assess the smoothness and stability of the spline fitting across

the dataset. Curves are smooth till the lambda value of 10, then there are sudden Large spikes or irregularities in the plots.

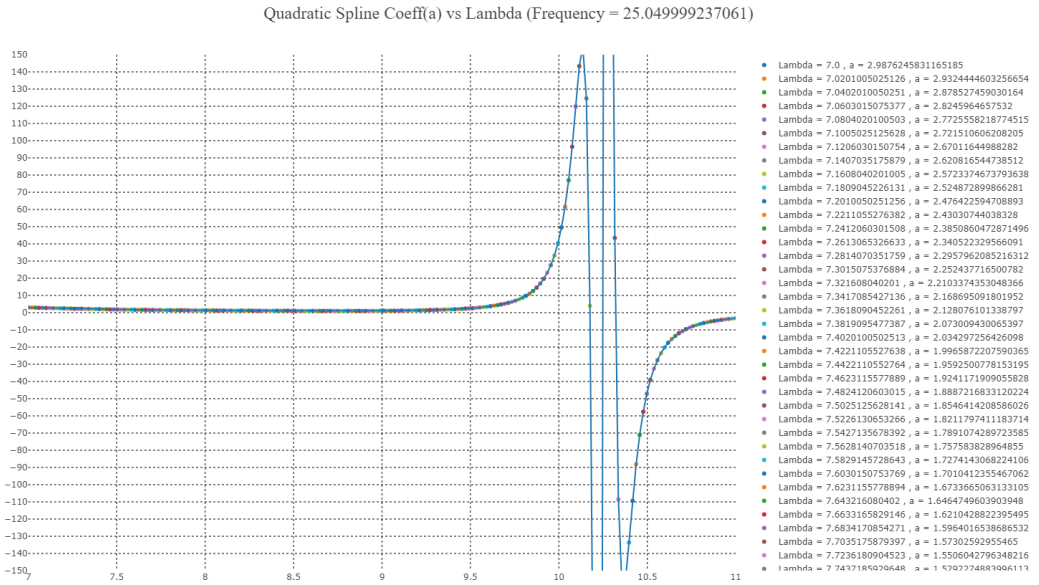


FIGURE 4.10: Quadratic Spline Coefficient(a) Vs Lambda (Frequency = 25.049999237061)

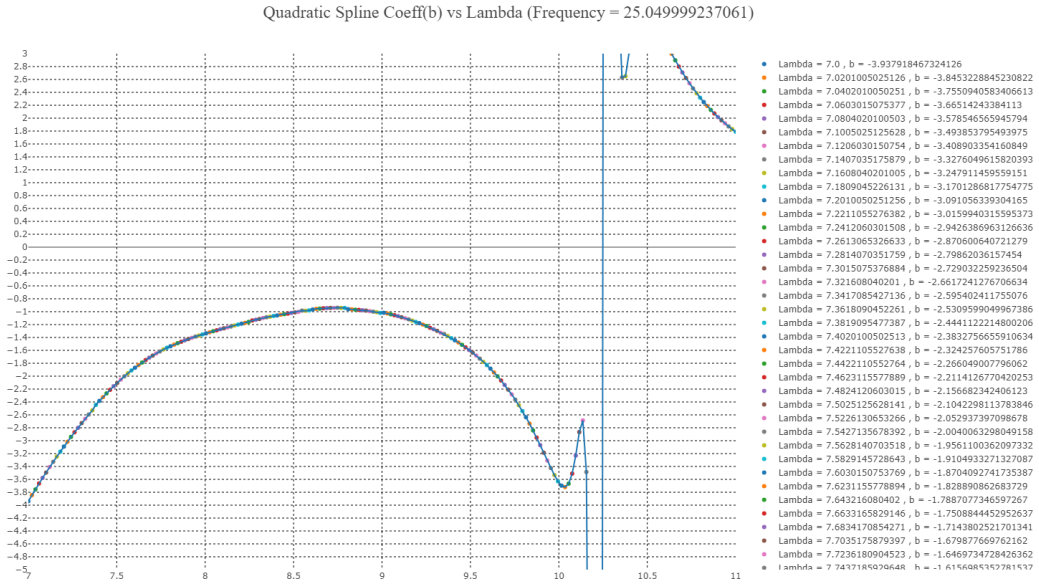


FIGURE 4.11: Quadratic Spline Coefficient(b) Vs Lambda (Frequency = 25.049999237061)

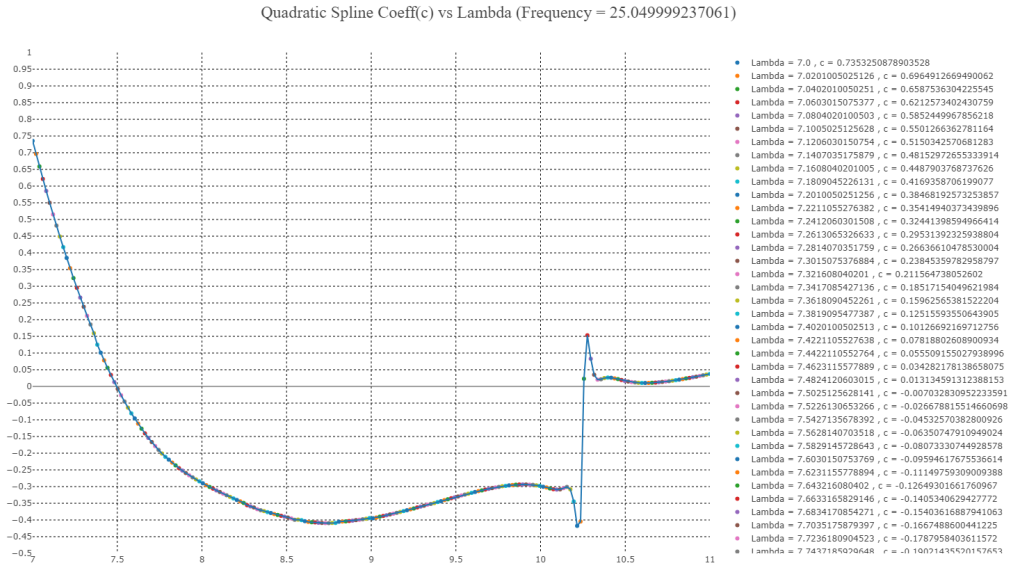


FIGURE 4.12: Quadratic Spline Coefficient(c) Vs Lambda (Frequency = 25.049999237061)

Now, let us look at the plots of the quadratic spline coefficients for 30 GHz frequency. The resulting curves from the figures 4.13, 4.14, and 4.15 are slightly different compared to corner frequency results (see Figures 4.10, 4.11, 4.12). Here, the spikes are in the range of 8.5-9 lambda values, and we can see this abnormal behavior in polar representation 4.16. These spikes in the region are due to the problem modality of $y = f(x)$ because, except for certain regions, all the curves look ideally smooth. This indicates that alternative fitting methods are necessary to achieve more accurate results.

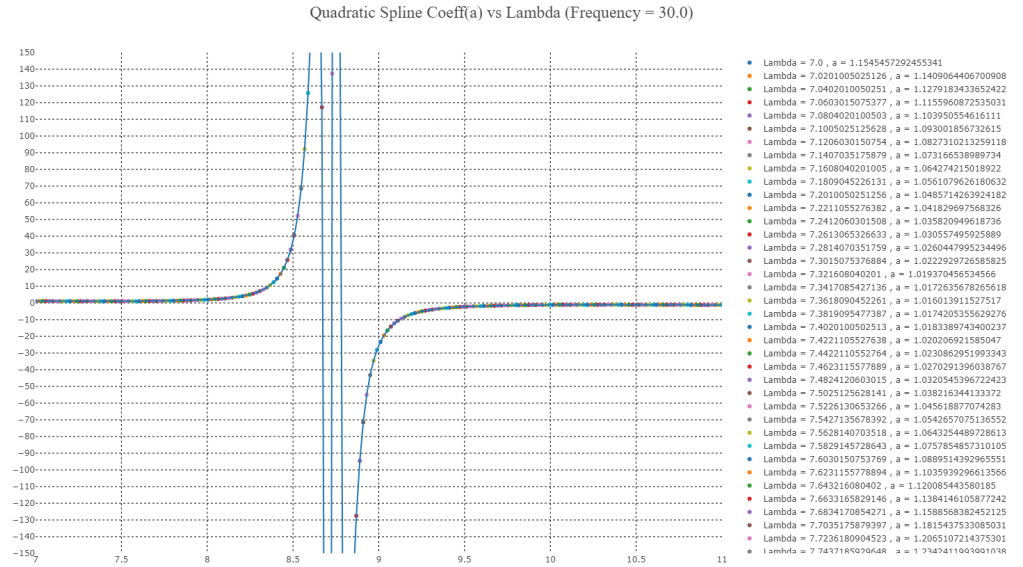


FIGURE 4.13: Quadratic Spline Coefficient(a) Vs Lambda (Frequency = 30.0)

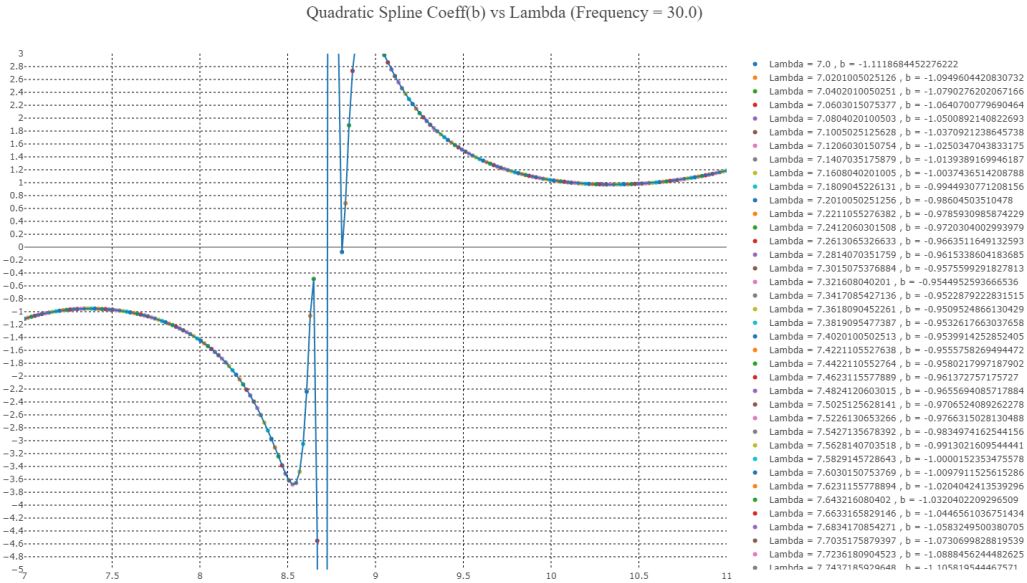


FIGURE 4.14: Quadratic Spline Coefficient(b) Vs Lambda (Frequency = 30.0)

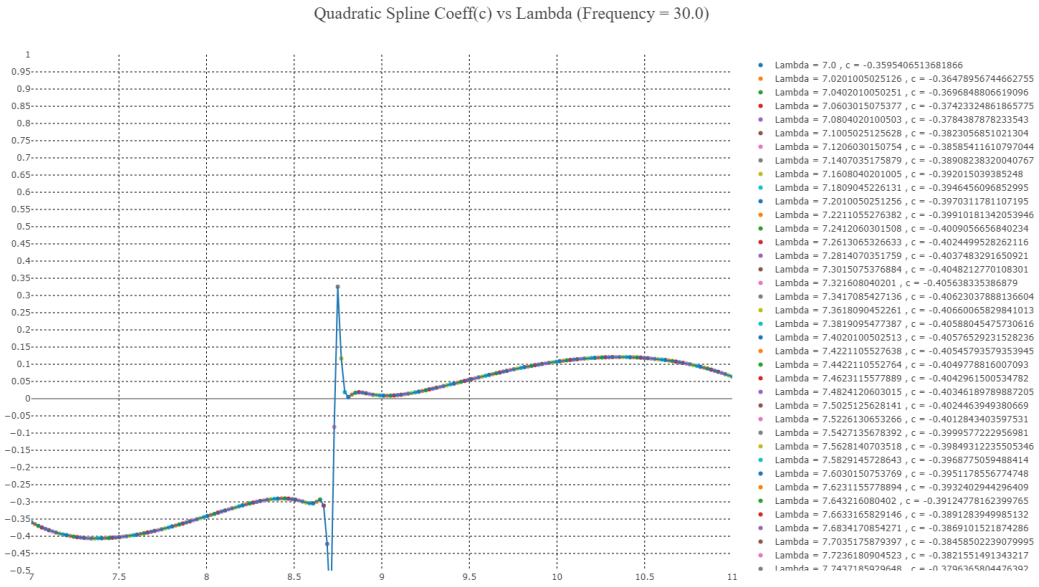


FIGURE 4.15: Quadratic Spline Coefficient(c) Vs Lambda (Frequency = 30.0)

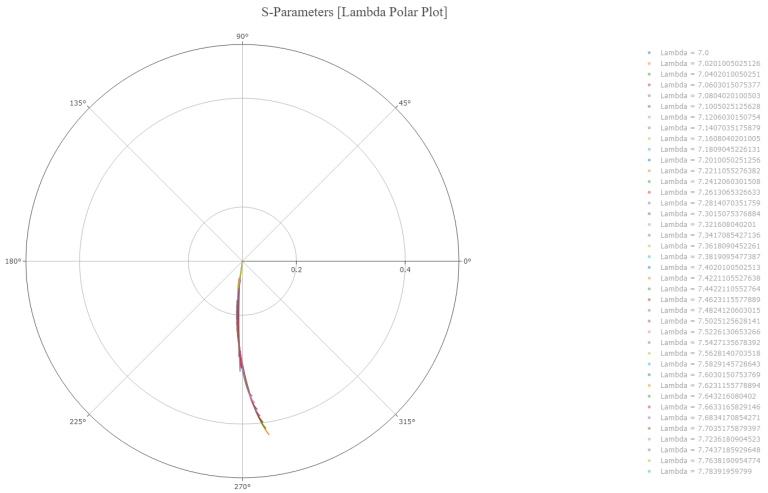


FIGURE 4.16: Polar Plot (Frequency = 30.0)

4.1.3 Evaluation of Cubic Spline:

Spline Coefficient as a function of Lambda:

The transition from quadratic to cubic spline fitting was a strategic move to address the limitations encountered with the quadratic model, particularly for the frequency of 5.049999237061 GHz. The cubic spline fitting involves an additional coefficient, 'd', contributing to the model's flexibility. Similar challenges were observed despite this advancement, as illustrated in the provided figures 4.17, 4.18, 4.19, and 4.20. Particularly in the lambda range of 10-10.5, the coefficients b, c, and d demonstrated substantial variability, diverging noticeably from adjacent values.

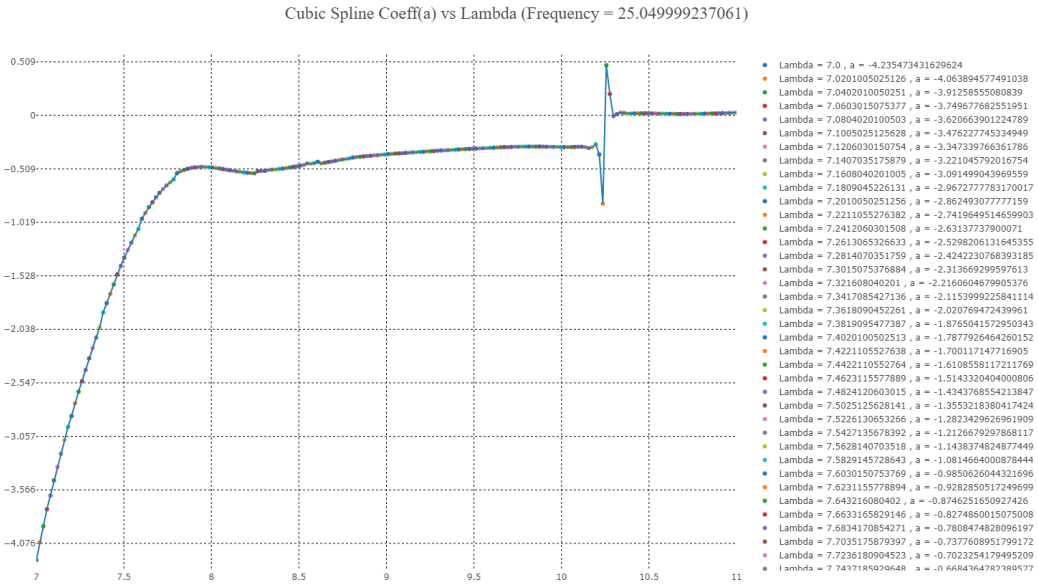


FIGURE 4.17: Cubic Spline Coefficient(a) Vs Lambda (Frequency = 25.049999237061)

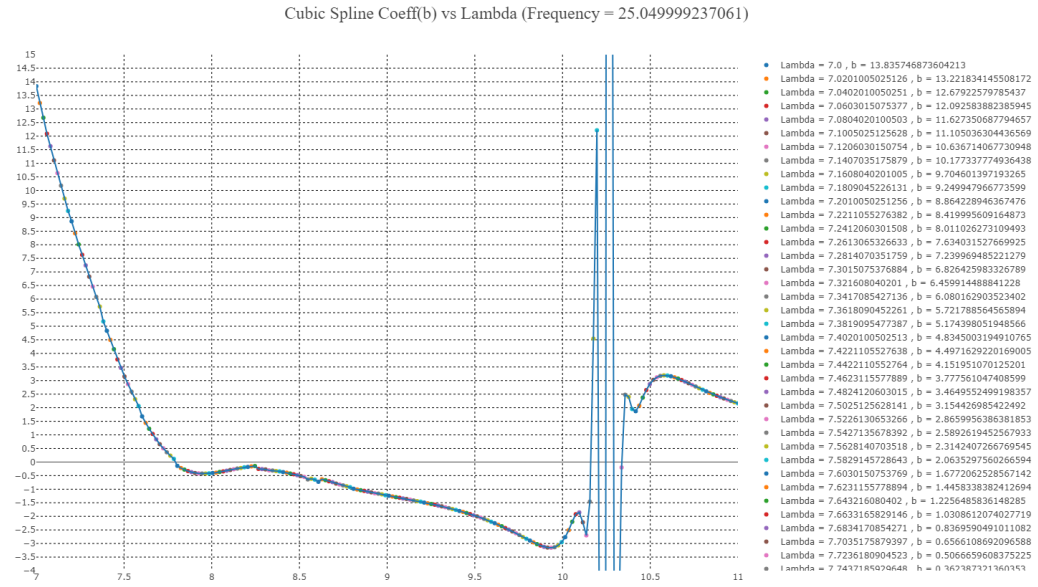


FIGURE 4.18: Cubic Spline Coefficient(b) Vs Lambda (Frequency = 25.049999237061)

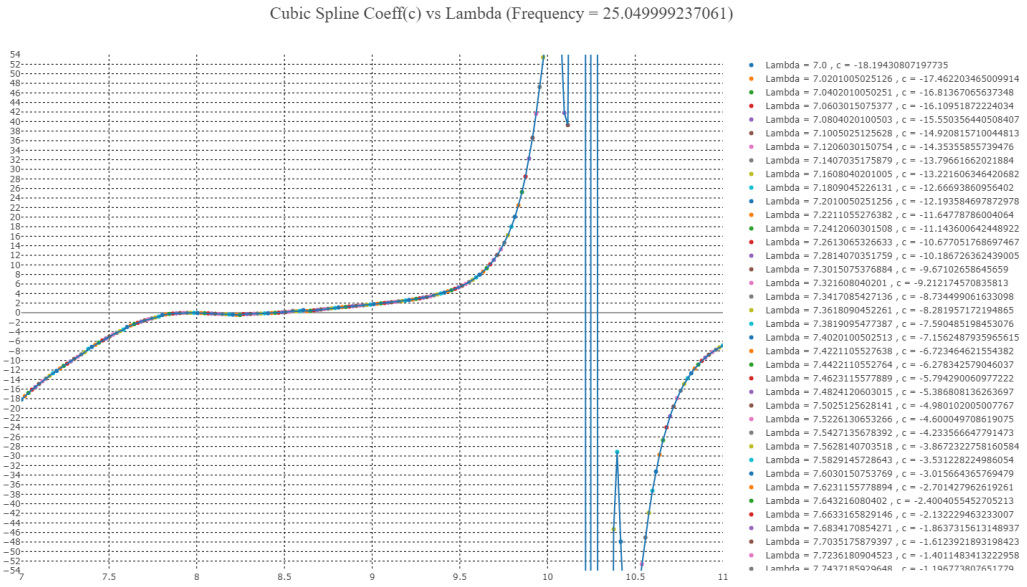


FIGURE 4.19: Cubic Spline Coefficient(c) Vs Lambda (Frequency = 25.049999237061)

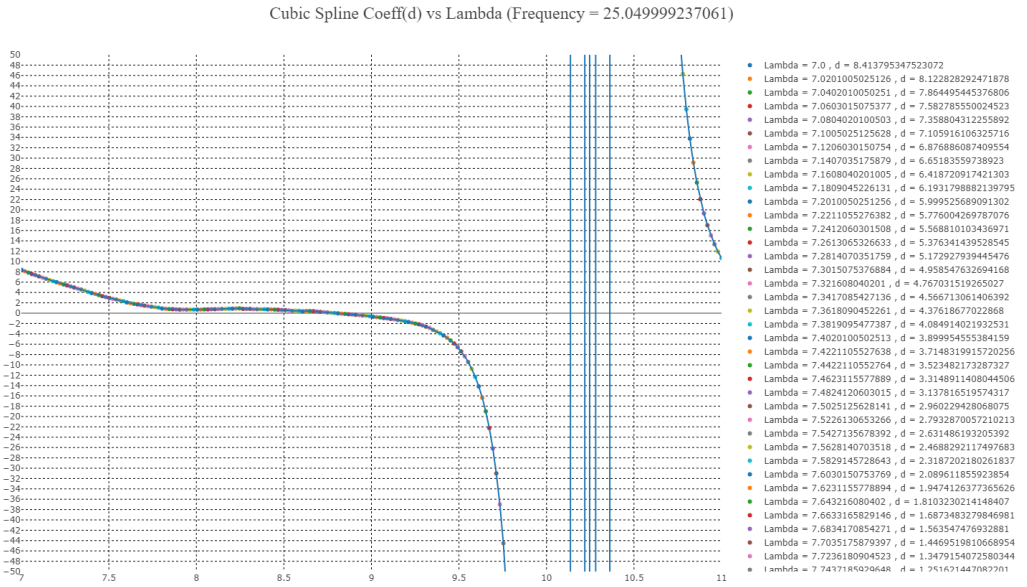


FIGURE 4.20: Cubic Spline Coefficient(d) Vs Lambda (Frequency = 25.049999237061)

To further examine this approach, another frequency, 30.0, which is the middle frequency, is used, and plots of cubic spline coefficients are depicted in the figures 4.21, 4.22, 4.23, and 4.24. The results are similar to the corner frequency results of the cubic spline and from the quadratic spline (see Figures 4.13, 4.14, 4.15) for the same frequency of 30 MHz. The range is also similar to the results of quadratic spline, which

is 8.5-9 lambda values, and here, as well, we represent the curve as $y = f(x)$, which might be the cause for the spike. For this reason, we will add one more polynomial to the spline fitting and investigate the results in the next section.

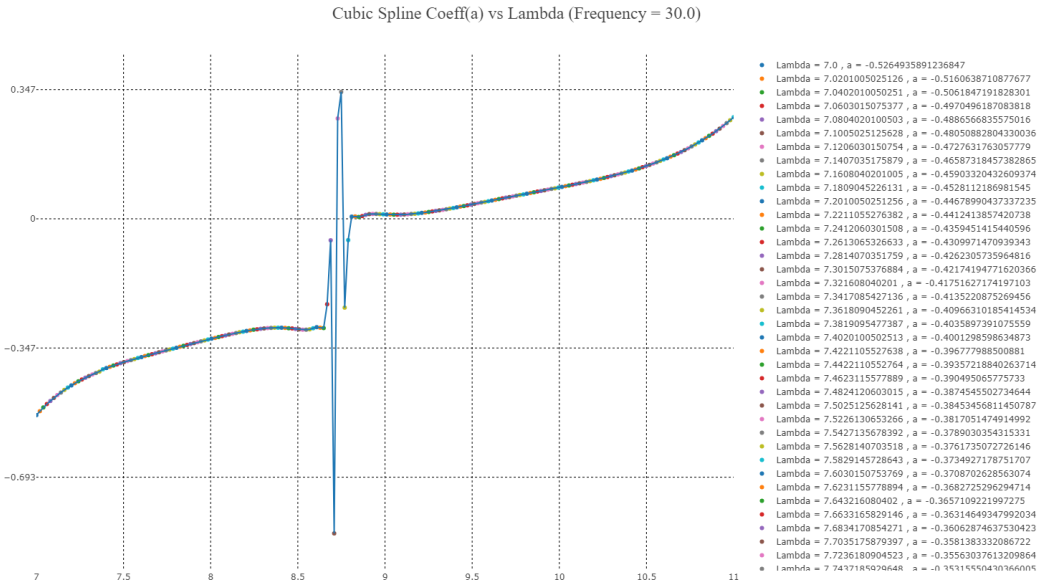


FIGURE 4.21: Cubic Spline Coefficient(a) Vs Lambda (Frequency = 30.0)

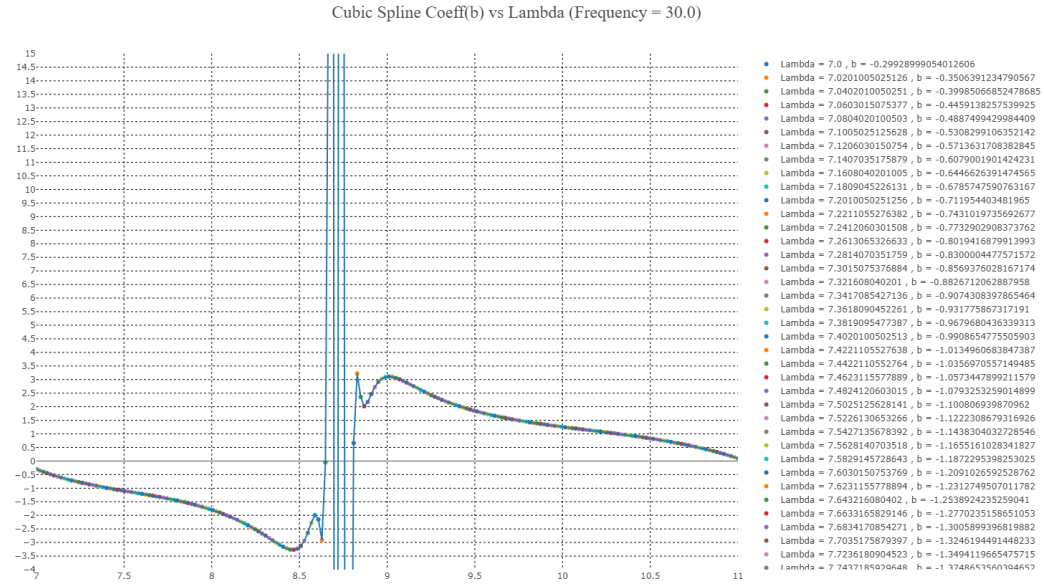


FIGURE 4.22: Cubic Spline Coefficient(b) Vs Lambda (Frequency = 30.0)

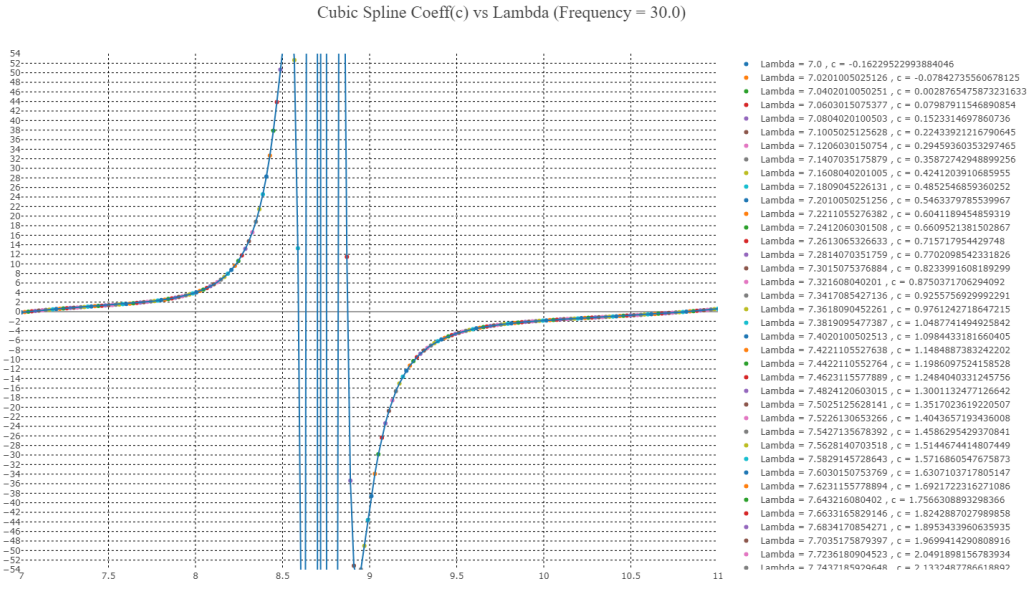


FIGURE 4.23: Cubic Spline Coefficient(*c*) Vs Lambda (Frequency = 30.0)

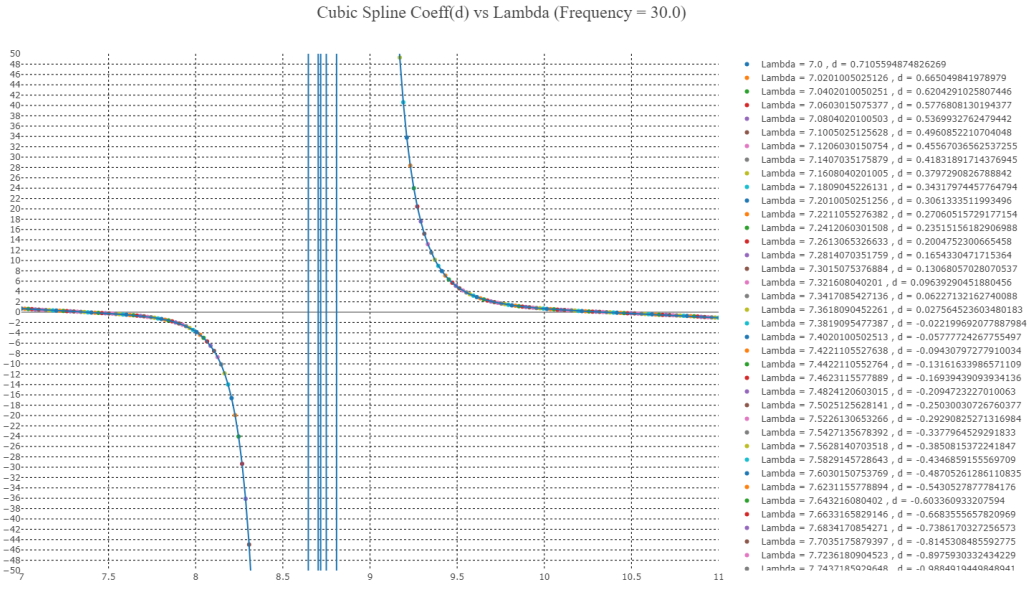


FIGURE 4.24: Cubic Spline Coefficient(*d*) Vs Lambda (Frequency = 30.0)

4.1.4 Evaluation of Quartic Spline:

Spline Coefficient as a function of Lambda:

By adding one more polynomial, as discussed in the section on the cubic spline, we now have a quartic spline with five coefficients, a, b, c, d, and e. If we plot this coefficient as a function of lambda, the curves look something like those shown in the figures 4.25, 4.26, 4.27, 4.28, and 4.29 for the corner frequency of 25.049999237061. The observed plots present a highly irregular, zigzagging pattern, indicating that the quartic spline model may not be the optimal choice for this dataset.

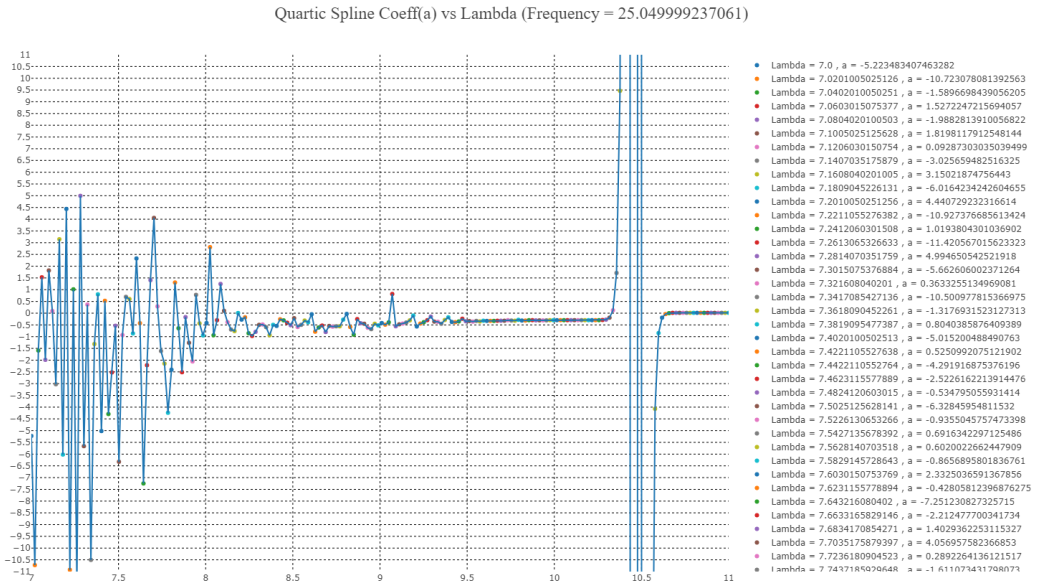


FIGURE 4.25: Quartic Spline Coefficient(a) Vs Lambda (Frequency = 25.049999237061)

As for the middle frequency of 30.0 GHz, the curve looks a bit different. Looking at the figures 4.30, 4.31, 4.32, 4.33, and 4.34, it is clear that having more polynomials does not help generate stable results. Moreover, using the function type $y = f(x)$ approach is not appropriate for our problem statement. We need to look for an alternative approach instead of different polynomial splines that use the $y = f(x)$ function formulation.

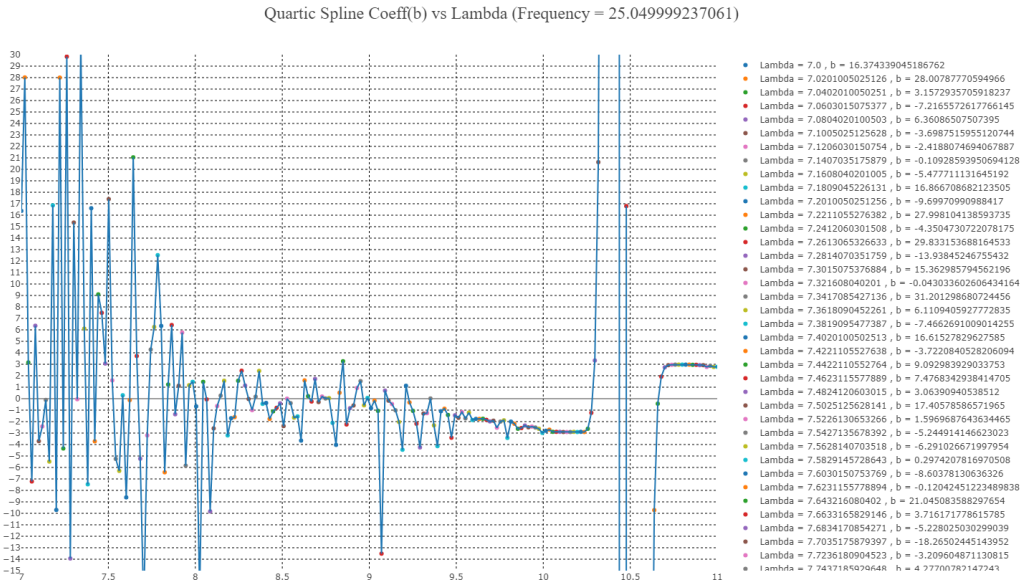


FIGURE 4.26: Quartic Spline Coefficient(b) Vs Lambda (Frequency = 25.049999237061)

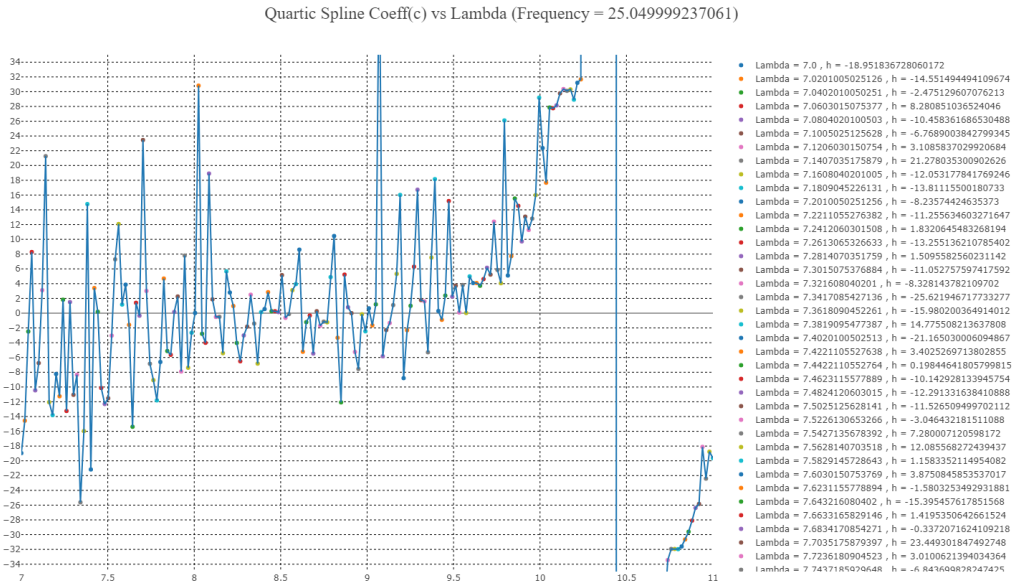


FIGURE 4.27: Quartic Spline Coefficient(c) Vs Lambda (Frequency = 25.049999237061)

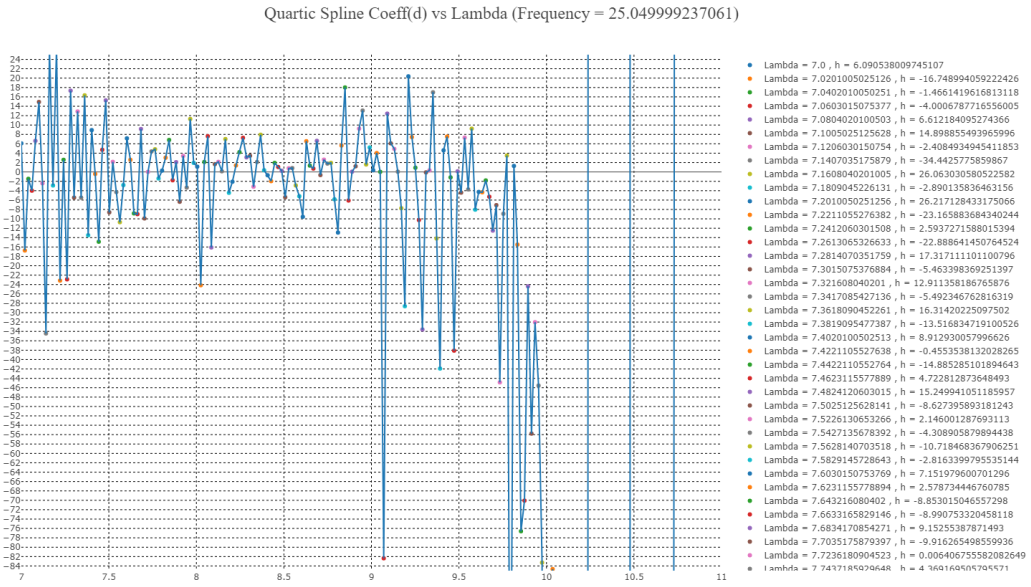


FIGURE 4.28: Quartic Spline Coefficient(d) Vs Lambda (Frequency = 25.049999237061)

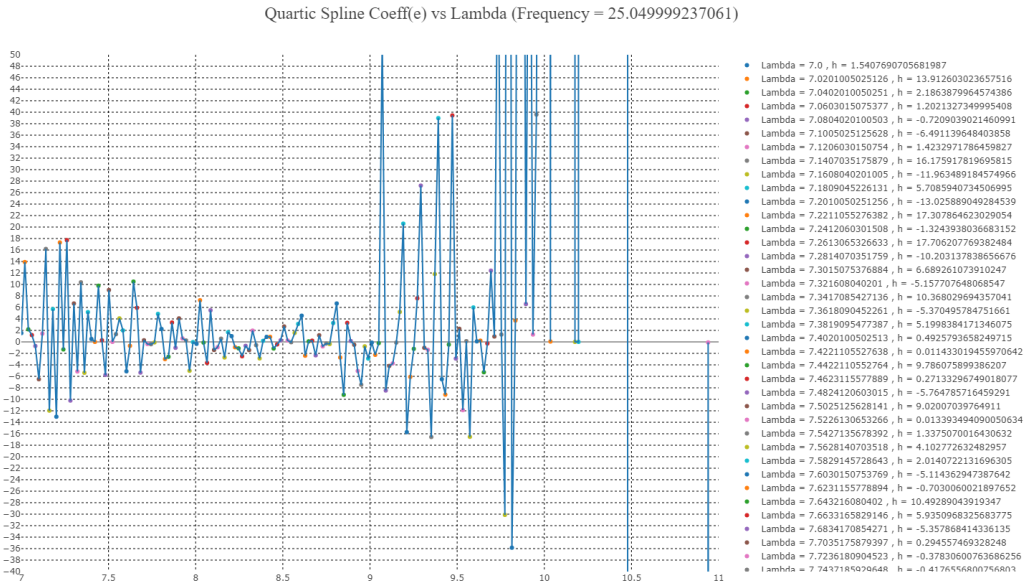


FIGURE 4.29: Quartic Spline Coefficient(e) Vs Lambda (Frequency = 25.049999237061)

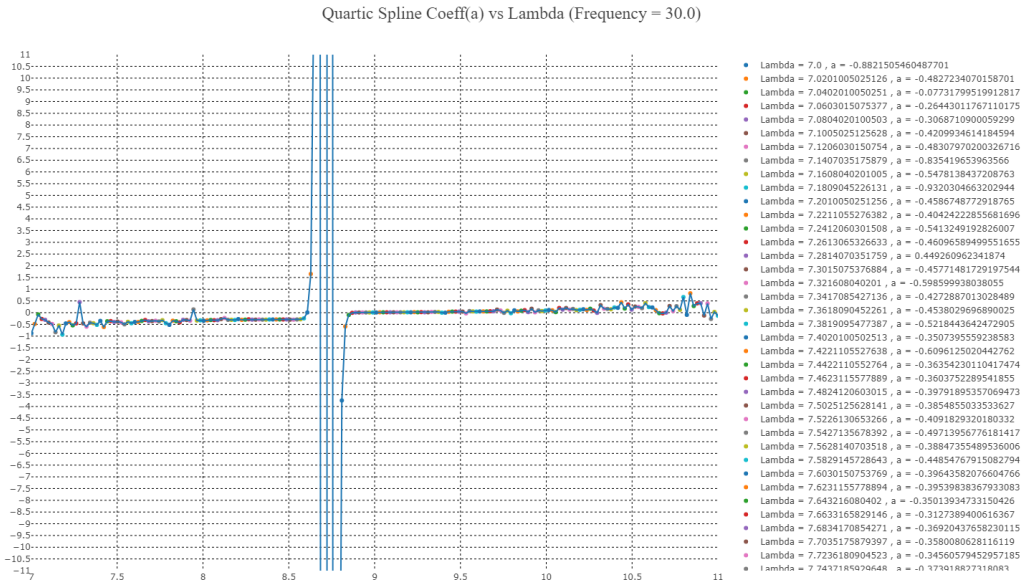


FIGURE 4.30: Quartic Spline Coefficient(a) Vs Lambda (Frequency = 30.0)

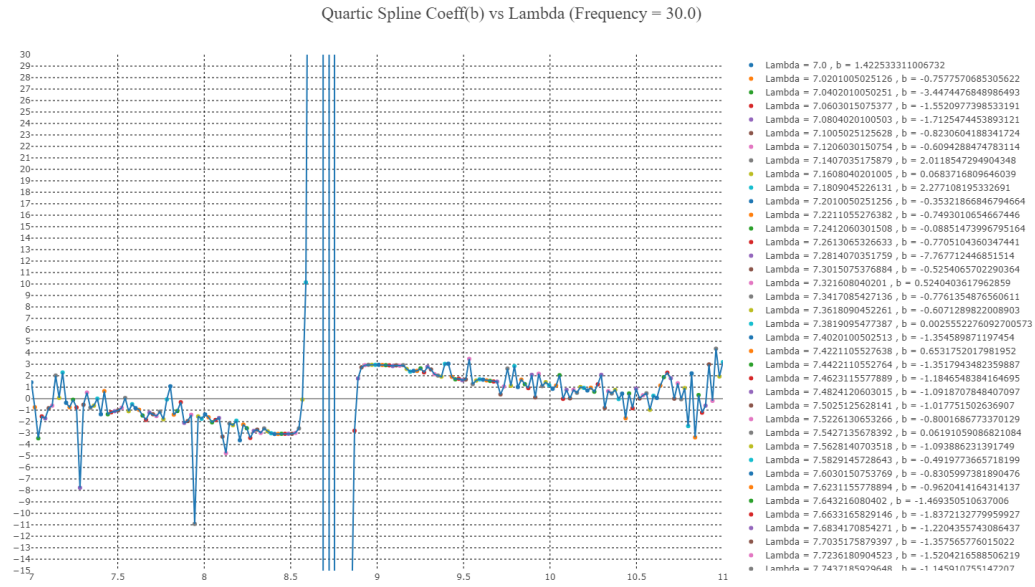


FIGURE 4.31: Quartic Spline Coefficient(b) Vs Lambda (Frequency = 30.0)

4.1.5 Evaluation of Circle Extraction:

The ellipse and spline extraction approaches were based on $y = f(x)$ formulation, which did not yield the desired result, and the circle extraction is also not possible using the same approach as mentioned in chapter 3. So, the new approach of vector function modality ($y = [x(t), y(t)]$) is used, and the results are discussed here.

- **Spline and Extracted Circle:**

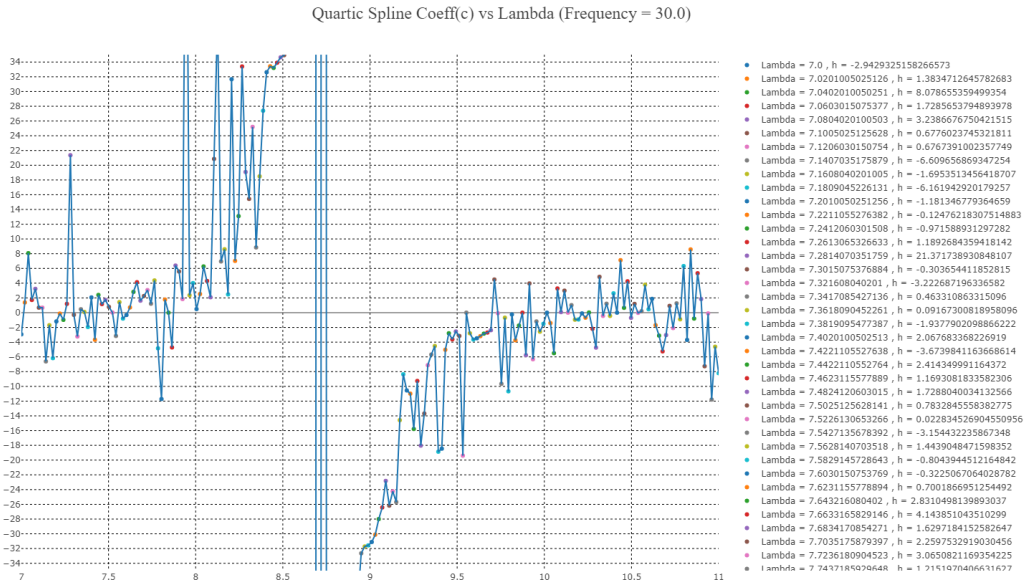


FIGURE 4.32: Quartic Spline Coefficient(c) Vs Lambda (Frequency = 30)

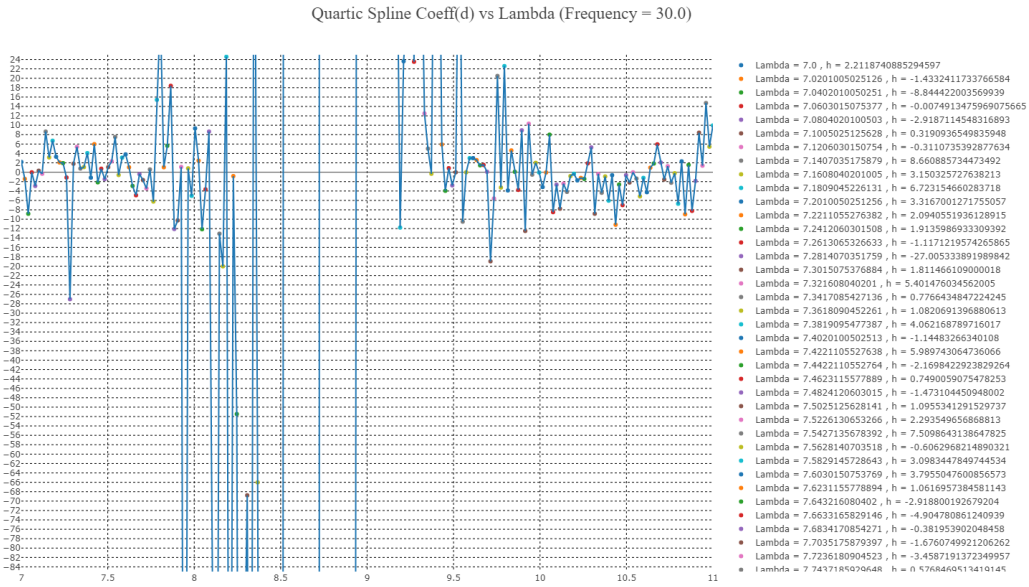


FIGURE 4.33: Quartic Spline Coefficient(d) Vs Lambda (Frequency = 30.0)

Fig. 4.35 shows the original view, and Fig 4.36 shows a zoomed view of the extracted cubic spline and circle for the corner frequency of 25.049999237061. The spline is extracted using the vector function, and the circle passes through the complex parameters at a given frequency of 25.049999237061. Fig. 4.35 and 4.36 are for the lambda value of 7, and the same plots for the lambda value of 11 are shown in Fig. 4.37 and 4.38

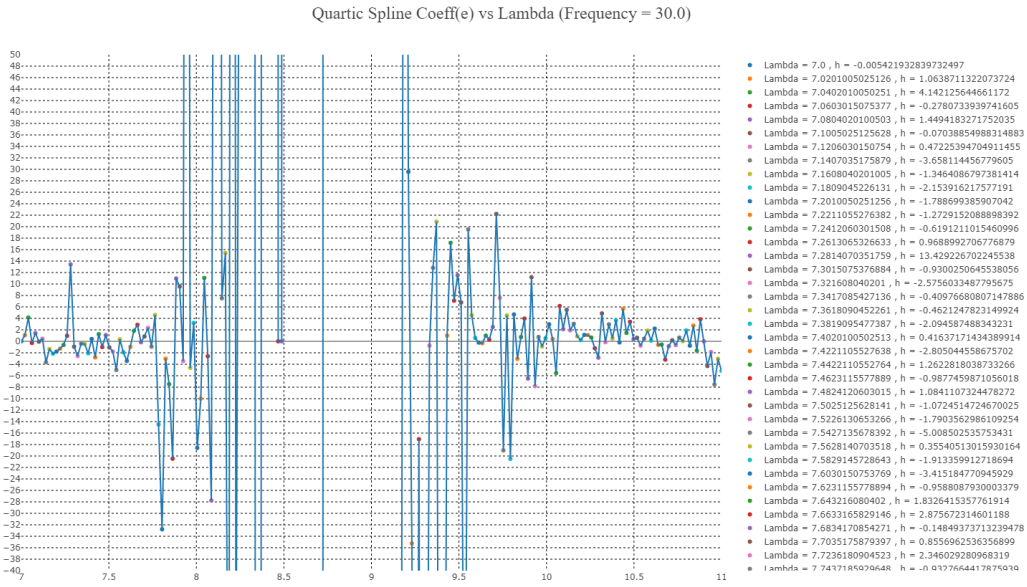


FIGURE 4.34: Quartic Spline Coefficient(e) Vs Lambda (Frequency = 30.0)

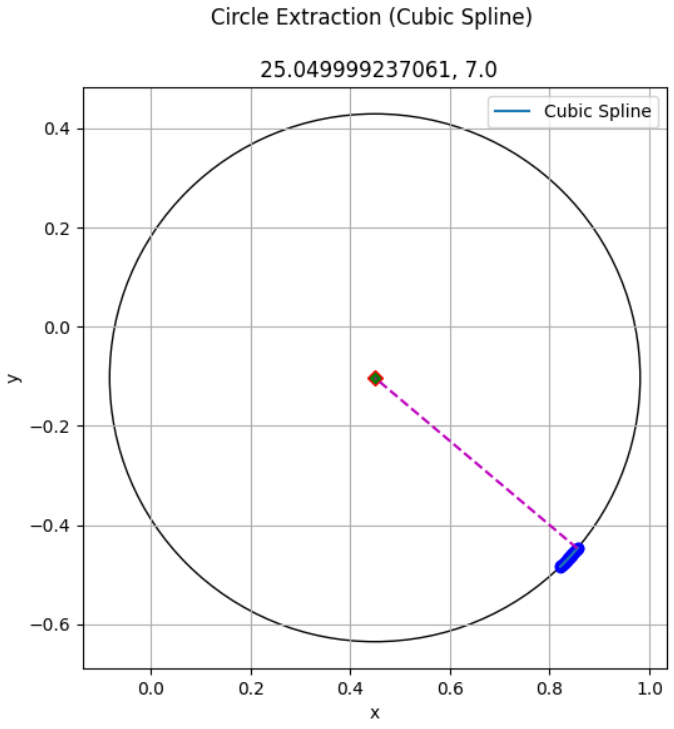


FIGURE 4.35: Circle Extraction Using Cubic Spline (Lambda = 7.0, Frequency = 25.049999237061)

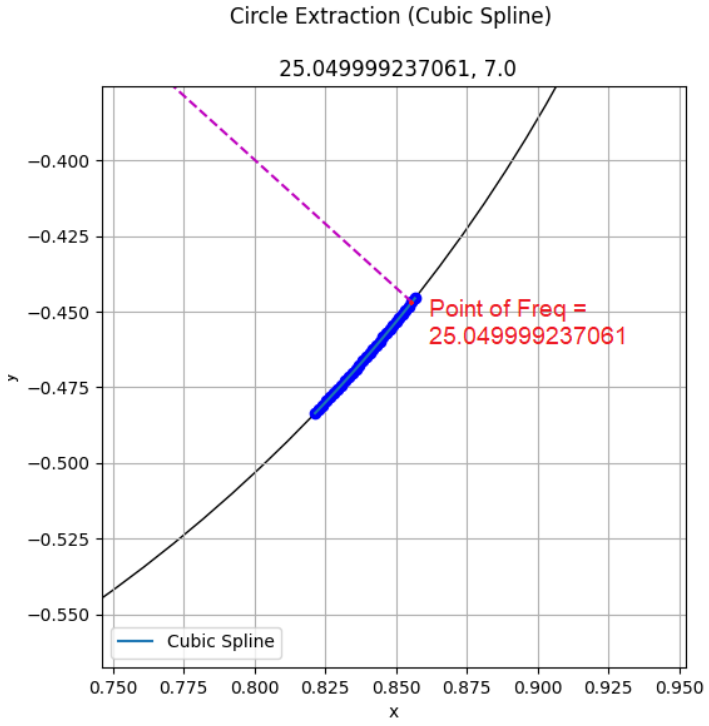


FIGURE 4.36: Circle Extraction Using Cubic Spline (Lambda = 7.0, Frequency = 25.049999237061) [Zoomed View]

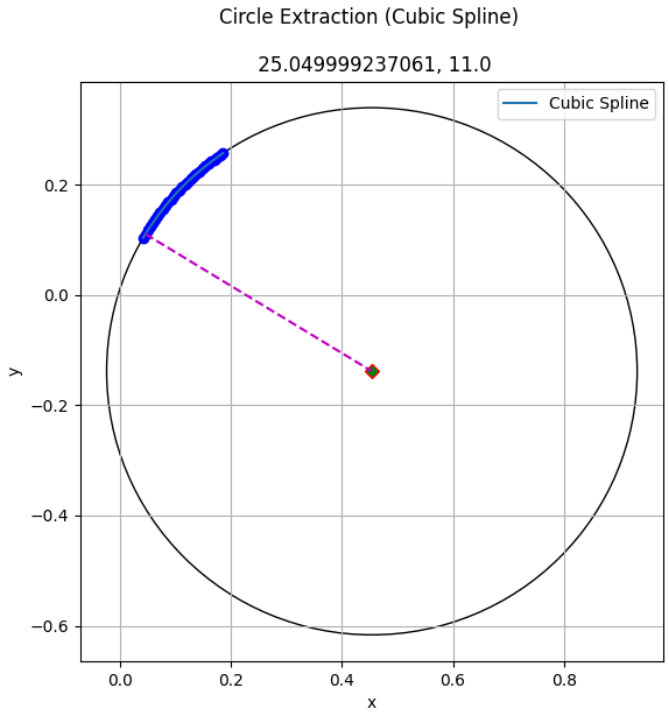


FIGURE 4.37: Circle Extraction Using Cubic Spline (Lambda = 11.0, Frequency = 25.049999237061)

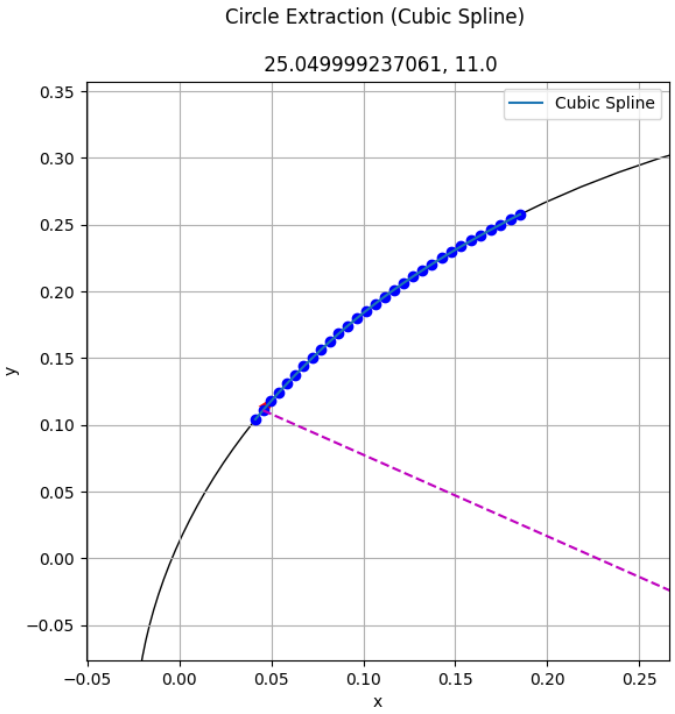


FIGURE 4.38: Circle Extraction Using Cubic Spline (Lambda = 11.0, Frequency = 25.049999237061) [Zoomed View]

Figures 4.39, 4.40, 4.41, and 4.42 show similar plots for the middle frequency of 30.0. The difference between the corner and middle-frequency figures is the neighbors' selection. In the corner frequency, the circle passes through the corner, while in the middle frequency, the circle passes through the middle point of the selected neighbors. Circle extraction is done by finding tangent and curvature as mentioned in the algorithm 4. Using curvature, the center and radius of the circle are found, and the same procedure is followed for all frequency points.

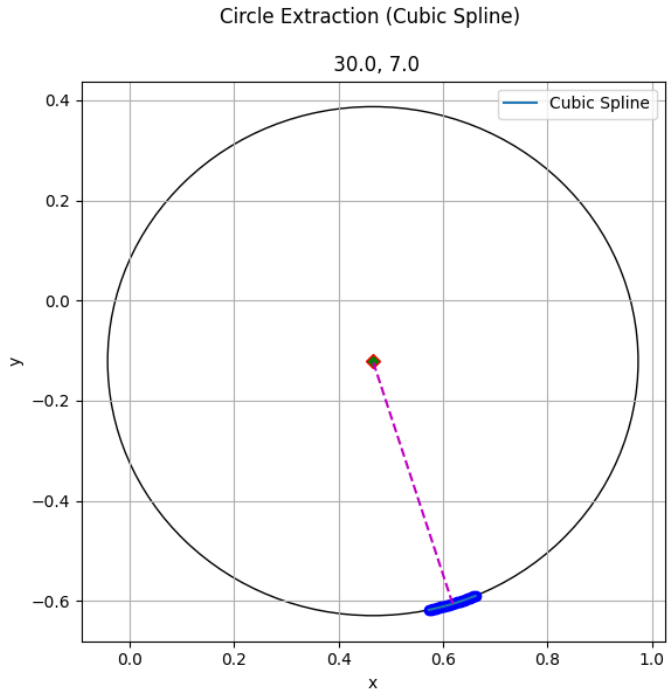


FIGURE 4.39: Circle Extraction Using Cubic Spline (Lambda = 7.0, Frequency = 30.0)

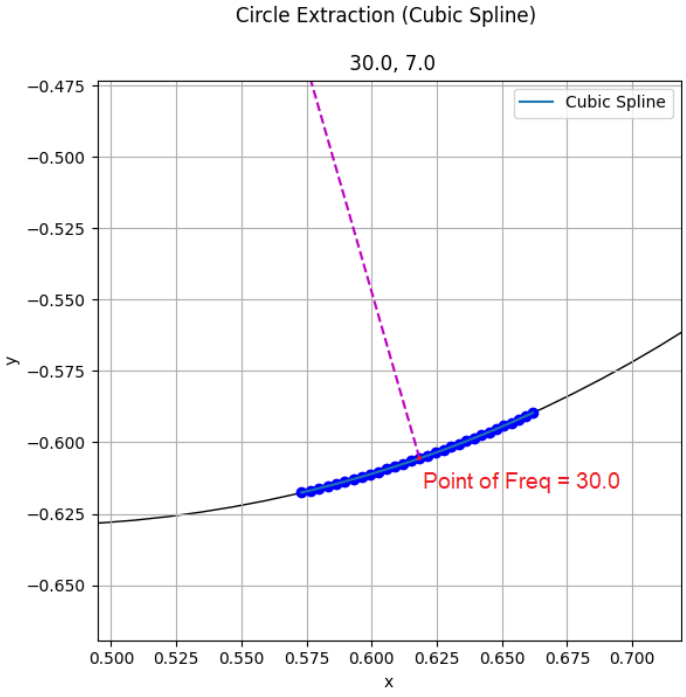


FIGURE 4.40: Circle Extraction Using Cubic Spline (Lambda = 7.0, Frequency = 30.0) [Zoomed View]

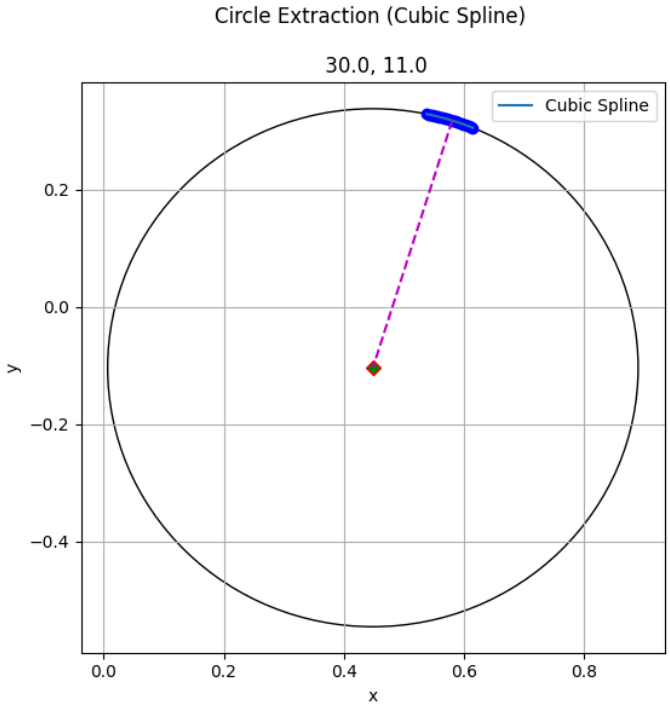


FIGURE 4.41: Circle Extraction Using Cubic Spline (Lambda = 11.0, Frequency = 30.0)

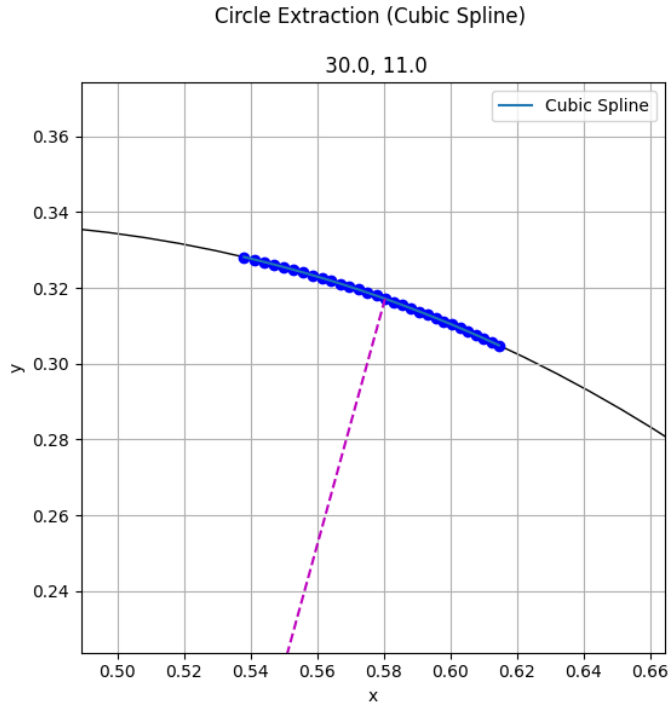


FIGURE 4.42: Circle Extraction Using Cubic Spline (Lambda = 11.0, Frequency = 30.0) [Zoomed View]

- **Circle Parameters as a Function of Lambda:**
  The previous section shows how spline and circle are extracted. Now, let us look at the circle parameters as a function of lambda in this section. Figures 4.43, 4.44, 4.45, and 4.46 illustrates phase, radius, X-Center, and Y-Center vs Lambda plots. In the Phase vs. Lambda plot, the phase varies smoothly as a function of lambda for a given corner frequency of 25.049999237061. That is ideally the primary goal of this thesis, but unfortunately, the results are not smooth for other plots such as Radius, X-Center, and Y-Center.

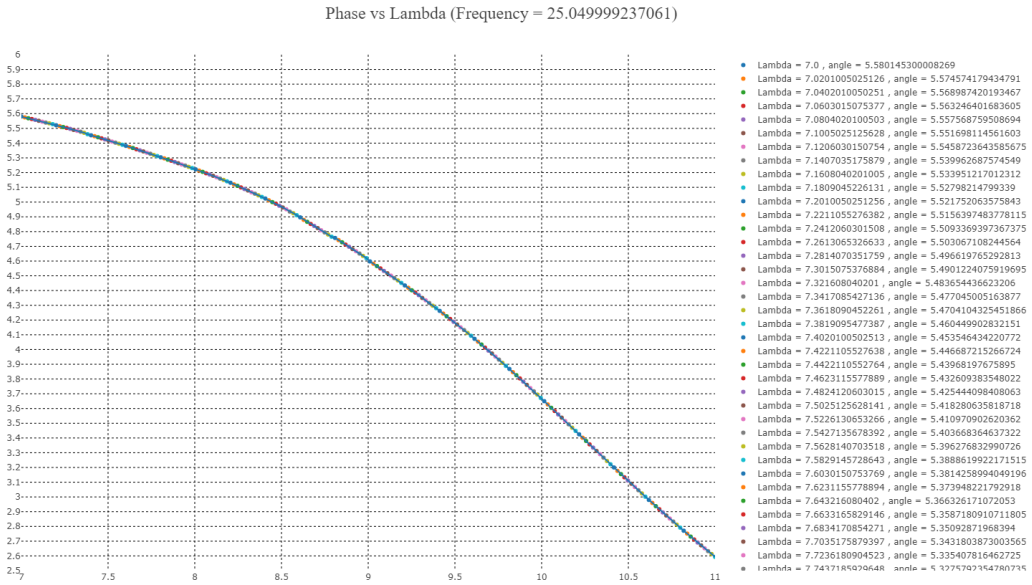


FIGURE 4.43: Phase Vs Lambda (Frequency = 25.049999237061)

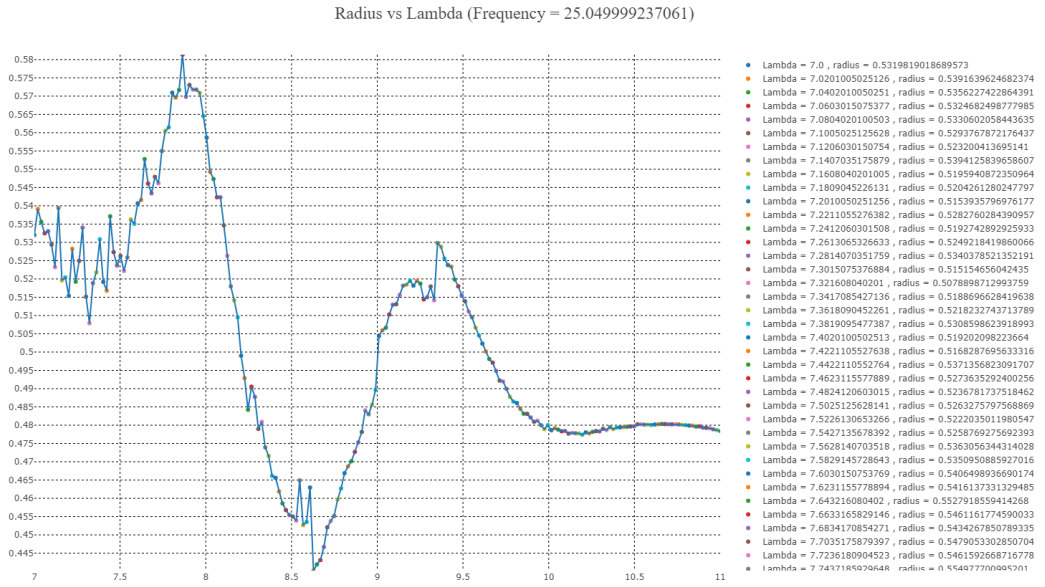


FIGURE 4.44: Radius Vs Lambda (Frequency = 25.049999237061)

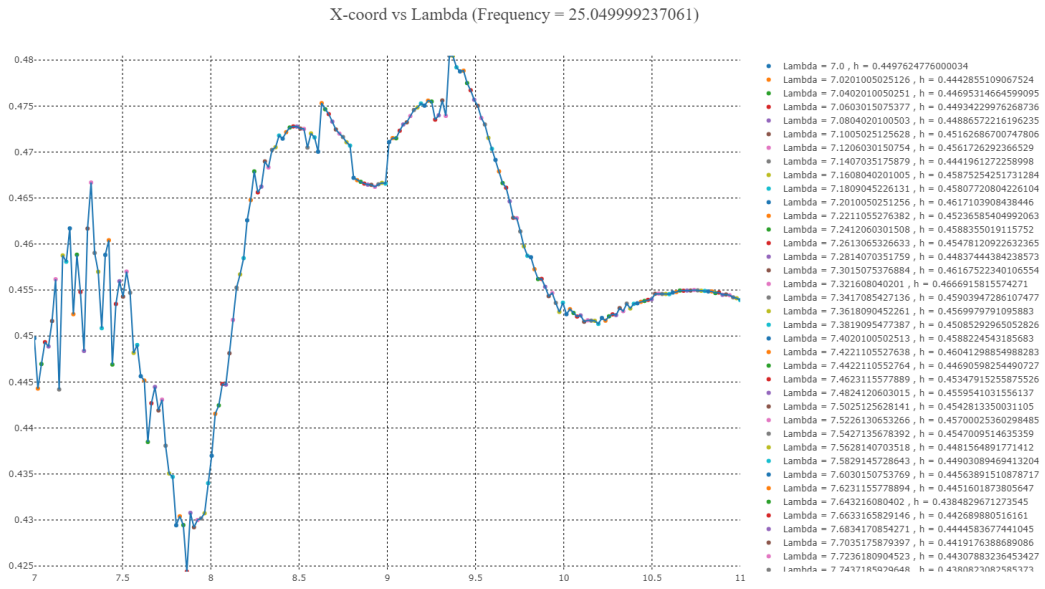


FIGURE 4.45: X-Center Vs Lambda (Frequency = 25.049999237061)

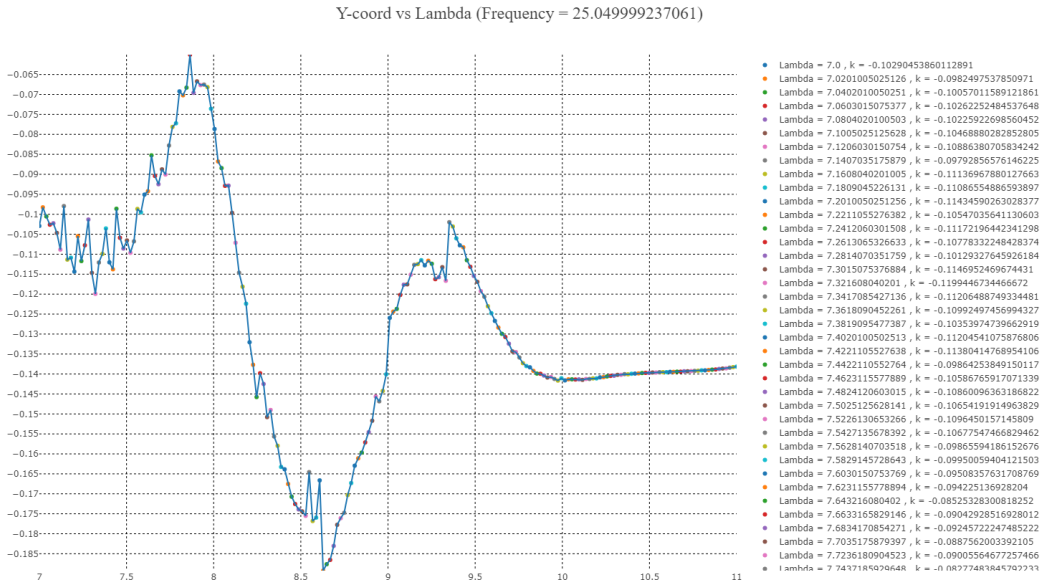


FIGURE 4.46: Y-Center Vs Lambda (Frequency = 25.049999237061)

Next, let us examine the results from the center frequency. Figures 4.47, 4.48, 4.49, and 4.50 depict the plots of circle parameters as a function of lambda for a frequency of 30 MHz. The result is similar to that of a corner frequency; the phase curve is very smooth, while other curves are still better than the corner frequency curves, but ideally, the smoothness should be the same as the phase curve. Here, we have used 30 neighboring frequencies.

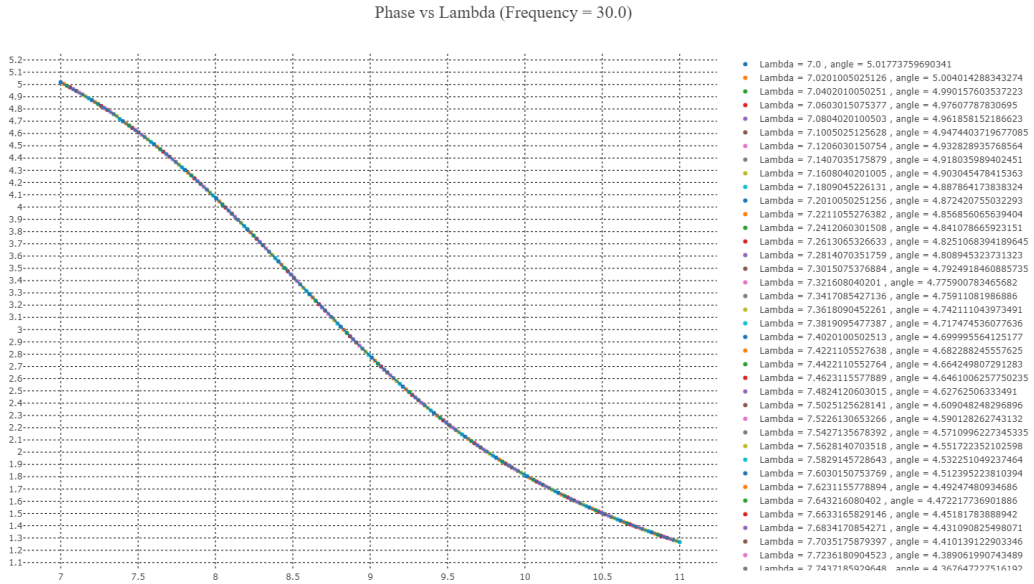


FIGURE 4.47: Phase Vs Lambda (Frequency = 30.0)

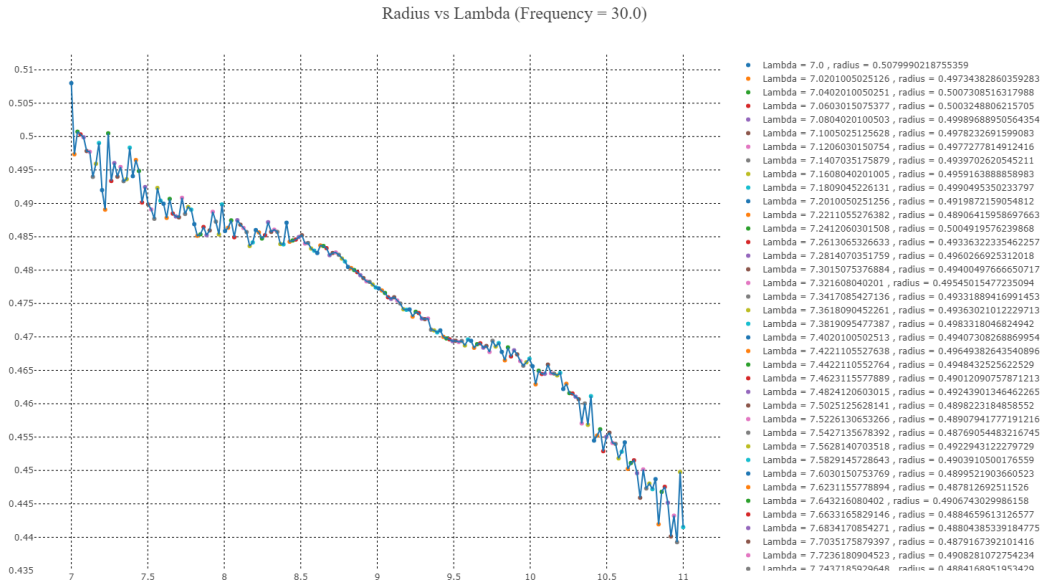


FIGURE 4.48: Radius Vs Lambda (Frequency = 30.0)

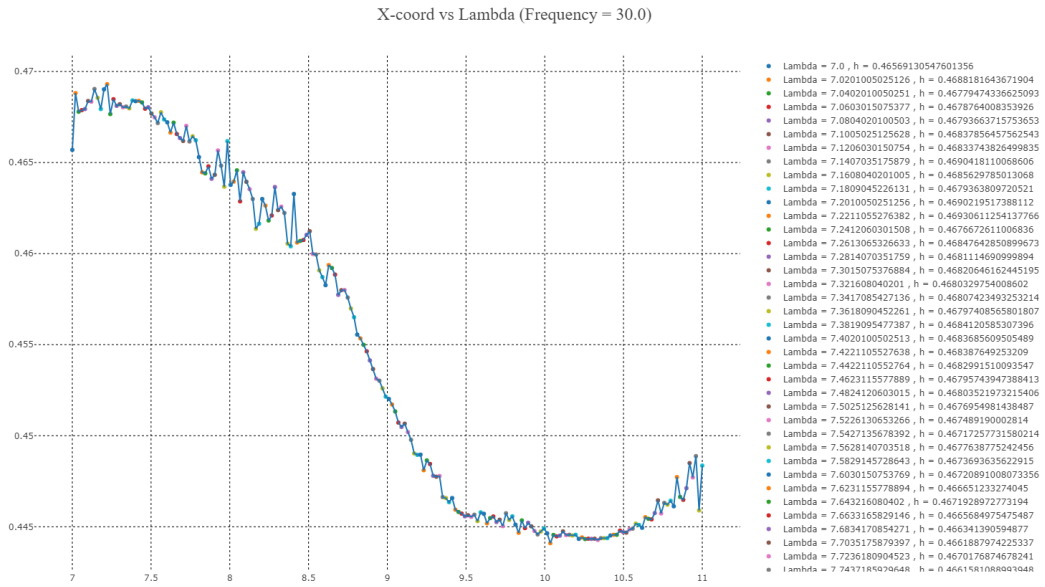


FIGURE 4.49: X-Center Vs Lambda (Frequency = 30.0)

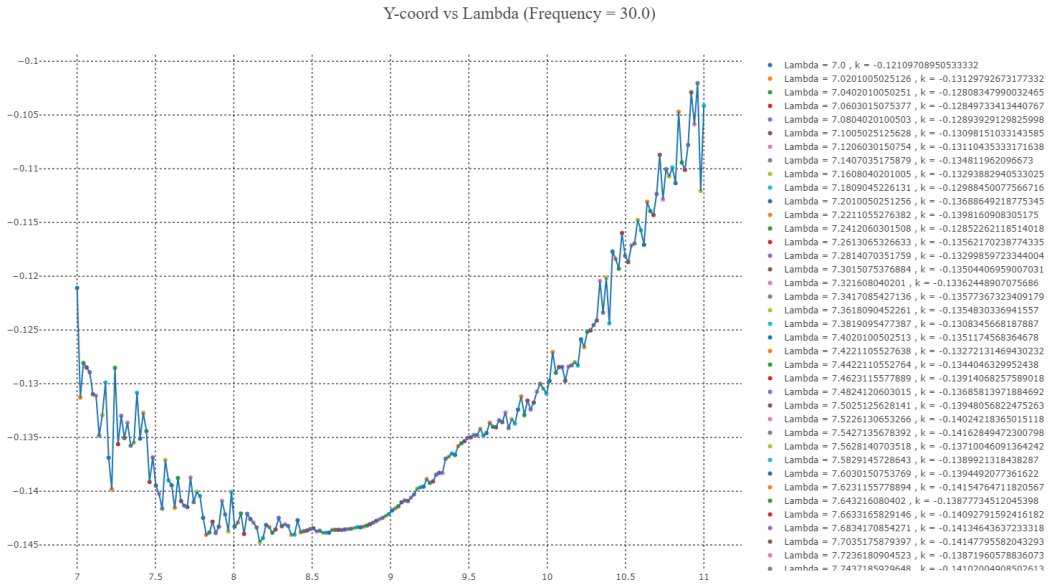


FIGURE 4.50: Y-Center Vs Lambda (Frequency = 30.0)

- **Polar Representation:**
  We want to analyze the underlying problem of circle extraction results from the previous section. To achieve this polar representation for all the extracted circles for the frequency of 25.049999237061 in the lambda range from 7-11 is depicted in Figure 4.51. Figures 4.52 and 4.53 illustrate the polar plot of only two lambdas and their extracted circles. The lambda values of 7.2814070351759 and 7.3015075376884 have their respective radius values of 0.5340378521352191 and 0.515154656042435. Although numerically, these values seem to be pretty close, from Fig. 4.52, it is clear that the difference is significant in our case. A similar case is observed (see Fig. 4.53) in the case of lambda values 8.608040201005 and 8.6281407035176, and their respective radius values are 0.46293778733751817 and 0.44021257580777806.

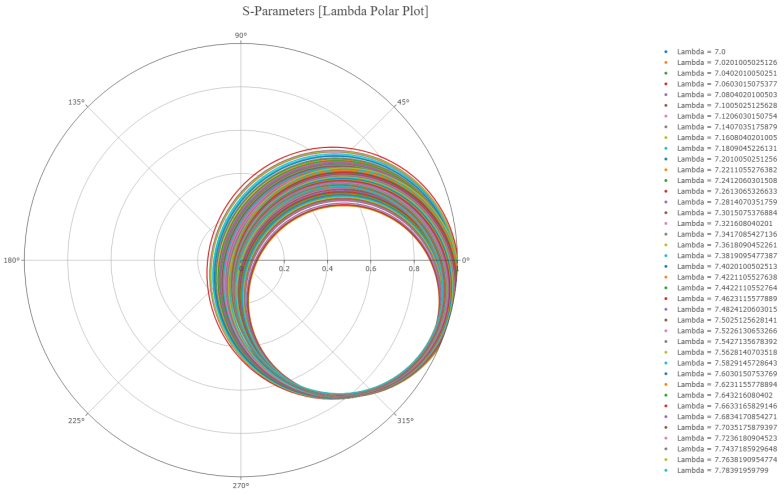


FIGURE 4.51: Polar Plot [Frequency = 25.049999237061

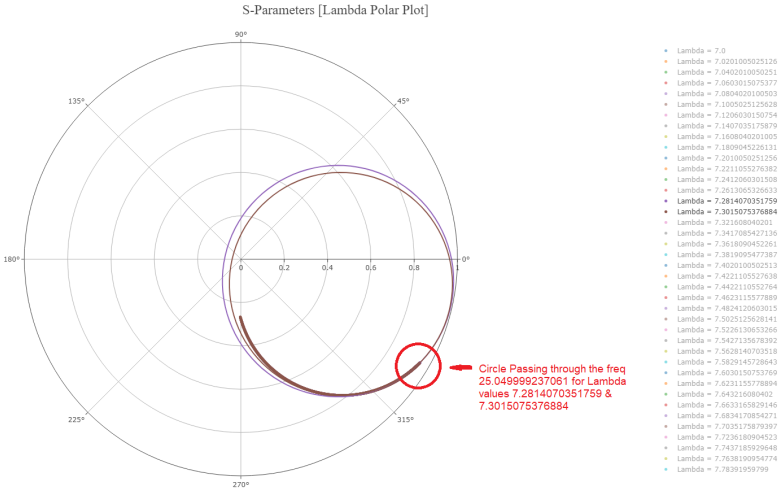


FIGURE 4.52: Polar Plot [Frequency = 25.049999237061, Lambda = 7.2814070351759, 7.3015075376884]

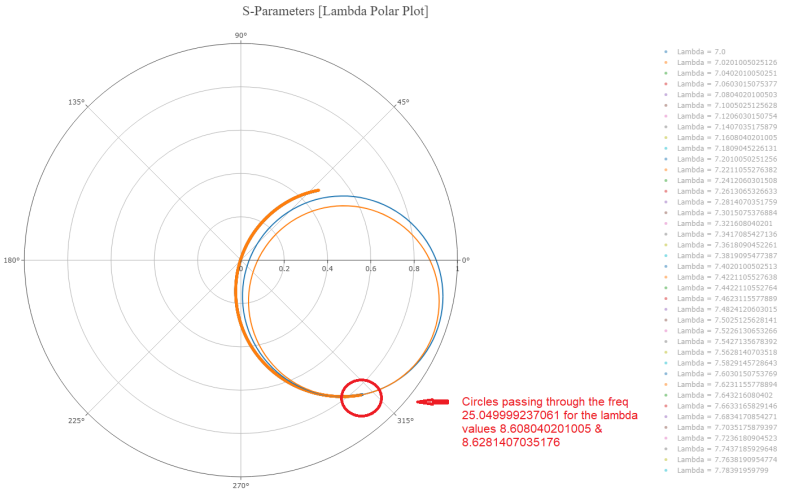


FIGURE 4.53: Polar Plot [Frequency = 25.049999237061, Lambda = 8.608040201005, 8.6281407035176]

Figures 4.51, 4.52, and 4.53 are polar representation of the corner frequency. Now, let us examine the polar representation of the middle frequency 30 MHz. Fig. 4.54 shows the extracted circles for a fixed frequency of 30 MHz and different lambda values. Figure 4.55 shows quite the difference between the radius of two circles extracted for the lambda values of 7.0 and 7.0201005025126. Their respective radius values are 0.507999021965445 and 0.4973438286062687. Another spike is between the lambda values of 7.2211055276382 and 7.2412060301508, where the radius value of the circle extracted for the lambda value of 7.2412060301508 suddenly rises from the radius value of the extracted circle for the lambda value of 7.2211055276382 which can be seen in fig. 4.56 and the radius values are as 0.4890641596106536 for lambda value of 7.2211055276382 and 0.5004919575690623 for lambda value of 7.2412060301508.

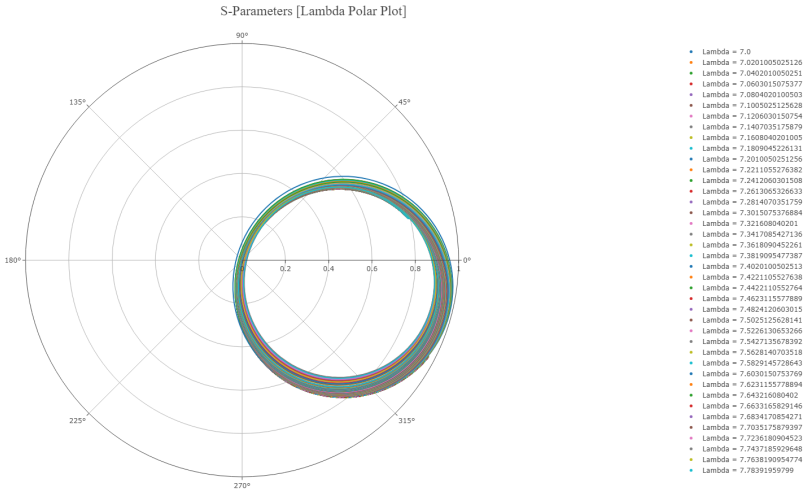


FIGURE 4.54: Polar Plot [Frequency = 30.0

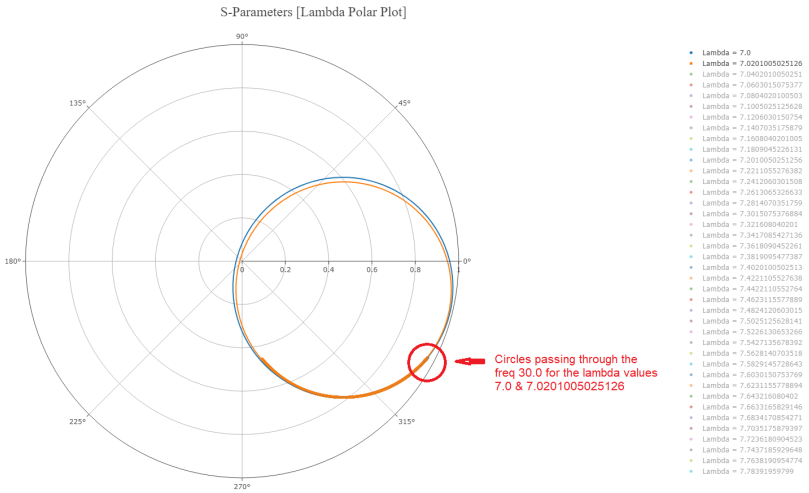


FIGURE 4.55: Polar Plot [Frequency = 30.0, Lambda = 7.0, 7.0201005025126]

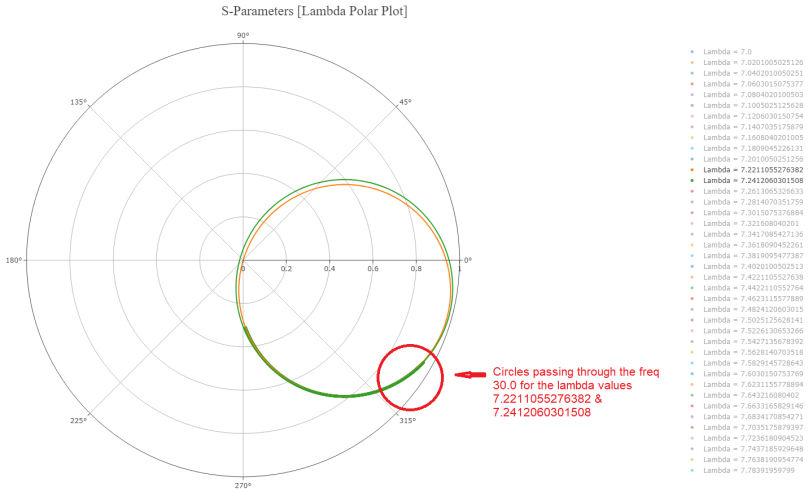


FIGURE 4.56: Polar Plot [Frequency = 30.0, Lambda = 7.2211055276382, 7.2412060301508]

With this, the results of the various experiments performed to improve the smoothness of the curves as a function of lambda using least-square fitting and circle extraction using tangent and curvature on frequency domain simulation data are analyzed. The upcoming section will consolidate the notable findings discovered during this research, extending the previous study performed by Dasi, 2023.

4.2 Discussion

In this section, we will summarize the results gathered throughout this thesis research. These results are in accordance with the problem statement (1.1) and research question (1.3) framed in Chapter 1 of this thesis. These research findings are as follows,

- Several experiments are performed to investigate the stability of least square-fitting approaches such as ellipse fitting, quadratic spline, cubic spline, and quartic spline, and another approach of the circle extraction using tangent and curvature to achieve the desired smoothness of the curves. Starting with the least square ellipse, the parameters are visualized as a function of lambda for corner frequency and center frequency. However, the results are not good as there was no smoothness for both corners and center frequency.

- Then, we decide to use the generic polynomial fitting approach using $y = f(x)$ function formulation, starting with a second-order polynomial referred to as a quadratic spline to fit the data. The curves were smooth for most of the part, but some range was not in conjunction with our research objective. The abnormalities were discovered in the last range of lambda values (10-10.5) for

the starting corner frequency and the middle range of lambda values (8.5-9) for the center frequency.

- Since the second-order polynomial did not work by adding one more order of degree, the cubic spline is used using the same function formulation ($y = f(x)$) to fit the data. Similar results were achieved here as in the quadratic spline, i.e., huge spikes were encountered for the starting corner frequency except for the intercept coefficient in the same range as in the quadratic spline, and for the center frequency, similar behavior in the same range was observed. From both quadratic and cubic spline results, there are some outliers in the gathered data, which causes the issue of fitting the data.

- The quartic spline fitting technique ($y = f(x)$) with four unknown coefficients was investigated to overcome the quadratic and cubic spline limitation. However, the results were utterly abnormal. The plots showed highly irregular patterns for all the coefficients of the quartic spline for both beginning and center frequency. The results indicated that adding more polynomials in the spline equation does not help us cope with the data's outliers and achieve the desired smoothness of the curves.

- Finally, as described in the previous research, by using circle extraction using least square circle fit, we were able to achieve reasonably stable results; we investigated the analytical approach of extracting the circle from the cubic spline using the tangent and curvature. If we use the same functional approach as the polynomial fitting ($y = f(x)$), it will not work for the circle fit, so we used vector calculus to formulate the problem as $y = [x(t), y(t)]$ vector function and extracted the circle parameters using tangent and curvature as a vector function. In the end, we found that only the phase of the circle as a function of the lambda curve had the desired smoothness that ideally we wanted to achieve other parameters radius, X-Center and Y-Center curves are not smooth but rather zigzag in most of the part for both starting corner and middle frequency. Due to noise in the data, the results at the starting point were bad compared to the middle frequency. Since we used least square optimization, the influence of noise made the results unstable at the corner frequencies.

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

The primary objective of this thesis was to achieve the desired smoothness of the curves as a function of lambda by utilizing best-suited data fitting techniques that represent our data to build a model that can fairly predict the moving resonances of FRSs.

The least squares fitting methods were explored, such as least square ellipse, least square splines, and circle extraction using cubic splines. To obtain the parameters of the shape, polynomial, and circle using these techniques, neighboring frequencies information is considered such that we extract the complex-valued S-parameter information of all these frequencies and pass this data to the fitting methods.

In the beginning, the least squares ellipse was studied, and after that, various polynomials such as quadratic, cubic, and quartic spline were studied. The stability of the extracted parameters was not good enough but rather worse than in previous studies. To overcome this problem, the technique is rather similar to previous studies (LSCF); circle extraction was used in conjunction with cubic spline. After extracting the cubic spline for both parameters from complex values S-parameters, we used this equation to create a vector function, and finding the derivative of this vector function gave us tangent at a particular frequency point, which we used to find the curvature and extract circle parameters.

However, the result was not as good as we expected. To analyze the result in further detail, we plotted the circles for each frequency in the polar plot and investigated the jumps that occurred in the circle parameters vs lambda plot. We found that the slight change in the data has a huge impact on the circle parameters, which indicated that the lambda values should not be coupled independently, i.e., we should not use local frequency neighbors from the same lambda value to the represent the moving resonances in FRFs data to build the Surrogate model.

5.2 Future Scope

The investigation into diverse methods for mathematical model fitting has revealed several opportunities for further research and development:

- The quest for a practical mathematical fitting approach that considers all the lambda values and their adjacent frequency values for representing Frequency Response Function (FRF) S-Parameter data is essential. The goal is to develop a technique that can accurately reflect the global trends and nuances of complex-valued S-parameters without the constraints and potential inaccuracies that can arise from relying solely on local frequency parameter data. Such a method would provide a more reliable and comprehensive understanding of the data's behavior over a broad frequency range.
- Discovering a robust and comprehensive mathematical fitting technique will help fulfill the desire for stable results and help us advance in developing more accurate and predictive surrogate models.

Bibliography

Ahn, Sung Joon, Wolfgang Rauh, and Hans-Jürgen Warnecke (2001). "Least-squares orthogonal distances fitting of circle, sphere, ellipse, hyperbola, and parabola". In: *Pattern Recognition* 34.12, pp. 2283–2303. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(00)00152-7. URL: https://www.sciencedirect.com/science/article/pii/S0031320300001527.

Ahn, Sung-Joon (2008). "Geometric Fitting of Parametric Curves and Surfaces". In: *Journal of Information Processing Systems* 4, pp. 153–158. DOI: https://doi.org/10.3745%2FJIPS.2008.4.4.153.

Arlinghuas, Sandra Lach (1994). *PHB Practical Handbook of Curve Fitting*. CRC Press.

Dasi, Monica (May 2023). "Surrogate Modeling Using Complex-Valued Frequency Domain Simulation Data". In: *Master's Thesis Frankfurt UAS, Computer Science and Engineering Department*, p. 124.

Dubey, Y. P. and K. K. Paroha (2014). *Quartic Spline Interpolation*. DOI: 10.5120/15843-4724.

Easum, John A., Jogender Nagar, and Douglas H. Werner (2017). "Multi-objective surrogate-assisted optimization applied to patch antenna design". In: *2017 IEEE International Symposium on Antennas and Propagation USNC/URSI National Radio Science Meeting*, pp. 339–340. DOI: 10.1109/APUSNCURSINRSM.2017.8072212.

Gander, Walter, Gene H Golub, and Rolf Strebel (Dec. 1994). "Least-squares fitting of circles and ellipses". In: *BIT Numerical Mathematics* 34.4, pp. 558–578.

Ghys, Étienne, Sergei Tabachnikov, and Vladlen Timorin (Mar. 2013). "Osculating Curves: Around the Tait-Kneser Theorem". In: *The Mathematical Intelligencer* 35.1, pp. 61–66.

github (2008). *GitHub*. URL: https://github.com/.

Guo, Fenghua (2009). "Data Fitting by Quadratic Splines". In: *2009 International Conference on Measuring Technology and Mechatronics Automation*. Vol. 1, pp. 445–448. DOI: 10.1109/ICMTMA.2009.506.

Hagan, Patrick (2005). "Interpolation Methods for Yield Curve Construction". In.

Halír, Radim and Jana Kostková (2000). "A Spline Approximation of a Large Set of Points." In.

Kong, Qingkai, Timmy Siauw, and Alexandre Bayen (2020). URL: https://pythonnumericalmethods.berkeley.edu/notebooks/chapter17.00-Interpolation.html.

Kraft, D. (1988). *A software package for sequential quadratic programming*. Tech. Rep. DFVLR-FB 88-28. Koln, Germany: DLR German Aerospace Center – Institute for Flight Mechanics.

Mekkawy, Ahmed (2023). "Quadratic Spline Interpolation". In: *MATLAB Central File Exchange*. URL: https://www.mathworks.com/matlabcentral/fileexchange/50622-quadratic-spline-interpolation.

Plass, Michael and Maureen Stone (July 1983). "Curve-Fitting with Piecewise Parametric Cubics". In: *SIGGRAPH Comput. Graph.* 17.3, pp. 229–239. ISSN: 0097-8930. DOI: 10.1145/964967.801153. URL: https://doi.org/10.1145/964967.801153.

Širca, Simon (2016). "Method of Least Squares". In: *Probability for Physicists*. Cham: Springer International Publishing, pp. 227–258. ISBN: 978-3-319-31611-6. DOI: 10.1007/978-3-319-31611-6_9. URL: https://doi.org/10.1007/978-3-319-31611-6_9.

Virtanen, Pauli et al. (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.