Shape-Preserving Interpolation on the Sphere

Iordan M. Iordanov
Advisor: Menelaos I. Karavelas

December 2015

University of Crete, School of Sciences and Engineering
MSc Applied and Computational Mathematics

Contents

Preface

This report contains the formulation and treatment of the problem on which I worked during my M.S. thesis at the University of Crete, Heraklion, under the supervision of Prof. Menelaos Karavelas. This is the result of many hours of frustration, bafflement, coding, writing and plotting, but it also represents the immense satisfaction of achieving something difficult and beautiful. I would like to express my vast gratitude towards Prof. Karavelas who not only entrusted me with the specific task, but also always had a way of guiding me through the difficulties which we inevitably encountered along the way. Be it practical considerations or theoretical background, suggestions and tips, advice and insight, he was always a reliable figure who inspired into me the persistence to come through and kept me on track.

I would also like to express my distinct gratitude to Prof. Eleni Tzanaki for her invaluable contribution to this work. Her help on the progression of the analytical expressions for the asymptotic behavior of the solution, as well as her insight into many of the topics treated in the text have been crucial for the quality of the results and the comprehension of their meaning. The many hours spent with her discussing various aspects of the problem, talking about results and exchanging ideas, contributed to realizing aspects of the problem which would have otherwise remained hidden to me, and I thank her for her patience and kindness.

However, none of this would have come to reality without the people in my life who have always supported me and kept me going – my family. They have always given me the courage to continue when doubt clouded my resolve, and in their own unique manner they have supported me in innumerable ways. Thank you from the bottom of my heart, Mother, Grandma and Kiki.

The presentation of this thesis took place on December 18, 2015 at the University of Crete, Heraklion. The assessment committee was composed of (in alphabetical order) Prof. Theodoulos Garefalakis, Prof. Menelaos Karavelas, and Prof. Michael Plexousakis, all of whom I thank for the availability and the time taken to examine my work. Without further ado, let us proceed to the text itself.

Περίληψη

Μια σημαντική επιθυμητή ιδιότητα των μεθόδων παρεμβολής με πολυώνυμα και με καμπύλες τύπου spline, είναι η ικανότητα να διατηρούν το σχήμα που φαίνεται να έχουν τα αρχικά δεδομένα εισόδου. Στη γενική περίπτωση όμως, δεν υπάρχει κάποια εγγύηση ότι η παρεμβάλλουσα καμπύλη που παράγουν αυτές οι μέθοδοι θα συνεχίζει να έχει αυτό το σχήμα. Αυτός είναι ο λόγος για τον οποίο έχουν προταθεί νέες μέθοδοι παρεμβολής οι οποίες συμπεριλαμβάνουν ελεύθερες μεταβλητές. Ο έλεγχος των τιμών αυτών των παραμέτρων είναι σε θέση να προκαλέσει την ικανοποίηση κάποιων περιορισμών που αφορούν το σχήμα της παρεμβάλλουσας. Ανάμεσα σε αυτές τις καινούριες μεθόδους συναντάμε και τις *μεθόδους τάσης* οι οποίες χρησιμοποιούν τις ελεύθερες παραμέτρους έτσι ώστε η παρεμβάλλουσα να τείνει σε μια κατά τμήματα γραμμική καμπύλη η οποία να παρεμβάλλει τα δεδομένα σημεία. Με αυτό τον τρόπο, οι απαιτήσεις διατήρησης σχήματος ικανοποιούνται τετριμμένα.

Στην παρούσα εργασία, διατυπώνουμε και υλοποιούμε μια μέθοδο παρεμβολής σημείων στη μοναδιαία σφαίρα $\mathbb{S}^2$. Η παρεμβάλλουσα καμπύλη μας είναι μια σφαιρική ν- spline, μια G^2-συνεχής κατά τμήματα κυβική καμπύλη η οποία ανήκει στην οικογένεια των μεθόδων τάσης, και η οποία «ζει» πάνω στη μοναδιαία σφαίρα. Η ασυμπτωτική συμπεριφορά της καμπύλης για πολύ μεγάλες τιμές των παραμέτρων τάσης μας δίνει έναυσμα για την διατύπωση του αλγορίθμου που παρουσιάζουμε. Ο αλγόριθμος είναι σε θέση να καθορίσει αυτόματα την κατάλληλη τιμή για κάθε παράμετρο τάσης έτσι ώστε η παρεμβάλλουσα καμπύλη να διατηρεί το σχήμα των δεδομένων σημείων πάνω στη σφαίρα. Η διατύπωση του αλγορίθμου, η υλοποίησή του σε γλώσσα προγραμματισμού C++, καθώς και αποτελέσματα για επιλεγμένες περιπτώσεις δοκιμής, παρουσιάζονται στο τέλος της εργασίας.

Chapter 1

Abstract

An important desirable trait of polynomial and spline interpolation schemes is the ability to preserve the shape suggested by the discrete input data. In the general case, however, no guarantee exists that the resulting interpolant will bear these shape-preserving traits. Therefore, new interpolation schemes, endowed with free parameters that can be adjusted to satisfy the shape-preservation constraints, have been proposed and developed. Among these methods we find the tension schemes which employ free parameters to cause a smooth interpolant to convergence towards a piecewise linear curve connecting the data points, thus trivially satisfying the requirements tied to shape-preservation.

In the present work we formulate and implement a method for interpolating data points lying on the unit sphere $\mathbb{S}^2$. Our interpolant is a spherical ν-spline, a G^2-continuous piecewise-cubic curve which belongs to the family of tension curves and lives on the unit sphere. The asymptotic behavior of the ν-spline for very large values of the tension parameters motivates the formulation of an algorithm which is able to determine the value for each tension parameter so that the resulting curve preserves the shape of the input points on the sphere. The algorithm, its implementation in C++, and the results from selected test cases are presented at the end of this thesis.

Chapter 2

Introduction

In the sections of this chapter we will present some of the basic concepts upon which this work is based. Examples are given on selected topics, and definitions are introduced for less common concepts which will be needed later.

2.1 Linear interpolation

Suppose that we have a high-precision electronic tracking system set to record the location of a remote control toy car every second. Our measurements may not be sufficient in order to fully describe the behavior of the toy car as it moves, or perhaps we need to know the location of the car for a time when we were unable to take a measurement. We are, however, able to *estimate* intermediate locations based on our measurements, and the process is known as *interpolation*. Let us consider a concrete example – suppose we have the function $f(t) = \frac{1}{1-t^2}\sin(t)$ which describes the distance of our toy car from a fixed reference point in space. Let us pretend that we do not know the formula of this function, but we do have some measurements, as shown in the figure below.

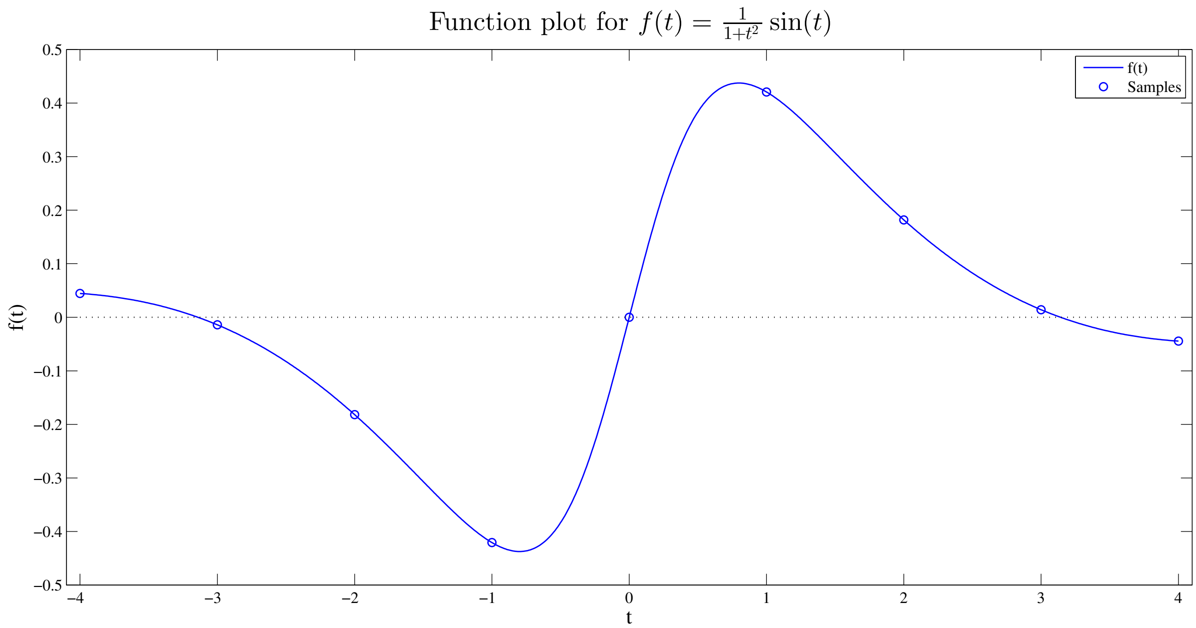


Figure 2.1: *Sample measurements of* $f(t) = \frac{1}{1-t^2}\sin(t)$ *for* $t = -4, -3, ..., 4$.

The simplest thing one could do in order to interpolate the sample points on the curve is to connect the dots. Formally, for every pair of consecutive measurements (or points on the plot) $(t_i, f(t_i)), (t_{i+1}, f(t_{i+1}))$, we seek a linear polynomial of the form

$$v = at + b, \quad a, b \in \mathbb{R}$$

which passes through $(t_i, f(t_i)), (t_{i+1}, f(t_{i+1}))$. It is not dificult to see that the polynomial we seek is given by the equation

$$g(t) = f(t_i) + \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i}(t - t_i).$$

Setting $t = t_i$ in the above equation yields $g(t_i) = f(t_i)$, while setting $t = t_{i+1}$ yields $g(t_{i+1}) = f(t_{i+1})$, so the linear polynomial $g(t)$ indeed *interpolates* $(t_i, f(t_i))$ and $(t_{i+1}, f(t_{i+1}))$.

The expression for $g(t)$ can be reformulated into a more convenient form. Considering $q_i = (t_i, f(t_i))$ and $q_{i+1} = (t_{i+1}, f(t_{i+1}))$, we can express the equation in *parametric form* as

$$p_i(\tau) = (1 - \tau)q_i + \tau q_{i+1}, \quad \tau \in \mathbb{R}.$$

Here the parameter τ is any real number, so we can calculate any point on the line passing through q_i and q_{i+1}. If, however, we confine τ in the interval $[0, 1]$, we get the parametric expression for the linear segment $[q_i, q_{i+1}]$ which is extremely convenient. It is easily verified that $p_i(0) = q_i$ and $p_i(1) = q_{i+1}$, hence this expression indeed defines a linear polynomial with respect to τ which interpolates the pair of points q_i, q_{i+1}. Repeating this procedure for all pairs of measurements, we end up with a piecewise-linear polynomial which interpolates every consecutive pair of sample points, as shown in the following figure.

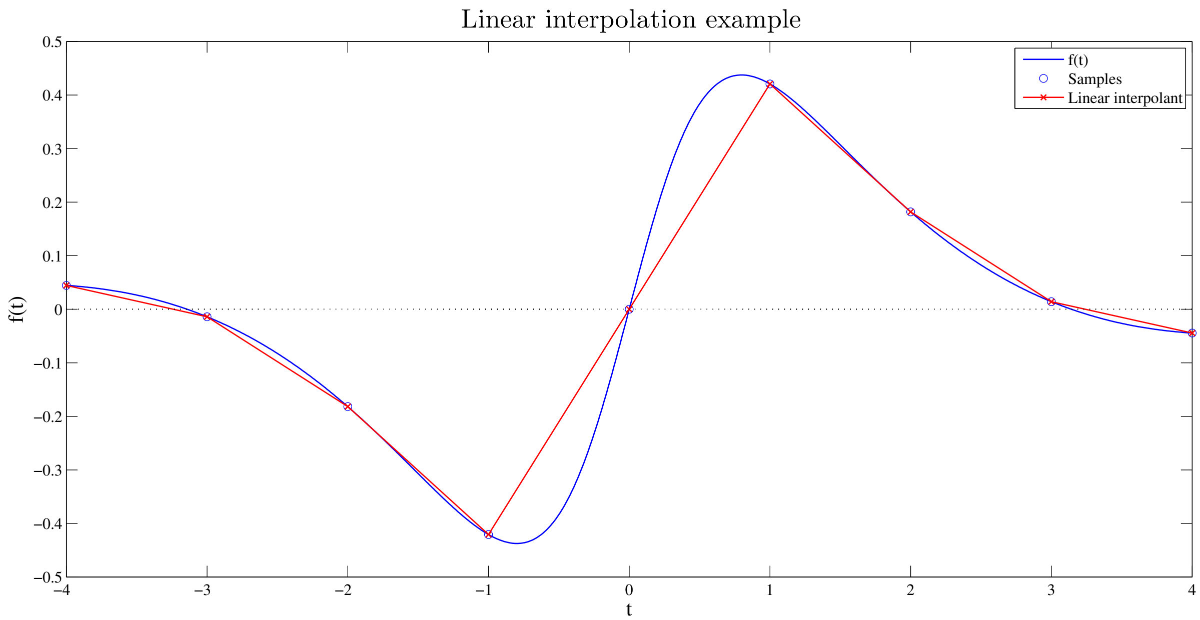


Figure 2.2: *Interpolation of the sample points with piecewise linear polynomials.*

It becomes evident from the last plot that there are significant errors in our approximation, and it can be proven that the error is proportional to the squared distance between the data points t_i. Moreover, the piecewise linear polynomial produced is not differentiable at the data points, so it doesn't look like the toy car could be able to actually follow this estimated route. It is unnatural to change direction so abruptly, we would expect a smoother motion. We can do something better which will make more sense, but for this we will need to look at splines of higher order.

2.2 Splines

The term *spline* was coined approximately in the middle of the 18th century and it is speculated that it originates from the word *splinter*. Initially, splines were flexible pieces of wood or metal, used in the construction of ships and bows. Nowadays their use is limited in this context, however their homonymous counterparts in Mathematics are powerful tools with very useful properties.

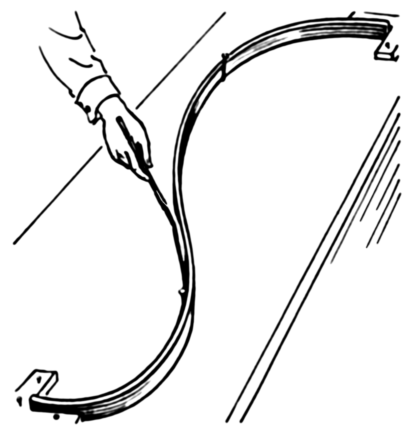


Figure 2.3: *Spline used in the construction of a curve. Note the use of wedges at control points.*

Formally, the spline is a piecewise-continuous sequence of functions which satisfies certain conditions, such as derivative(s) continuity at the interpolation nodes. The most widely used variant are the polynomial splines, the most common being the cubic spline (i.e., piecewise continuous polynomials of degree 3 with continuity requirements at the interpolation points for the first and second derivative). The continuity requirements translate into a smooth curve without sharp variations at the interpolation points. Based on this definition, it becomes clear that in the example of the previous section, we actually constructed a piecewise linear spline, which only requires continuity of the curve itself at the interpolation nodes. Requiring continuity of the first and second derivative in addition yields a significantly smoother curve. The comparison of the two techniques can be seen in figure 2.4.

It is evident, even from this simple example, that the cubic interpolant is much more representative of the movement of the remote-controlled toy car as the transitions at the interpolation points are much smoother. Our interpolant does not completely match the "real" function $f(t)$, however, and this is because we have not imposed other requirements on the spline. One solution is to acquire different samples which will allow for better approximation of $f(t)$. Another idea would be to incorporate free parameters into the spline itself, so that based on their value the resulting curve has a different shape, either more "relaxed" or "tense".

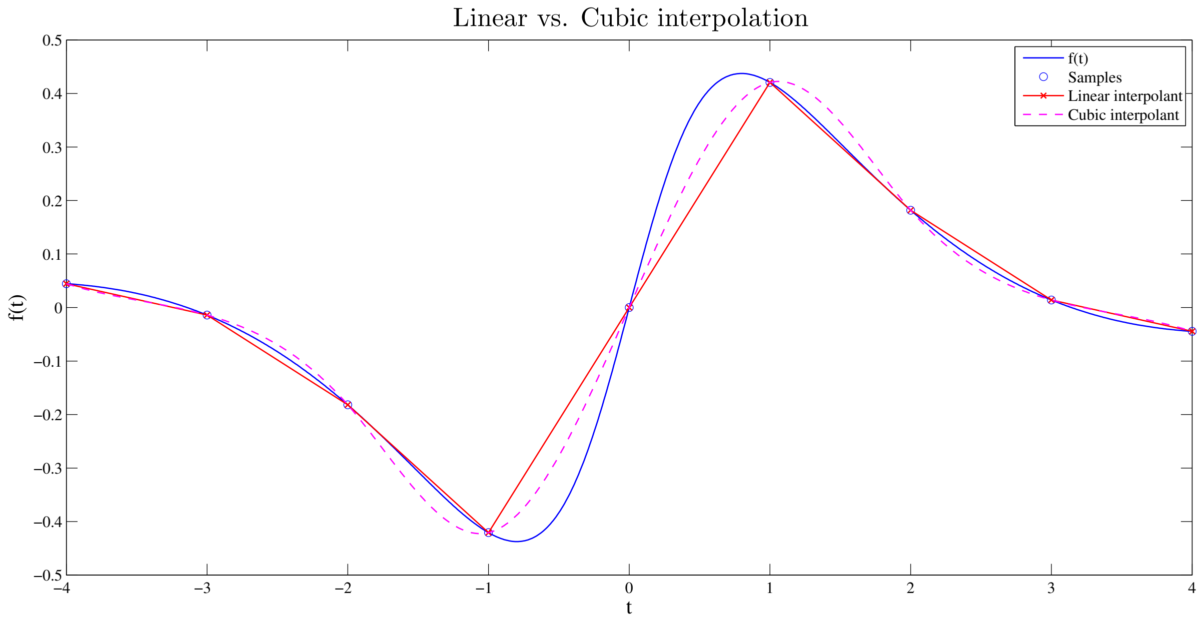


Figure 2.4: *Comparison of interpolation using linear and cubic splines.*

Our work focuses on cubic ν-splines which belong to the family of parametric splines defined by tension parameters ν (hence the name). The parameters ν are defined for each interpolation node and affect how relaxed or tense the curve is at that node – large values for the ν parameters cause the spline to tighten, while parameters close to 0 produce more relaxed curves. In fact, when all tension parameters are equal to 0, ν-splines reduce to piecewise-cubic splines. The concept is easily perceived if we return to the origin of splines: suppose that the spline is actually a flexible piece of metal, wire, and at every control point there is a winch. The tension values express how many times each winch has been turned. An example regarding ν-splines living on the unit sphere is given in figure 2.5.

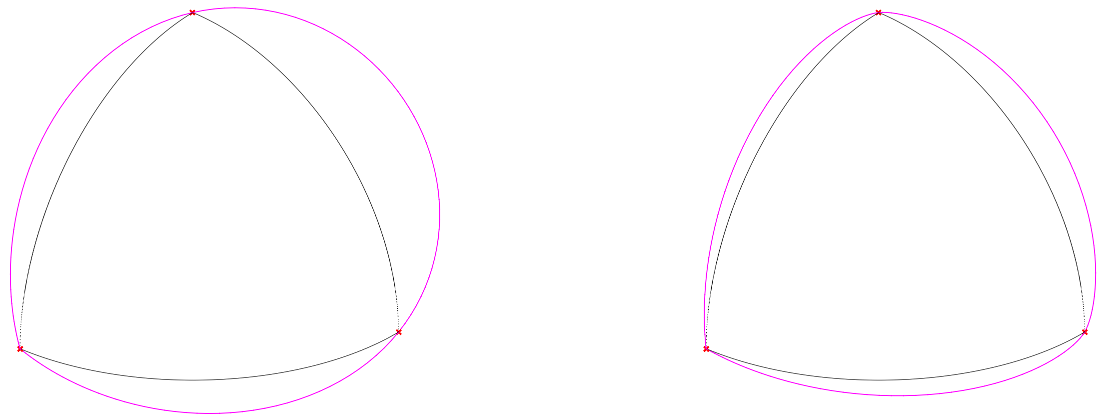


Figure 2.5: *Comparison between spherical splines for ν-values equal to 0 (left) and 10 (right). Note that the ν-spline to the right is closer to the spherical triangle defined by the three interpolation points.*

2.3 The de Casteljau algorithm

Figure 2.6: *Paul de Casteljau*

Paul de Faget de Casteljau is a French physicist and mathematician, born in 1930. In 1959, while in the employment of Citroën, he developed a recursive method for computing points on a particular set of curves which were later formalized and popularized by engineer Pierre Bézier.

The algorithm developed by de Casteljau can be applied to recursively compute points on Bézier curves of arbitrary rank. It relies on successive linear interpolations, eventually resulting in a point which lies on the desired curve. For instance, in order to evaluate a cubic Bézier curve at a certain value, we need four control points to define it, and the algorithm recurs 3 times. Let us use an example to illustrate the idea. Suppose the input of four control points P_i, $i = 0, 1, 2, 3$. For every value of the parameter $t \in [0, 1]$, compute the intermediate points

$$P_{i,i+1} = (1 - t)P_i + tP_{i+1}, \quad i = 0, 1, 2$$

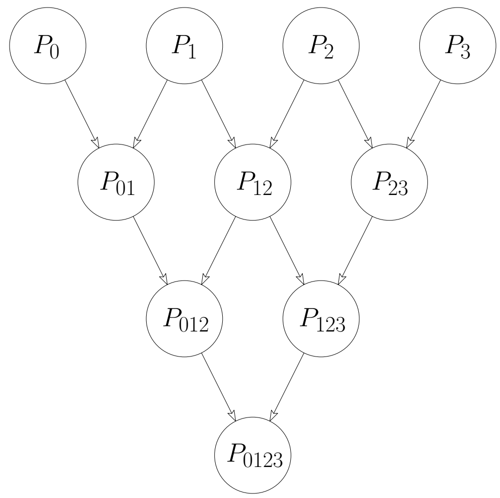
which are then used in the same way to compute the intermediate points of second order

$$P_{i,i+1,i+2} = (1 - t)P_{i,i+1} + tP_{i+1,i+2}, \quad i = 0, 1$$

and finally, the third-order approximation is computed as

$$P_{i,i+1,i+2,i+3} = (1-t)P_{i,i+1,i+2} + tP_{i+1,i+2,i+3}, \quad i = 0.$$

Schematically, the procedure can be illustrated in the following directed acyclic graph.



An example of a cubic Bézier curve in 3-dimensional space is given below. Note that the algorithm can be generalized to arbitrary dimension, as it relies only on the notion of linear interpolation. If we wish to determine a point on a Bézier curve of order n in a d-dimensional space, we need $n+1$ control points in this d-dimensional space. The procedure is applied as shown in the previous figure. An actual example can be seen in 2.7, where point annotation follows the notation used so far.

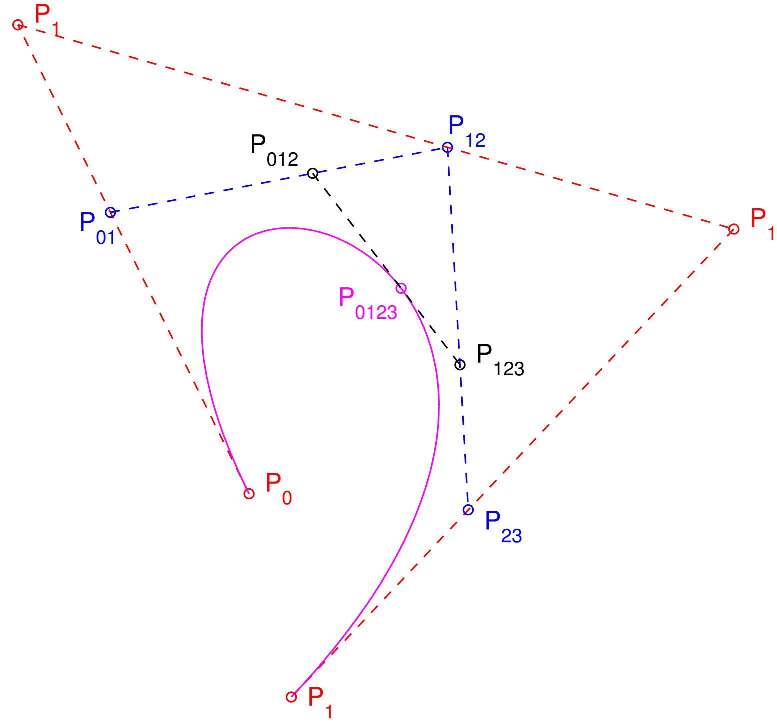


Figure 2.7: *Example of Bézier curve computed with de Casteljau's method. The curve is composed by the set of points produced by the procedure for all values of $t \in [0, 1]$.*

Bézier curves can also be computed via explicit formulae. For given control points P_i, $i = 0, ..., n$, the Bézier curve of order n is defined as

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \qquad i = 0, ..., n$$

is the binomial coefficient. The polynomials

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i, \quad i = 0, 1, ..., n$$

are called Bernstein polynomials and form a basis of the linear space of polynomials of degree at most n. Hence, in the case of $n = 3$ (cubic curve) the formula becomes

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

and it is easily verified that

$$B(0) = P_0 \quad \text{and} \quad B(1) = P_3.$$

2.4 Geodesics

In the Euclidean space, distance between two points is measured by the length of the linear segment connecting them. Geodesics are the generalization of this concept to curved spaces. In our case, we work on the sphere, hence geodesic curves arise naturally in the formulation of the problem at hand. On the sphere, the geodesic is defined as the shortest route between two points on the surface, which is equivalent to saying that the geodesic curve on the sphere is a segment of a *great circle*. A great circle is defined as the intersection of the sphere with a plane passing through its center. It divides the sphere into two equal hemispheres and the diameter of any great circle coincides with the diameter of the sphere. The intersection of the sphere with any other plane (not passing through its center) produces a so-called *minor circle*. An example is given in the figure below.

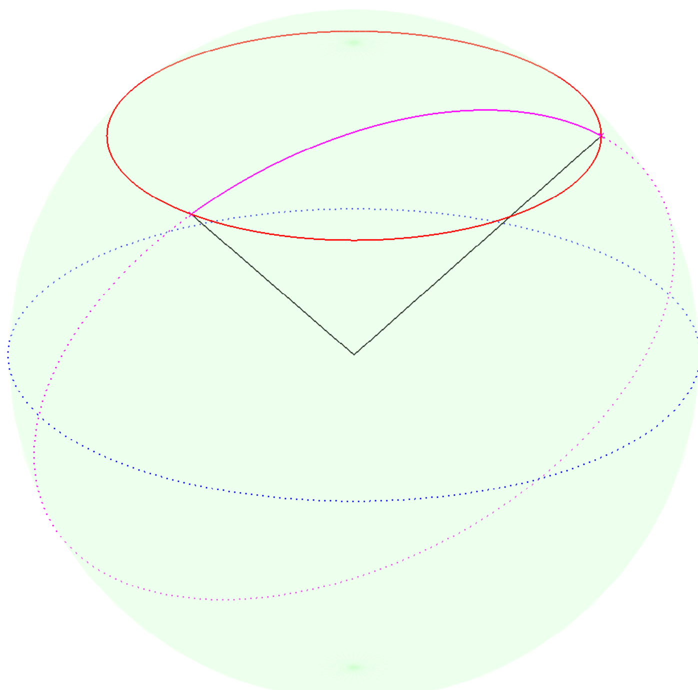


Figure 2.8: *Example of Great (blue and purple) and Minor (red) circles on the sphere. The length of the red segment is greater than the length of the purple segment on the sphere.*

Based on what we have already said, the minimum distance between two points on the sphere is equal to the length of the geodesic arc between them. By definition, the geodesic between any two points on the sphere has length at most π. However, in the case when two points are diametrical, i.e., their geodesic distance is exactly π, special care should be exercised as the geodesic is not uniquely defined – there are, in fact, infinitely many geodesics in this case. The geodesic distance between two points p and q on the unit sphere is defined as

$$\theta(p,q) = \cos^{-1}\left(\langle p,q\rangle\right).$$

Note that since p and q are on the unit sphere, it holds that $\|p\| = \|q\| = 1$.

Since the geodesic is the generalization of the linear segment connecting two points on the sphere, it is natural that linear interpolation is also generalized in the form of geodesic interpolation on the sphere, as we will see in section 3.1.

The geodesic segment between p and q can be parameterized as

$$G[p,q](t) = \frac{\sin((1-t)\cdot\theta(p,q))}{\sin(\theta(p,q))}\cdot p + \frac{\sin(t\cdot\theta(p,q))}{\sin(\theta(p,q))}\cdot q, \quad t \in [0,1].$$

As in the linear case, a simple substitution yields $G[p,q](0) = p$ and $G[p,q](1) = q$. It can be proved that, since $\|p\| = \|q\| = 1$, it also holds that $\|G[p,q](t)\| = 1,\ \forall\ t \in [0,1]$, hence all points described by G indeed lie on the unit sphere.

2.5 C^n and G^n continuity

Our problem, in its essence, is to find a piecewise continuous curve which fits certain criteria, so we need to explain what is continuity. We will consider two different notions of continuity.

C^n continuity. This is the so-called *parametric continuity*, since it is expressed in terms of the parameter used to describe the curve. Consider the curve $S(t),\ t \in \mathcal{D} \subseteq \mathbb{R}$. Continuity of order $n \in \mathbb{N}$ means that *all* derivatives of the curve with respect to the parameter t, up to order n, are continuous. In other words,

$$\lim_{t\to t_*^-}\frac{d^k}{dt^k}S(t) = \lim_{t\to t_*^+}\frac{d^k}{dt^k}S(t), \quad k = 0,1,...,n, \quad t_* \in \mathcal{D}.$$

G^n continuity. This is the so-called *geometric continuity* and expresses the smoothness of the curve in terms of geometric quantities. For instance, continuity of order G^1 expresses continuity of the slope, while G^2 continuity expresses continuity of the curvature.

Now, the question that naturally arises is, what is the difference between these two definitions? Suppose the curve $S(t)$ describes the motion of a rigid body in space, such as the remote - controlled toy car we used in our example in section 2.1. Then its first derivative with respect to t describes the velocity of the body, while the second derivative with respect to t describes the body's acceleration. Requiring that the curve $S(t)$ is C^2-continuous translates into requiring a smooth motion of the body, i.e., if we made a movie of the moving body, we would not see it change its position, velocity or acceleration in an abrupt manner. Requiring that the curve is G^2-continuous, on the other hand, translates into requiring that the trace the body leaves during its movement is smooth up to the quantities involved, i.e., the trace, its slope and curvature would not change abruptly. In this sense, we can say that G-continuity requirements are weaker that C-continuity requirements. An equivalent statement is that C-continuity also entails G-continuity, but the inverse is not true.

2.6 Shape-preservation

The notion of shape-preservation is a flexible one, in the sense that one may define different types of shape-preservation criteria relative to the requirements of a concrete application. It is also possible to express the same concept with different quantities. Our definition is based upon the definition presented in [1], which we will include here. In [1], the problem is set up in $\mathbb{R}^3$, thus the following formulation regards curves in three-dimensional Cartesian space. We will see in section 3.3 that the formulation of shape-preservation needs to change to account for the space in which we work. Keeping that in mind, let us define the quantities

$$
\begin{aligned}
\mathcal{D} &= \left\{P_m, m = 1, 2, ..., N\right\}, \\
L_m &= P_{m+1} - P_m, m = 1, 2, ..., N-1, \\
V_m &= L_{m-1} \times L_m, m = 2, 3, ..., N-1, \\
\Gamma_m &= |L_{m-1}\ L_m\ L_{m+1}| = \det\left[L_{m-1}\ L_m\ L_{m+1}\right].
\end{aligned}
$$

Then the problem treated in [1] is stated as follows.

Problem ($\boldsymbol{\mathcal{P}}$). *Find a G^2-continuous curve $S(u), u \in [u_1, u_N]$ which interpolates the point set $\mathcal{D}$ with parameterization $\mathcal{U}$, satisfies given boundary conditions $\mathcal{B}$ and is* shape-preserving *in the following sense:*

1. *(convexity) If $V_m \cdot V_{m+1} > 0$ then*

$$w(u) \cdot V_n > 0, \quad n = m, m+1, \quad w(u) = \dot{S}(u) \times \ddot{S}(u), \quad u \in \left[u_m, u_{m+1}\right].$$

2. *(torsion) If $\Gamma_m \neq 0$, then*

$$\tau(u)\Gamma_m > 0, u \in \left[u_m^+, u_{m+1}^-\right].$$

3. *(coplanarity) If $\Gamma_n = 0$ and*

   - *$V_m \cdot V_{m+1} > 0$ then, for $n = m$ and/or $n = m+1$*

$$\frac{\|w(u) \times V_n\|}{\|w(u)\|\,\|V_n\|} < \varepsilon_1, \quad \|w(u)\| \neq 0, \quad u \in \omega_m$$

   where ε_1 and ω_m are user-defined such that $\varepsilon_1 \in (0, 1]$ and $[u_m, u_{m+1}] \subseteq \omega_m \subset (u_{m-1}, u_{m+2})$.

- $V_m \cdot V_{m+1} < 0$ *then, for* $n = m$ *and/or* $n = m + 1$

$$\frac{\|w(u) \times V_n\|}{\|w(u)\| \, \|V_n\|} < \varepsilon_1, \quad \|w(u)\| \neq 0, \quad u \in \vartheta_m \cup \varphi_m$$

where ε_1 *is as above and* $\vartheta_m = [\vartheta_{m1}, \vartheta_{m2}]$ *and* $\varphi_m = [\varphi_{m1}, \varphi_{m2}]$ *are user-defined intervals such that* $\vartheta_{m1} \leq u_m < \vartheta_{m2} < u_m^*$ *and* $u_m^* < \varphi_{m1} < u_{m+1} \leq \varphi_{m2}$ *for some user-specified point* $u_m^* \in (u_m, u_{m+1})$.

4. *(collinearity) If* $\|V_m\| = 0$ *and* $L_{m-1} \cdot L_m > 0$*, then for* $n = m - 1, m$

$$\frac{\left\|\dot{S}(u) \times L_n\right\|}{\left\|\dot{S}(u)\right\| \left\|L_n\right\|} < \varepsilon_0, \quad u \in \eta_m$$

where $\varepsilon_0 \in (0, 1]$ *is user-specified and also* η_m *is a user-specified closed subinterval of* (u_{m-1}, u_{m+1}) *containing* u_m.

These requirements formulate a well-defined manner of saying that we want the resulting curve to follow the inherent *orientation* of the points in the given set. The first criterion (convexity) requires that all points of the resulting curve are on the same side of the hyperplane defined by a subset of the points, under certain conditions. The torsion criterion is tied to sharp variations in the orientation of a body traveling along the curve – it would be unnatural for a plane to instantly do a barrel roll, for example. The coplanarity criterion translates into requiring that whenever four consecutive points lie on the same plane (i.e., they are not affinely independent), the curve should also be relatively "close" to the plane (the user defines how close). Finally, the collinearity criterion asks that whenever three consecutive points lie on the same line, the curve should locally align with this line.

It should be noted that the coplanarity and torsion criteria only make sense when we are working in three-dimensional space, as is the case in [1]. In the current work we are working on the unit sphere which is a two-dimensional manifold in three-dimensional space. We choose to work in a coordinate system which is innate to the sphere, namely the Spherical Equatorial reference system. In this reference system we only need two quantities (longitude and latitude) to fully determine the location of a point on the surface of the sphere.

In section 3.3 we redefine the collinearity and convexity criteria in order to best address the requirements of shape-preservation on the sphere.

2.7 Discrete derivatives

In the process of verifying the shape-preserving conditions presented in section 3.3, the need to compute the value of some derivative of the spline arises. Since we do not have an explicit form for the curve in the general case, we employ finite difference methods to estimate its derivatives. The technique is a well-known and established one, and an automatic way of constructing a finite difference scheme for approximating derivatives of arbitrary order with arbitrary accuracy has been employed. The formulation of the method in its general form, as well as the following rationale, are explained in detail in [10].

Given a univariate function $f(t) : \mathbb{R} \to \mathbb{R}^n$ and a small value $\mathbb{R} \ni h > 0$, we can select a desired order of error p and require that the Taylor series expansion of f satisfy the following relation

$$\frac{h^d}{d!} f^{(d)}(t) = \sum_{i=i_{\min}}^{i_{\max}} C_i f(t + ih) + \mathcal{O}\left(h^{d+p}\right) \tag{2.1}$$

for some indices $i_{\min} \leq i \leq i_{\max},\ i_{\min}, i, i_{\max} \in \mathbb{Z}$, and coefficients $C_i \in \mathbb{R}$. Excluding the term $\mathcal{O}\left(h^{d+p}\right)$ transforms the above relation into an approximation for the derivative $f^{(p)}$. We are interested in formulating a way to compute the coefficients C_i in order to obtain an approximation of desired order.

A formal Taylor series for $f(t + ih)$ is

$$f(x + ih) = \sum_{n=0}^{\infty} i^n \frac{h^n}{n!} f^{(n)}(t) \tag{2.2}$$

and by substituting (2.2) into (2.1) we get

$$\begin{aligned}
\frac{h^d}{d!} f^{(d)}(t) &= \sum_{i=i_{\min}}^{i_{\max}} C_i \sum_{n=0}^{\infty} i^n \frac{h^n}{n!} f^{(n)}(t) + \mathcal{O}\left(h^{d+p}\right) \\
&= \sum_{n=0}^{\infty} \left(\sum_{i=i_{\min}}^{i_{\max}} i^n C_i \right) \frac{h^n}{n!} f^{(n)}(t) + \mathcal{O}\left(h^{d+p}\right) \\
&= \sum_{n=0}^{d+p-1} \left(\sum_{i=i_{\min}}^{i_{\max}} i^n C_i \right) \frac{h^n}{n!} f^{(n)}(t) + \mathcal{O}\left(h^{d+p}\right)
\end{aligned}$$

hence the desired approximation is

$$f^{(d)}(t) = \frac{d!}{h^d} \sum_{n=0}^{d+p-1} \left(\sum_{i=i_{\min}}^{i_{\max}} i^n C_i \right) \frac{h^n}{n!} f^{(n)}(t) + \mathcal{O}\left(h^p\right). \tag{2.3}$$

In order to satisfy (2.1), the following must hold

$$\sum_{i=i_{\min}}^{i_{\max}} i^n C_i = \begin{cases} 0, & 0 \leq n \leq d + p - 1 \text{ and } n \neq d, \\ 1, & n = d. \end{cases}$$

The above constraint yields a system of $d+p$ linear equations in $i_{\max}-i_{\min}+1$ unknowns. Requiring also that $i_{\max}-i_{\min}+1=d+p$ causes the system to have a unique solution, which gives us the coefficients needed to compute an approximation of $f^{(d)}(t)$ from (2.1). We can also select the modality of the approximation scheme (forward, backward or centered) by selecting the values of $i_{\min}$ and $i_{\max}$ as shown in the following reference table.

Modality	$\boldsymbol{i_{\min}}$	$\boldsymbol{i_{\max}}$
Forward	0	$d+p-1$
Backward	$-(d+p-1)$	0
Centered	$-\left\lfloor \frac{d+p-1}{2} \right\rfloor$	$\left\lfloor \frac{d+p-1}{2} \right\rfloor$

For instance, suppose we want to approximate $f^{(3)}(t)$ using forward finite differences and we need accuracy of order $\mathcal{O}(h)$. This means that $d=3, p=1, i_{\min}=0$ and $i_{\max}=3$, so the linear system we need to solve is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 4 & 9 \\ 0 & 1 & 8 & 27 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The solution of the system is $[C_0, C_1, C_2, C_3]^T = \frac{1}{6}\left[-1, 3, -3, 1\right]^T$, so the approximation resulting from (2.1) is

$$f^{(3)}(t) = \frac{-f(t)+3f(t+h)-3f(t+2h)-f(t+3h)}{h^3} + \mathcal{O}(h).$$

For the needs of the present work, approximations of the first, second and third derivative are needed. We have used fourth-order schemes in each case, either forward, backward or centered, depending on the value of t for which we wish to calculate the derivative. Requiring a fourth-order approximation of each derivative results in different needs for each derivative. In order to calculate the first derivative of f with fourth-order accuracy, we need four values of f. For the second derivative we need five values, and for the third derivative six values, in order to achieve fourth-order accuracy. All the values we calculate for f, however, should be in the same domain, i.e., in the same segment $[P_\kappa, P_{\kappa+1}],\ \kappa = 0, 1, ..., n-1$. Thus, for calculating the derivative at $t=0$, we would select a forward scheme, so that the values of f would be calculated at $t_j = jh,\ j = 0, 1, ..., m$ where $m=4$ for the first derivative, $m=5$ for the second derivative and $m=6$ for the third derivative. The same choice is made if $0 > t_m = mh$, obviously. Similarly, in the case where $t=1$ or, more generally, when $1 < t_m = mh$, we choose a backward scheme in order to calculate values for t_j less than 1. If t is anywhere in-between 0 and 1 (and as long as all indices are within the domain of interest), we prefer to use a centered scheme for all derivatives.

Chapter 3

Spherical splines

In this chapter we present the formulation of our problem and its solution. We present the theory behind interpolating points on the unit sphere with spherical cubic splines as described in [2], and give our own definition for shape-preservation constraints based on the concepts in [1]. We study the asymptotic behavior of the spline produced by the procedure as the tension values tend to infinity, and we also prove that the procedure eventually provides a solution which satisfies all criteria. Concluding, the method itself is presented, along with results arising from selected test cases.

3.1 Interpolation on the sphere

As we already mentioned in section 2.4, given q_κ, q_λ two points on the unit sphere, their geodesic distance is defined as

$$\theta(q_\kappa, q_\lambda) = \cos^{-1}(\langle q_\kappa, q_\lambda \rangle)$$

and the geodesic that interpolates them can be parameterized as

$$G[q_\kappa, q_\lambda](t) = \frac{\sin((1-t)\cdot\theta(q_\kappa, q_\lambda))}{\sin(\theta(q_\kappa, q_\lambda))} q_\kappa + \frac{\sin(t\cdot\theta(q_\kappa, q_\lambda))}{\sin(\theta(q_\kappa, q_\lambda))} q_\lambda.$$

Using this definition of the geodesic, we are able to define a *spherical Bézier curve* by recursively applying the geodesic interpolation, following the idea of de Casteljau. For a set of $(n+1)$ points on the sphere $q_i, i = 0, 1, ..., n$ we define the spherical Bézier curve as

$$q_n^n(t) = S[q_0, q_1, q_2, ..., q_n](t), \quad 0 \le t \le 1$$

where

$$q_i^k(t) = G[q_{i-1}^{k-1}(t), q_i^{k-1}(t)](t), \quad k \le i \le n$$

and $q_i^0(t) = q_i, i = 0, 1, ..., n$. It can be shown that $q_n^n(0) = q_0$ and $q_n^n(1) = q_n$.

A cubic spherical spline results from four control points, as in the example case presented in section 2.3. An example on the sphere is given in the figure below.

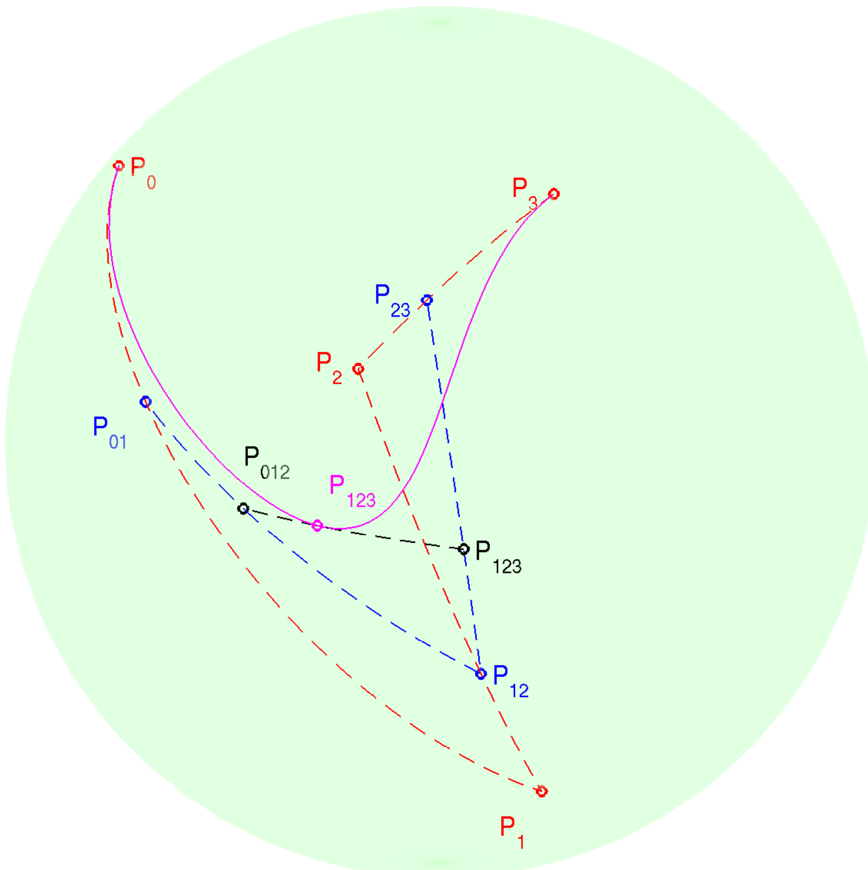


Figure 3.1: *Example of cubic Bézier curve on the sphere. The red dotted curves represent geodesics between the input control points, while the blue and black dotted curves approximations of first and second order, respectively. The purple solid curve is the resulting Bézier curve. The points have been numbered in correspondence with the example given in the planar case.*

3.2 Spherical ν-splines

The theory presented in this section has been introduced by Nielson in [2]. Based on our previous discussion, we define the spherical ν-spline. The notation introduced at this point will be retained for the rest of the text.

Definition. *Given*

- *control points* $d_i,\ i = 0, 1, ..., n$

- *knots* $t_i,\ i=0,1,...,n$ *with knot spacing* $h_i=t_i-t_{i-1},\ i=1,2,...,n$ *and* $h_0=h_{n+1}=0$
- *tension values* $\nu_i,\ i=0,1,...,n$

the third-order ν*-spline is defined as the composite curve consisting of* n *curve segments and is denoted as*

$$\nu(d_j,t_j,\nu_j)(t).$$

Each segment is a third-order spherical Bézier curve as previously defined in the form

$$S[P_{i-1},R_{i-1},L_i,P_i]\left(\frac{t-t_{i-1}}{h_i}\right)=S(t)$$

where $i=1,...,n$ *and* $t_{i-1}\le t\le t_i$.

In the above definition, the missing quantities are defined as

$$\begin{aligned}
L_i &= G\left[d_{i-1},d_i\right]\left(\frac{\gamma_{i-1}h_{i-1}+h_i}{\gamma_{i-1}h_{i-1}+h_i+\gamma_i h_{i+1}}\right), \quad i=1,2,...,n,\\
R_i &= G\left[d_i,d_{i+1}\right]\left(\frac{\gamma_i h_i}{\gamma_i h_i+h_{i+1}+\gamma_{i+1}h_{i+2}}\right), \quad i=0,1,...,n-1,\\
P_i &= G\left[L_i,R_i\right]\left(\frac{h_i}{h_i+h_{i+1}}\right), \quad i=0,1,...,n,\\
\gamma_i &= \frac{2(h_i+h_{i+1})}{\nu_i h_i h_{i+1}+2(h_i+h_{i+1})}, \quad i=0,1,...,n.
\end{aligned}$$

It is clear that the spherical ν-spline is defined by the global control points d_i. In our formulation, these quantities are unknown and we seek to compute them. We are instead given the points $P_i,\ i=0,1,...,n$ which lie on the spline, and the auxiliary quantities L_i and R_i defined as above. Each quadruple $\{P_i,R_i,L_{i+1},P_{i+1}\},\ i=0,1,...,n-1$ defines a third-order Bézier curve which interpolates both P_i and P_{i+1}, thus the resulting spline interpolates all input points P_i, and is also a composite cubic Bézier curve. The concept is illustrated in the figure below.

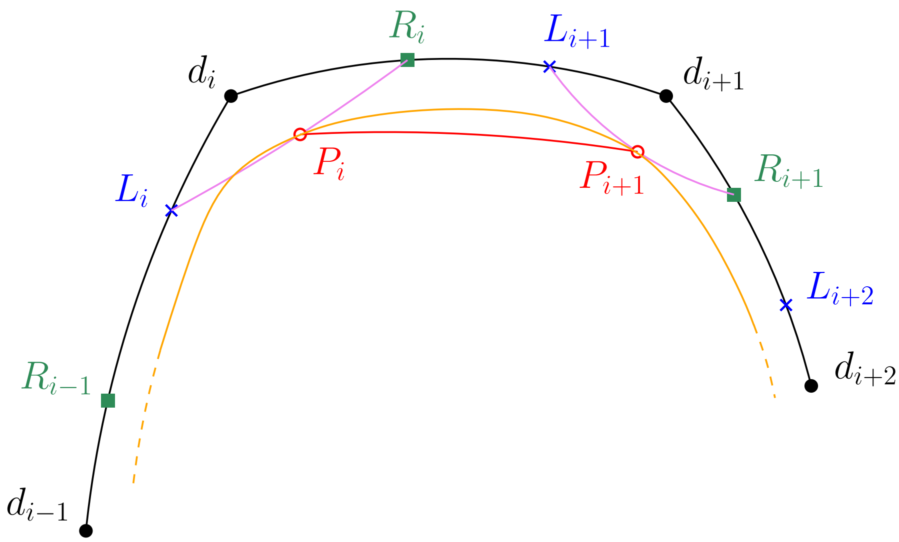


Figure 3.2: *Representation of the de Casteljau algorithm on the sphere, illustrating the construction of a piecewise cubic ν-spline. In the forward formulation, we are given the control points d_i and compute the points P_i which lie on the curve. In the inverse formulation, we are given the points P_i and seek the control points d_i which will cause the spline to interpolate.*

A solution to this problem has been given in [2] and is summarized in the following formulation.

Definition. *Consider the non-linear system of equations*

$$
\begin{aligned}
d_0 &= P_0 \\
d_i &= \frac{P_i \sin(\beta_i) - \frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)} d_{i-1} - \frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})} d_{i+1}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} \\
d_n &= P_n
\end{aligned}
$$

where

$$
\begin{gathered}
\alpha_i = \cos^{-1}(d_{i-1}, d_i), \quad \alpha_{i+1} = \cos^{-1}(d_i, d_{i+1}) \\
L_i = G[d_{i-1}, d_i](\lambda_i), \quad R_i = G[d_i, d_{i+1}](\mu_i) \\
\beta_i = \cos^{-1}(L_i, R_i) \\
\lambda_i = \frac{\gamma_{i-1}h_{i-1} + h_i}{\gamma_{i-1}h_{i-1} + h_i + \gamma_i h_{i+1}}, \quad \mu_i = \frac{\gamma_i h_i}{\gamma_i h_i + h_{i+1} + \gamma_{i+1}h_{i+2}} \\
\delta_i = \frac{h_i}{h_i + h_{i+1}}, \quad \gamma_i = \frac{2(h_i + h_{i+1})}{\nu_i h_i h_{i+1} + 2(h_i + h_{i+1})}
\end{gathered}
$$

for $i = 0, 1, ..., n$. Compute the control points $d_i,\ i = 0, 1, ..., n$ so that

$$\nu(d_j, t_j, \nu_j)(t_i) = P_i, \quad i = 0, 1, ..., n$$

with the following algorithm:

1. *Select initial approximation for the quantities* $d_0^{(0)}, d_1^{(0)}, ..., d_n^{(0)}$*. Setting* $d_i^{(0)} = P_i, \ i = 0, 1, ..., n$ *is as good a choice as any.*

2. *Compute updated values for the quantities* α_i, β_i *based on current values of the* d_i*.*

3. *Compute new approximation for the control points* d_i *in the form*

$$d_i^{(k)} = F\left(d_{i-1}^{(k-1)}, d_i^{(k-1)}, d_{i+1}^{(k-1)}\right), \quad i = 1, 2, ..., n-1.$$

4. *If convergence has been achieved, exit. If not, return to step 2.*

Convergence in the above algorithm is considered achieved if the number of iterations exceeds a user-defined threshold $S_{N_{\max}}$*, or if for a user defined tolerance parameter* S_{tol} *it holds that*

$$\sqrt{\frac{\sum_{i=0}^{n}\left[d_i^{(k)} - d_i^{(k-1)}\right]^2}{\sum_{i=0}^{n}\left[d_i^{(k)}\right]^2}} \leq S_{tol}.$$

The spherical curve resulting from the above procedure interpolates the input points P_i and is guaranteed to be G^2-continuous at the interpolation nodes. In the special case where all tension values are 0, the resulting curve is actually C^2-continuous. However, in [2] the argument of shape-preservation is not treated.

It should be noted that the choice for the knot values t_i plays a significant role in the quality of the resulting spline. One naive choice would be to select a uniformly spaced set of knot values, for instance $t_0 = 0, \ t_i = t_{i-1} + h, \ i = 1, 2, ..., n$ for some positive parameter h. This choice, however, may lead to curves which behave unnaturally between the interpolation points. For instance, if the distance between consecutive input points P_i is not uniform, the resulting spline may present loops. We therefore adopt an *arc-length parameterization*, meaning that knot values are placed at a distance corresponding to the geodesic distance between consecutive input interpolation points P_i. In other words,

$$\begin{aligned} &t_0 = 0, \\ &t_i = t_{i-1} + \cos^{-1}\left(\langle P_{i-1}, P_i\rangle\right), \quad i = 1, 2, ..., n. \end{aligned}$$

This, in turn, means that the previously introduced quantities h_i express, in fact, the geodesic distances between consecutive interpolation points P_{i-1} and P_i.

The presence of the free parameters ν_i in the formulation of the algorithm is crucial for the application of this method to tackling the requirements of shape-preservation. Other than giving the resulting curve its distinctive name, it has a major effect on its shape. In subsection 3.4.1 and subsection 3.4.2 we will examine the behavior of the spline when the tension values tend to infinity.

3.3 Shape-preservation on the sphere

In this chapter we proceed to formulate shape-preservation criteria on the sphere. In section 2.6, we encountered the definition introduced by Karavelas & Kaklis in [1]. However, the formulation in section 2.6 refers to a Cartesian 3D setting in which the quantities introduced make sense. We need to take into account the fact that our work is conducted on the unit sphere, which is a curved 2D surface. The ideas expressed here and the formulation of the shape-preservation criteria are based upon [3].

Initially, let us consider the set of given points on the unit sphere $\mathcal{P} = \{P_0, P_1, ..., P_n\}$ such that $P_i \neq P_{i+1},\ i = 0, 1, ..., n-1$ and $\max(\theta(P_i, P_{i+1})) < \pi/2, i = 0, 1, ..., n-1$ with

$$\theta(P_i, P_{i+1}) = \cos^{-1}\left(\langle P_i, P_{i+1}\rangle\right), \quad \|P_i\| = \|P_{i+1}\| = 1.$$

Also, consider given the set of knot values t_i such that for the interpolating spline $S(t),\ t \in [0, t_n]$ we have $S(t_i) = P_i$.

In [3], Kaklis offers a generalization of shape preservation on curved surfaces, based on *geodesic curvature*. The geodesic curvature of a curve $c(t)$ (which is a subset of a curved surface) is defined as

$$\kappa_g(t; c) = \frac{\mathbf{n}(t) \cdot [\dot{c}(t) \times \ddot{c}(t)]}{\|\dot{c}(t)\|^3},$$

where $\mathbf{n}(t)$ is the normal vector of the surface at t. This quantity expresses (quote) *the curvature of the projection of $c(t)$ on the plane tangent to the surface at the point under consideration. Curves for which $\kappa_g = 0$ are geodesic curves.*

Following the idea in section 2.6, we want to generalize the notion of shape preservation. In particular, regarding convexity criteria, the idea is to exploit the pattern of the sign changes of the so-called *convexity indicators*. In section 2.6, these are the vector-valued quantities V_i which, in our case, need to be defined differently. Considering only the numerator in the definition for the geodesic curvature, we can see that at an interpolation node P_i, corresponding to a knot parameter t_i, the following limit is valid.

$$\begin{aligned}
\mathbf{n}(t_i) \cdot \left[\dot{c}(t_i) \times \ddot{c}(t_i)\right] &= \lim_{h \to 0} \mathbf{n}(t_i) \cdot \left[\dot{c}(t_i - h) \times \frac{\dot{c}(t_i + h) - \dot{c}(t_i - h)}{2h}\right] \\
&= \lim_{h \to 0} \frac{1}{2h} \mathbf{n}(t_i) \cdot \left[\dot{c}(t_i - h) \times \dot{c}(t_i + h)\right] \\
&- \lim_{h \to 0} \frac{1}{2h} \mathbf{n}(t_i) \cdot \left[\dot{c}(t_i - h) \times \dot{c}(t_i - h)\right] \\
&= \lim_{h \to 0} \frac{1}{2h} \mathbf{n}(t_i) \cdot \left[\dot{c}(t_i - h) \times \dot{c}(t_i + h)\right].
\end{aligned} \tag{3.1}$$

This means that for sufficiently small h the sign of the geodesic curvature is defined by the sign of the quantity $\mathbf{n}(t_i) \cdot [\dot{c}(t_i - h) \times \dot{c}(t_i + h)]$. We need to notice here that in our case, the input points P_i are essentially the normal vectors to the surface of the sphere. We can thus replace $\mathbf{n}(t_i) = P_i$ and, by substituting $c(t)$ with the spline curve $S(t)$, we define the convexity indicators on the sphere as

$$Q_i = \frac{P_i \cdot [G[P_{i-1}, P_i](1) \times G[P_i, P_{i+1}](0)]}{\|G[P_{i-1}, P_i](1) \times G[P_i, P_{i+1}](0)\|}.$$

Introducing a slightly better notation, let us define the following quantities summarizing the tools we need.

$$\begin{aligned}
\Gamma_i(t) &= G[P_i, P_{i+1}](t), \quad i = 0, 1, ..., n-1, \quad t \in [0, 1], \\
V_i &= \dot{\Gamma}_{i-1}(1) \times \dot{\Gamma}_i(0), \quad i = 1, 2, ..., n-1, \\
\kappa_g(t; S) &= \frac{S(t) \cdot \left[\dot{S}(t) \times \ddot{S}(t)\right]}{\left\|\dot{S}(t)\right\|^3}, \quad t \in [0, t_n] \\
Q_i &= \frac{P_i \cdot V_i}{\|V_i\|}, \quad i = 1, 2, ..., n-1.
\end{aligned}$$

The quantity V_i expresses a *binormal* vector to the spline, defined by the geodesics on both sides of the internal nodes. If the geodesic Γ_{i-1} does not coincide with the geodesic Γ_i, then this vector is non-zero and is perpendicular to the surface of the sphere. It can either point inwards or outwards, depending on the relative position of the input points. The orientation of V_i is easily found by the right-hand rule. The quantity Q_i, on the other hand, is a scalar whose value can be either 1, 0 or -1 and provides a definition of the local orientation of the poly-geodesic curve at each node P_i. In the same manner, the quantity $\kappa_g(t; S)$ is a scalar quantity in which the term $S(t)$ is essentially the normal vector of the curve $S(t)$ at each $t \in [0, t_n]$. Where appropriate, we will use a localized notation for the geodesic curvature defined as

$$\kappa_g(t; S_i) = \frac{S_i(t) \cdot \left[\dot{S}_i(t) \times \ddot{S}_i(t)\right]}{\left\|\dot{S}_i(t)\right\|^3}, \quad S_i \in [P_i, P_{i+1}], \quad t \in [0, 1]$$

With these quantities in mind, the problem we treat can be expressed in the following way.

Problem ($\boldsymbol{\mathcal{S}}$). *Find a G^2-continuous curve $S(u)$ that interpolates the point set $\mathcal{P}$ with parameterization $\mathcal{U}$ and is shape-preserving in the sense:*

- *(Co-circularity) If $Q_i = 0$ and $\theta(P_{i-1}, P_{i+1}) > \max\{\theta(P_{i-1}, P_i), \theta(P_i, P_{i+1})\}$, $i = 1, 2, ..., n-1$, then*

  $$|\kappa_g(t_i\,; S)| < \varepsilon, \quad i = 1, 2, ..., n-1,$$

  where $\varepsilon \in (0, 1]$ is a user-defined variable.

- *(Nodal convexity) If $Q_i \neq 0,\ i = 1, 2, ..., n-1$, then the following must be true*

$$\kappa_g(t_i\,; S)\, Q_i \geq 0.$$

- *(Segment convexity) If $Q_i\, Q_{i+1} > 0$, then it must be true that*

$$\kappa_g(t; S_i)\, Q_\lambda > 0, \quad t \in [0,1], \quad \lambda \in \{i, i+1\}\,, \quad i = 1, 2, ..., n-2.$$

The first difference we notice in our formulation from the one we saw in section 2.6 is that we do not treat coplanarity and torsion. As we said, in [1] Karavelas & Kaklis are working in a 3D Cartesian space, where coplanarity is not guaranteed, and torsion also makes sense. In our case, we are working on the sphere, hence coplanarity with a more general meaning is implied, since all points examined lie on the surface of the unit sphere, and torsion need not be treated in this context. Let us now take a moment to appraise the meaning of the above requirements.

The co-circularity criterion states that whenever a triplet P_{i-1}, P_i, P_{i+1} lies on the same great circle arc, the resulting spline should also follow this orientation at the interpolation point P_i within some tolerance implied by ε. This is verified by requiring that the geodesic curvature of the spline at the interpolation point is smaller than a threshold ε, which is the case when the curve locally aligns with the geodesic – when κ_g becomes zero, the spline actually coincides with the geodesic. Now, this criterion only makes sense when the points are in the correct *order*. Co-circularity has to be examined only when P_{i+1} comes *after* P_i on the geodesic segment $P_{i-1} \to P_i \to P_{i+1}$. If P_{i+1} is between P_{i-1} and P_i, it makes no sense to examine co-circularity. These degenerate cases are covered by controlling that the geodesic distance between P_{i-1} and P_{i+1} must exceed both the geodesic distances between P_i, P_{i+1} and P_{i-1}, P_i. This is a suitable control in the case where the maximum distance between the input points is less than $\pi/2$. An example illustrating the motivation for this requirement is given in the following figure.

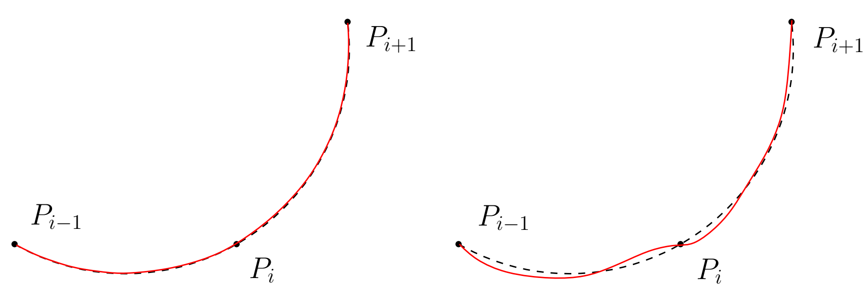


Figure 3.3: *Illustration of the motivation behind the co-circularity requirement. On the left, the shape of the curve required by the criterion. On the right, one possible scenario where the curve does not obey the criterion. Note that the curve on the right interpolates the points and is G^2-continuous at the interpolation point, but its shape does not follow the natural orientation expected in a similar setup. The tangent vector at P_i has a significantly different orientation than the one implied by the geodesics. The black dashed curves represent the geodesics between the nodes.*

The convexity criterion, on the other hand, expresses the shape we would like to achieve either on an interpolation node or a segment of the spline. Convexity at an interpolation node is examined via the sign of the product of the geodesic curvature and the convexity indicator at each point. Nodal convexity is only requested at nodes which are not co-circular with their neighbors. The nodal convexity requires non-negative sign of the product because, as we will see in the next chapter, the geodesic curvature at the interpolation nodes tends to become 0. An illustration of the idea behind this requirement is shown below.

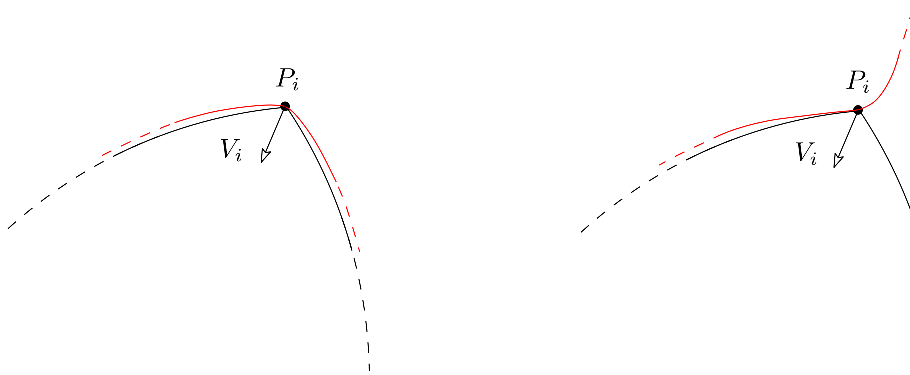


Figure 3.4: *Illustration of the motivation behind the node convexity requirement. On the left, the shape of the curve required by the criterion. On the right, one possible scenario where the curve does not obey the criterion. Note that the curve does not follow the natural orientation defined by the nodes on both sides of P_i. The black curves represent parts of the geodesics between the nodes.*

Segment convexity states a more strict requirement. We request that whenever four consecutive points on the sphere form a “convex” poly-geodesic setting, the curve must follow this orientation on the segment between the two middle points. A convex poly-geodesic setting is easily verified by controlling whether the sign of the quantities Q_i and Q_{i+1} is the same, indicating compatible alignment of the geodesics on both sides of the nodes P_i and P_{i+1}. Think again of the right-hand rule – a convex setting means that our thumb remains in the same half-space if we go all the way from P_{i-1} to P_{i+2}. If such a setting is verified, it is natural to require that the resulting curve also follows this orientation, and this requirement is easily expressed by using the geodesic curvature. An example of segment convexity is given below.

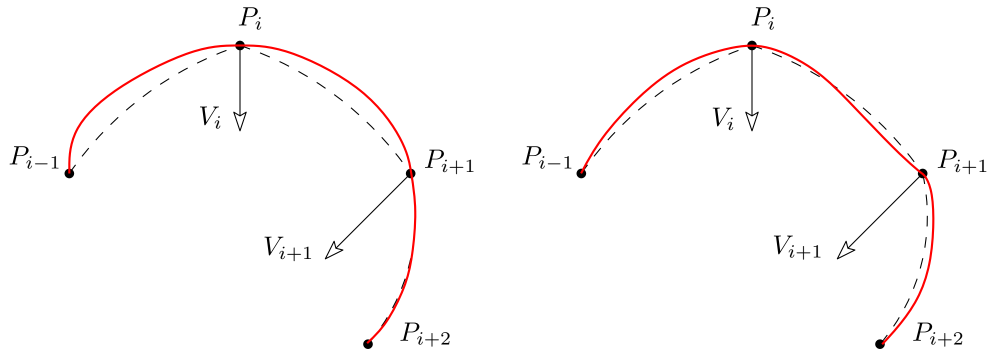


Figure 3.5: *Illustration of the motivation behind the segment convexity requirement. On the left, the shape of the curve required by the criterion. On the right, one possible scenario where the curve does not obey the criterion. Note that the curve does not follow the natural orientation defined by the interpolation nodes in the interval* $[P_i, P_{i+1}]$. *The black dashed curves represent the geodesics between the nodes.*

3.4 Qualitative asymptotic analysis

In this section we will examine the behavior of the spherical ν-spline as previously defined when the tension values tend to infinity. Our analysis is motivated by the ansatz that when $\nu_i \to \infty$ for suitable indices $i \in \{0, 1, ..., n\}$, the shape-preserving criteria established in the previous section are validated. For the rest of the section we will adopt the notation introduced in section 3.2.

3.4.1 Control point limits

Let us recall that the control points $d_i,\ i = 1, 2, ..., n-1$ are computed via the nonlinear expression

$$d_i = \frac{P_i \sin(\beta_i) - \frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)} d_{i-1} - \frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})} d_{i+1}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}, \quad i = 1, 2, ..., n-1$$

where

$$\alpha_i = \cos^{-1}(d_{i-1}, d_i), \quad \alpha_{i+1} = \cos^{-1}(d_i, d_{i+1})$$
$$L_i = G[d_{i-1}, d_i](\lambda_i), \quad R_i = G[d_i, d_{i+1}](\mu_i)$$
$$\beta_i = \cos^{-1}(L_i, R_i)$$
$$\lambda_i = \frac{\gamma_{i-1}h_{i-1} + h_i}{\gamma_{i-1}h_{i-1} + h_i + \gamma_i h_{i+1}}, \quad \mu_i = \frac{\gamma_i h_i}{\gamma_i h_i + h_{i+1} + \gamma_{i+1}h_{i+2}}$$
$$\delta_i = \frac{h_i}{h_i + h_{i+1}}, \quad \gamma_i = \frac{2(h_i + h_{i+1})}{\nu_i h_i h_{i+1} + 2(h_i + h_{i+1})}$$

With a little attention, we can see that the quantities $d_i,\ i = 1, 2, ..., n-1$ depend on the triplet of tension values $\{\nu_{i-1}, \nu_i, \nu_{i+1}\}$. With this remark in mind, our analysis is based on the following assumptions.

Assumption 3.1. *Assume that the following are true.*

1. *The limit*

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i, \qquad i = 1, 2, ..., n-1$$

exists.

2. *The limit*

$$\lim_{\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} d_{i+1}, \qquad i = 1, 2, ..., n-2$$

also exists.

3. *The limits for d_i and d_{i+1} are such that*

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \alpha_i \neq 0, \qquad i = 1, 2, ..., n-1.$$

We can now begin to examine the behavior of the quantities involved in the computation. First, note that the control points $d_i,\ i = 0, 1, ..., n$ all lie on the unit sphere. The validity of this observation is based on the fact that the nonlinear system of equations described in section 3.2 is the inverse version of de Casteljau's algorithm for points on the unit sphere. The auxiliary quantities $L_i,\ i = 1, 2, ..., n-1$ and $R_i,\ i = 0, 1, ..., n$ are computed via geodesics interpolation between every two consecutive control points, thus they are also situated on the unit sphere.

A second important observation is the fact that the quantities $\alpha_i,\ i = 1, 2, ..., n$ are non-zero and also bounded. Since the control points d_i lie on the unit sphere, and we have assumed that they do not coincide neither initially nor at their limit, it is true $\alpha_i < \pi$, hence we conclude that

$$0 < \alpha_i < \pi, \qquad i = 1, 2, ..., n. \tag{3.2}$$

Let us now examine the other quantities involved. It is evident that the quantity directly affected by the tension values is γ_i. Is is not difficult to verify that

$$\lim_{\nu_i\to\infty} \gamma_i = 0, \quad i = 0, 1, ..., n.$$

We can also see that when $\nu_i = 0$, then $\gamma_i = 1$, hence

$$0 \leq \gamma_i \leq 1, \quad i = 0, 1, ..., n.$$

Now, in order to obtain the limit for the quantities λ_i and μ_i, we will need to evaluate the limit for all tension values involved, hence

$$\lim_{\nu_i,\nu_{i+1}\to\infty} \mu_i = \lim_{\nu_i,\nu_{i+1}\to\infty} \frac{\gamma_i h_i}{\gamma_i h_i + h_{i+1} + \gamma_{i+1} h_{i+2}}.$$

The limit for the numerator is

$$\lim_{\nu_i\to\infty} \gamma_i h_i = 0$$

while for the denominator we have

$$\lim_{\nu_i,\nu_{i+1}\to\infty} \gamma_i h_i + h_{i+1} + \gamma_{i+1} h_{i+2} = h_{i+1} \neq 0$$

thus we conclude that

$$\lim_{\nu_i,\nu_{i+1}\to\infty} \mu_i = 0.$$

On the other hand, if $\nu_i = 0 = \nu_{i+1}$, we have seen that $\gamma_i = \gamma_{i+1} = 1$ which implies

$$\mu_i = \frac{h_i}{h_i + h_{i+1} + h_{i+2}} < 1$$

thus we have

$$0 \leq \mu_i < 1 \Longleftrightarrow 0 < 1 - \mu_i \leq 1, \quad i = 0, 1, ..., n-1. \tag{3.3}$$

A result that follows from (3.3), given (3.2) (which obviously holds for a_{i+1} as well), is that

$$0 \leq \sin(\mu_i \alpha_{i+1}) < 1 \quad \text{and} \quad 0 < \sin((1-\mu_i)\alpha_{i+1}) \leq 1, \quad i = 0, 1, ..., n-1. \tag{3.4}$$

For the quantity λ_i, the limit is

$$\lim_{\nu_{i-1},\nu_i\to\infty} \lambda_i = \lim_{\nu_{i-1},\nu_i\to\infty} \frac{\gamma_{i-1} h_{i-1} + h_i}{\gamma_{i-1} h_{i-1} + h_i + \gamma_i h_{i+1}}.$$

As in the previous case, for the numerator we have

$$\lim_{\nu_{i-1}\to\infty} \gamma_{i-1} h_{i-1} + h_i = h_i \neq 0$$

and for the denominator

$$\lim_{\nu_{i-1},\nu_i\to\infty} \gamma_{i-1} h_{i-1} + h_i + \gamma_i h_{i+1} = h_i \neq 0$$

hence

$$\lim_{\nu_{i-1},\nu_i\to\infty} \lambda_i = \frac{h_i}{h_i} = 1.$$

In this case we can again verify that for $\nu_{i-1} = 0 = \nu_i$ we have $\gamma_{i-1} = \gamma_i = 1$ which leads to

$$\lambda_i = \frac{h_{i-1} + h_i}{h_{i-1} + h_i + h_{i+1}} < 1$$

but positive, no less. Therefore for λ_i we have

$$0 < \lambda_i \leq 1 \Longleftrightarrow 0 \leq 1 - \lambda_i < 1, \quad i = 1, 2, ..., n. \tag{3.5}$$

Combining the results of (3.5) with (3.4), we conclude that

$$0 < \sin(\lambda_i \alpha_i) \leq 1 \quad \text{and} \quad 0 \leq \sin((1 - \lambda_i)\alpha_i) < 1, \quad i = 1, 2, ..., n. \tag{3.6}$$

It is evident that, by definition,

$$0 < \delta_i < 1 \Longleftrightarrow 0 < 1 - \delta_i < 1, \quad i = 0, 1, ..., n. \tag{3.7}$$

Now, given the conclusions we have reached so far, we can prove the following lemma.

Lemma 3.1. *For the quantities L_i and R_i, the limits for the corresponding tension values exist and are*

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} L_i = \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i, \quad i = 1, 2, ..., n - 1$$

and

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} R_i = \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i, \quad i = 1, 2, ..., n - 1.$$

Proof. Based on Assumption 3.1, the limits for the quantities d_i exist, and are such that the quantities α_i are non-zero. We do not know the limits for the control points d_i at this point, but we do know that they exist. With this knowledge, we can write

$$\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} L_i &= \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} G[d_{i-1}, d_i](\lambda_i) \\
&= G\left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_{i-1}, \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i\right]\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \lambda_i\right) \\
&= G\left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_{i-1}, \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i\right](1) \\
&= \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i.
\end{aligned}$$

Following the same logic,

$$\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} R_i &= \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} G[d_i, d_{i+1}](\mu_i) \\
&= G\left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i, \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_{i+1}\right]\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \mu_i\right) \\
&= G\left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i, \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_{i+1}\right](0) \\
&= \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i.
\end{aligned}$$

□

An immediate consequence of Lemma 3.1 is that

$$\begin{aligned}\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\beta_i &= \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\cos^{-1}\left(L_i,R_i\right)\\ &=\cos^{-1}\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}L_i,\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}R_i\right)\\ &=\cos^{-1}\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}d_i,\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}d_i\right)\\ &=0.\end{aligned}\tag{3.8}$$

Bearing these results in mind, we will examine the behavior of the control point d_i with respect to the corresponding input point P_i as the appropriate tension values tend to infinity. The indices i are in the range $i=1,2,...,n-1$. In order to proceed, we will first consider the difference

$$\begin{aligned}d_i-P_i &= \frac{P_i\sin(\beta_i)-\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}d_{i-1}-\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}d_{i+1}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}-P_i\\ &=\left[\frac{\sin(\beta_i)}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}-1\right]P_i\\ &\quad-\left[\frac{\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right]d_{i-1}\\ &\quad-\left[\frac{\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right]d_{i+1}.\end{aligned}$$

Let $\|\cdot\|$ denote the Euclidean norm in $\mathbb{R}^3$. We can thus write

$$\begin{aligned}\|d_i-P_i\| &\leq \left|\frac{\sin(\beta_i)}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}-1\right|\|P_i\|\\ &\quad+\left|\frac{\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right|\|d_{i-1}\|\\ &\quad+\left|\frac{\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right|\|d_{i+1}\|.\end{aligned}$$

However, given that P_i, d_{i-1} and d_{i+1} are points on the unit sphere, it holds that $\|P_i\|=$

$\|d_{i-1}\| = \|d_{i+1}\| = 1$. Therefore, the last expression becomes

$$
\begin{aligned}
\|d_i - P_i\| \leq & \left| \frac{\sin(\beta_i)}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} - 1 \right| \\
& + \left| \frac{\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} \right| \\
& + \left| \frac{\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} \right|.
\end{aligned}
\tag{3.9}
$$

We are interested in examining the behavior of the above quantities when the tension values $\{\nu_{i-1}, \nu_i, \nu_{i+1}\}$ tend to infinity, in other words we want to see what happens when we apply the limit to both sides of (3.9). Initially, we can see that

$$
\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \|d_i - P_i\| \leq & \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \left| \frac{\sin(\beta_i)}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} - 1 \right| \\
& + \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \left| \frac{\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} \right| \\
& + \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \left| \frac{\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} \right|,
\end{aligned}
\tag{3.10}
$$

however it becomes abundantly clear that we will have clearer results by handling each expression separately. For the first limit, we want to examine

$$
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \left| \frac{\sin(\beta_i)}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} - 1 \right|. \tag{3.11}
$$

Instead on examining the entire expression, we will focus on showing that the fraction's limit is the unit. This, however, is equivalent to showing that the inverse of the fraction has the unit as its limit. Hence, we now have

$$
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}{\sin(\beta_i)}. \tag{3.12}
$$

Recall that our assumptions state that

$$
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_i \neq 0,
$$

which allows us to see that

$$
\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} &= \frac{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \lambda_i \cdot \lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_i\right)}{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_i\right)} \\
&= \frac{\sin\left(1 \cdot \lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_i\right)}{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_i\right)} \\
&= 1,
\end{aligned}
$$

and also

$$
\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})} &= \frac{\sin\left(\left(1-\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \mu_i\right) \cdot \lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_{i+1}\right)}{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_{i+1}\right)} \\
&= \frac{\sin\left((1-0) \cdot \lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_{i+1}\right)}{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \alpha_{i+1}\right)} \\
&= 1.
\end{aligned}
$$

Now (3.12) can be rewritten as

$$
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\sin((1-\delta_i)\beta_i)+\sin(\delta_i\beta_i)}{\sin(\beta_i)} = \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\sin((1-\delta_i)\beta_i)}{\sin(\beta_i)} + \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\sin(\delta_i\beta_i)}{\sin(\beta_i)}. \tag{3.13}
$$

The quantity δ_i does not depend on the tension values ν_i, and we have seen that the limit for the quantity β_i is zero, hence by de l'Hôspital's rule we can easily verify that (3.13) becomes

$$
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \frac{\sin((1-\delta_i)\beta_i)+\sin(\delta_i\beta_i)}{\sin(\beta_i)} = (1-\delta_i)+\delta_i = 1. \tag{3.14}
$$

Returning to (3.11) with the result of (3.14), we conclude that

$$
\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} \left| \frac{\sin(\beta_i)}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)} + \frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}} - 1 \right| = |1-1| = 0. \tag{3.15}
$$

Let us now continue with the second limit in order. We have

$$\begin{aligned}
&\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\left|\frac{\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right| \leq \\
&\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\left|\frac{\frac{\sin((1-\delta_i)\beta_i)\sin((1-\lambda_i)\alpha_i)}{\sin(\alpha_i)}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}}\right| = \\
&\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\left|\frac{\sin((1-\lambda_i)\alpha_i)}{\sin(\lambda_i\alpha_i)}\right| = \left|\frac{\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\sin((1-\lambda_i)\alpha_i)}{\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\sin(\lambda_i\alpha_i)}\right| = \\
&\left|\frac{\sin\left(\left(1-\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\lambda_i\right)\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_i\right)}{\sin\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\lambda_i\cdot\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_i\right)}\right| = \\
&\left|\frac{\sin\left((1-1)\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_i\right)}{\sin\left(1\cdot\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_i\right)}\right| = \\
&\left|\frac{\sin(0)}{\sin\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_i\right)}\right| = \\
&\left|\frac{0}{\sin\left(\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_i\right)}\right| = 0.
\end{aligned} \tag{3.16}$$

The last result is correct under the Assumption (3.1).

Finally, let us examine the third limit, for which we have

$$
\begin{aligned}
&\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\left|\frac{\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}}{\frac{\sin((1-\delta_i)\beta_i)\sin(\lambda_i\alpha_i)}{\sin(\alpha_i)}+\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right| \leq \\
&\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\left|\frac{\frac{\sin(\delta_i\beta_i)\sin(\mu_i\alpha_{i+1})}{\sin(a_{i+1})}}{\frac{\sin(\delta_i\beta_i)\sin((1-\mu_i)\alpha_{i+1})}{\sin(\alpha_{i+1})}}\right| = \\
&\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\left|\frac{\sin(\mu_i\alpha_{i+1})}{\sin((1-\mu_i)\alpha_{i+1})}\right| = \left|\frac{\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\sin(\mu_i\alpha_{i+1})}{\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\sin((1-\mu_i)\alpha_{i+1})}\right| = \\
&\left|\frac{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\mu_i\cdot\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_{i+1}\right)}{\sin\left(\left(1-\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\mu_i\right)\cdot\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_{i+1}\right)}\right| = \\
&\left|\frac{\sin\left(0\cdot\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_{i+1}\right)}{\sin\left((1-0)\cdot\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_{i+1}\right)}\right| = \\
&\left|\frac{\sin(0)}{\sin\left(\lim\limits_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\alpha_{i+1}\right)}\right| = 0.
\end{aligned}
\tag{3.17}
$$

Again, the last expression is valid under Assumption (3.1).

Returning to the limit (3.10) with results (3.15), (3.16) and (3.17), we see that

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty}\|d_i-P_i\|\leq 0. \tag{3.18}$$

The remarkable result from (3.18) is that, as the tension values $\{\nu_{i-1},\nu_i,\nu_{i+1}\}$ tend to infinity, the control point d_i comes closer and closer to P_i, until the two points finally coincide. Remember that $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^3$. The results of our asymptotic analysis so far can be summarized in the following theorem.

Theorem 3.1. *The computation of the control points $d_i,\ i=1,2,...,n-1$ via the algorithm described in section 3.2 depends on the triplet of tension values $\{\nu_{i-1},\nu_i,\nu_{i+1}\}$. Under the Assumption 3.1, it holds that*

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} d_i = P_i, \quad i=1,2,...,n-1.$$

A direct consequence of the above theorem is the following corollary.

Corollary 3.1. *Under the Assumption 3.1, it holds that*

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} L_i = P_i, \quad i=1,2,...,n-1$$

and also

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} R_i = P_i, \quad i=1,2,...,n-1.$$

It has to be noted that Assumption 3.1 has not been justified at the point at which we introduced it. However, the conclusions at which we arrive are consistent with every one of the assumptions made. Moreover, these assumptions are also consistent with the numerical results obtained from all test cases.

3.4.2 Curve limits

Now, having solid knowledge of the behavior of the control points defining the ν-spline as the tension values tend to infinity, we will examine the behavior of the curve itself and the behavior of its derivatives at the limit for the tension values. For the rest of this section we will be working in the interval $[P_i, P_{i+1}],\ i = 1, 2, ..., n-2$. Our boundary conditions state that $d_0 = P_0$ and $d_n = P_n$, hence we will focus on the internal nodes. We will also use a local parameterization $t \in [0, 1]$ instead of a global $u \in [P_i, P_{i+1}]$, and we will follow the steps of de Casteljau's algorithm.

With a little thought, we can see that every point on the spherical spline interpolating P_i and P_{i+1} can be found from the expression

$$S(t) = G\left[G\left[G\left[P_i, R_i\right](t), G\left[R_i, L_{i+1}\right](t)\right](t), G\left[G\left[R_i, L_{i+1}\right](t), G\left[L_{i+1}, P_{i+1}\right](t)\right](t)\right](t). \tag{3.19}$$

In the previous chapter, we saw that

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} L_i = P_i = \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} R_i, \quad i = 1, 2, ..., n-1.$$

Evidently, R_i depends on the triplet $\nu_{i-1}, \nu_i, \nu_{i+1}$, while L_{i+1} depends on the triplet $\nu_i, \nu_{i+1}, \nu_{i+2}$. Applying the limits for the union of these tension values to the innermost quantities in (3.19), we have

$$\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} G\left[P_i, R_i\right](t) &= G\left[P_i, \lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} R_i\right](t) \\
&= G[P_i, P_i](t) \\
&= P_i \\
\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} G\left[L_{i+1}, P_{i+1}\right](t) &= G\left[\lim_{\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} L_{i+1}, P_{i+1}\right](t) \\
&= G\left[P_{i+1}, P_{i+1}\right](t) \\
&= P_{i+1} \\
\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} G\left[R_i, L_{i+1}\right](t) &= G\left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1}\to\infty} R_i, \lim_{\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} L_{i+1}\right](t) \\
&= G\left[P_i, P_{i+1}\right](t).
\end{aligned}$$

We can now rewrite the expression for (3.19) in the following manner:

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(t) = G\left[G\left[P_i, G[P_i, P_{i+1}](t)\right](t), G\left[G[P_i, P_{i+1}](t), P_{i+1}\right](t)\right](t). \tag{3.20}$$

As a next step, we use the explicit forms for each of the parts of the expression. Let us define $\omega = \cos^{-1}(\langle P_i, P_{i+1}\rangle)$ and $\vartheta = \cos^{-1}(\langle P_i, G[P_i, P_{i+1}](t)\rangle)$ so we can write

$$\begin{aligned}
G\left[P_i, G[P_i, P_{i+1}](t)\right](t) &= \frac{\sin((1-t)\vartheta)}{\sin(\vartheta)} \cdot P_i + \frac{\sin(t\vartheta)}{\sin(\vartheta)} \cdot G[P_i, P_{i+1}](t) \\
&= \frac{\sin((1-t)\vartheta)}{\sin(\vartheta)} \cdot P_i + \frac{\sin(t\vartheta)}{\sin(\vartheta)} \cdot \left[\frac{\sin((1-t)\omega)}{\sin(\omega)} P_i + \frac{\sin(t\omega)}{\sin(\omega)} P_{i+1}\right] \\
&= \frac{\sin((1-t)\vartheta)\sin(\omega) + \sin(t\vartheta)\sin((1-t)\omega)}{\sin(\vartheta)\sin(\omega)} \cdot P_i + \frac{\sin(t\vartheta)\sin(t\omega)}{\sin(\vartheta)\sin(\omega)} \cdot P_{i+1}.
\end{aligned} \tag{3.21}$$

Focusing on the quantity ϑ for a moment, we see that

$$\begin{aligned}
\vartheta &= \cos^{-1}(\langle P_i, G[P_i, P_{i+1}](t)\rangle) \\
&= \cos^{-1}\left(\left\langle P_i, \frac{\sin((1-t)\omega)}{\sin(\omega)} \cdot P_i + \frac{\sin(t\omega)}{\sin(\omega)} \cdot P_{i+1}\right\rangle\right) \\
&= \cos^{-1}\left(\frac{\sin((1-t)\omega)}{\sin(\omega)} \cdot \|P_i\| + \frac{\sin(t\omega)}{\sin(\omega)} \cdot \langle P_i, P_{i+1}\rangle\right).
\end{aligned}$$

Recalling that $\|P_i\| = 1$ and $\omega = \cos^{-1}(\langle P_i, P_{i+1}\rangle) \Leftrightarrow \cos(\omega) = \langle P_i, P_{i+1}\rangle$ we can write

$$\begin{aligned}
\vartheta &= \cos^{-1}\left(\frac{\sin((1-t)\omega) + \sin(t\omega)\cos(\omega)}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\sin((1-t)\omega) + \frac{1}{2}\left[\sin((1+t)\omega) - \sin((1-t)\omega)\right]}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\frac{1}{2}\left[\sin((1-t)\omega) + \sin((1+t)\omega)\right]}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\sin(\omega)\cos(t\omega)}{\sin(\omega)}\right) \\
&= \cos^{-1}(\cos(t\omega)). \\
&= t\omega
\end{aligned}$$

Applying this result to (3.21), the expression becomes

$$\begin{aligned}
G\left[P_i, G[P_i, P_{i+1}](t)\right](t) &= \frac{\sin((1-t)\vartheta)\sin(\omega) + \sin(t\vartheta)\sin((1-t)\omega)}{\sin(\vartheta)\sin(\omega)} P_i + \frac{\sin(t\vartheta)\sin(t\omega)}{\sin(\vartheta)\sin(\omega)} P_{i+1} \\
&= \frac{\sin((1-t)t\omega)\sin(\omega) + \sin(t^2\omega)\sin((1-t)\omega)}{\sin(t\omega)\sin(\omega)} P_i + \frac{\sin(t^2\omega)\sin(t\omega)}{\sin(t\omega)\sin(\omega)} P_{i+1} \\
&= \frac{\sin(t\omega)\sin((1-t^2)\omega)}{\sin(t\omega)\sin(\omega)} P_i + \frac{\sin(t^2\omega)\sin(t\omega)}{\sin(t\omega)\sin(\omega)} P_{i+1} \\
&= \frac{\sin((1-t^2)\omega)}{\sin(\omega)} P_i + \frac{\sin(t^2\omega)}{\sin(\omega)} P_{i+1}.
\end{aligned}$$

This concludes the procedure for the first member of our complex expression. Consider now $\varphi = \cos^{-1}(\langle G[P_i, P_{i+1}](t), P_{i+1}\rangle)$ and ω as previously defined. Following the same steps to tackle the second part, we have

$$
\begin{aligned}
G\left[G[P_i,P_{i+1}](t),P_{i+1}\right](t) &= \frac{\sin((1-t)\varphi)}{\sin(\varphi)}\cdot G[P_i,P_{i+1}](t)+\frac{\sin(t\varphi)}{\sin(\varphi)}\cdot P_{i+1} \\
&= \frac{\sin((1-t)\varphi)}{\sin(\varphi)}\cdot\left[\frac{\sin((1-t)\omega)}{\sin(\omega)}\cdot P_i+\frac{\sin(t\omega)}{\sin(\omega)}\cdot P_{i+1}\right]+\frac{\sin(t\varphi)}{\sin(\varphi)}\cdot P_{i+1} \\
&= \frac{\sin((1-t)\varphi)\sin((1-t)\omega)}{\sin(\varphi)\sin(\omega)}\cdot P_i\,+ \\
&\quad \frac{\sin((1-t)\varphi)\sin(t\omega)+\sin(\omega)\sin(t\varphi)}{\sin(\varphi)\sin(\omega)}\cdot P_{i+1}.
\end{aligned}
\tag{3.22}
$$

Focusing on φ, we find that

$$
\begin{aligned}
\varphi &= \cos^{-1}\left(\langle G[P_i,P_{i+1}](t),P_{i+1}\rangle\right) \\
&= \cos^{-1}\left(\left\langle\frac{\sin((1-t)\omega)}{\sin(\omega)}\cdot P_i+\frac{\sin(t\omega)}{\sin\omega}\cdot P_{i+1},P_{i+1}\right\rangle\right) \\
&= \cos^{-1}\left(\frac{\sin((1-t)\omega)}{\sin(\omega)}\cdot\langle P_i,P_{i+1}\rangle+\frac{\sin(t\omega)}{\sin\omega}\cdot\|P_{i+1}\|\right).
\end{aligned}
$$

Recall that $\|P_{i+1}\|=1$ and that $\langle P_i,P_{i+1}\rangle=\cos(\omega)$ and we can write

$$
\begin{aligned}
\varphi &= \cos^{-1}\left(\frac{\sin((1-t)\omega)\cos(\omega)+\sin(t\omega)}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\frac{1}{2}\left[\sin((1-t)\omega+\omega)-\sin(\omega-(1-t)\omega)\right]+\sin(t\omega)}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\frac{1}{2}\left[\sin((2-t)\omega)-\sin(t\omega)\right]+\sin(t\omega)}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\frac{1}{2}\left[\sin((2-t)\omega)+\sin(t\omega)\right]}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\frac{\sin(\omega)\cos((1-t)\omega)}{\sin(\omega)}\right) \\
&= \cos^{-1}\left(\cos((1-t)\omega)\right) \\
&= (1-t)\omega.
\end{aligned}
$$

Substituting back into (3.22), we get

$$\begin{aligned}
G\left[G[P_i, P_{i+1}](t), P_{i+1}\right](t) &= \frac{\sin((1-t)\varphi)\sin((1-t)\omega)}{\sin(\omega)\sin(\varphi)}P_i + \\
&\quad \frac{\sin((1-t)\varphi)\sin(t\omega)+\sin(\omega)\sin(t\varphi)}{\sin(\omega)\sin(\varphi)}P_{i+1} \\
&= \frac{\sin((1-t)^2\omega)\sin((1-t)\omega)}{\sin(\omega)\sin((1-t)\omega)}P_i + \\
&\quad \frac{\sin((1-t)^2\omega)\sin(t\omega)+\sin(\omega)\sin(t(1-t)\omega)}{\sin(\omega)\sin((1-t)\omega)}P_{i+1} \\
&= \frac{\sin((1-t)^2\omega)\sin((1-t)\omega)}{\sin(\omega)\sin((1-t)\omega)}P_i - \\
&\quad \frac{\sin((t-2)t\omega)\sin((1-t)\omega)}{\sin(\omega)\sin((1-t)\omega)}P_{i+1} \\
&= \frac{\sin((1-t)^2\omega)}{\sin(\omega)}P_i - \frac{\sin((t-2)t\omega)}{\sin(\omega)}P_{i+1} \\
&= \frac{\sin((1-t)^2\omega)}{\sin(\omega)}P_i - \frac{\sin\left(\left[(1-t)^2-1\right]\omega\right)}{\sin(\omega)}P_{i+1} \\
&= \frac{\sin((1-t)^2\omega)}{\sin(\omega)}P_i + \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin(\omega)}P_{i+1}.
\end{aligned}$$

Summarizing our results so far, we have found that

$$G\left[P_i, G[P_i, P_{i+1}](t)\right](t) = \frac{\sin((1-t^2)\omega)}{\sin(\omega)}P_i + \frac{\sin(t^2\omega)}{\sin(\omega)}P_{i+1} \tag{3.23}$$

and

$$G\left[G[P_i, P_{i+1}](t), P_{i+1}\right](t) = \frac{\sin((1-t)^2\omega)}{\sin(\omega)}P_i + \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin(\omega)}P_{i+1}. \tag{3.24}$$

Substituting (3.23) and (3.24) into (3.20) yields

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(t) = G\left[aP_i + bP_{i+1}, cP_i + dP_{i+1}\right](t) \tag{3.25}$$

with

$$a = \frac{\sin((1-t^2)\omega)}{\sin(\omega)}, \qquad b = \frac{\sin(t^2\omega)}{\sin(\omega)}$$
$$c = \frac{\sin((1-t)^2\omega)}{\sin(\omega)}, \qquad d = \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin(\omega)}.$$

The reason behind writing the expression into the above form is that we can still do better. For simplicity's sake we will first examine the quantity $\rho = \cos^{-1}\left(\langle aP_i + bP_{i+1}, cP_i + dP_{i+1}\rangle\right)$, for which we have

$$
\begin{aligned}
\cos(\rho) &= \langle aP_i + bP_{i+1}, cP_i + dP_{i+1} \rangle \\
&= ac\,\|P_i\|^2 + ad\,\langle P_i, P_{i+1}\rangle + bc\,\langle P_{i+1}, P_i\rangle + bd\,\|P_{i+1}\|^2 \\
&= ac + bd + \big[ad + bc\big]\,\langle P_i, P_{i+1}\rangle \\
&= ac + bd + \big[ad + bc\big]\cos(\omega) \\
&= c(a + b\cos(\omega)) + d(b + a\cos(\omega)) \\
&= \frac{\sin((1-t)^2\omega)}{\sin^2(\omega)}\big[\sin((1-t^2)\omega) + \sin(t^2\omega)\cos(\omega)\big] + \\
&\quad \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin^2(\omega)}\big[\sin(t^2\omega) + \sin((1-t^2)\omega)\cos(\omega)\big] \\
&= \frac{\sin((1-t)^2\omega)}{\sin^2(\omega)} \cdot \sin(\omega)\cos(t^2\omega) + \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin^2(\omega)} \cdot \sin(\omega)\cos((1-t^2)\omega) \\
&= \frac{\sin((1-t)^2\omega)\cos(t^2\omega)}{\sin(\omega)} + \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)\cos((1-t^2)\omega)}{\sin(\omega)} \\
&= \frac{\sin(\omega - 2t\omega + t^2\omega)\cos(t^2\omega)}{\sin(\omega)} + \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)\cos(\omega - t^2\omega)}{\sin(\omega)} \\
&= \frac{\sin(\omega)\cos(-2t\omega + t^2\omega)\cos(t^2\omega)}{\sin(\omega)} + \cancel{\frac{\cos(\omega)\sin(-2t\omega + t^2\omega)\cos(t^2\omega)}{\sin(\omega)}} \\
&\quad + \cancel{\frac{\sin(2t\omega - t^2\omega)\cos(\omega)\cos(-t^2\omega)}{\sin(\omega)}} + \frac{\sin(2t\omega - t^2\omega)\sin(\omega)\sin(-t^2\omega)}{\sin(\omega)} \\
&= \cos(-2t\omega + t^2\omega)\cos(t^2\omega) + \sin(2t\omega - t^2\omega)\sin(-t^2\omega) \\
&= \cos(-2t\omega + t^2\omega)\cos(t^2\omega) - \sin(2t\omega - t^2\omega)\sin(t^2\omega) \\
&= \cos(-2t\omega + t^2\omega + t^2\omega) \\
&= \cos(2t(t-1)\omega).
\end{aligned}
$$

The final result states that $\rho = \cos^{-1}\left(\langle aP_i + bP_{i+1}, cP_i + dP_{i+1}\rangle\right) = 2t(t-1)\omega$, hence (3.25) can be rewritten as

$$
\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(t) &= G\left[aP_i + bP_{i+1}, cP_i + dP_{i+1}\right](t) \\
&= \frac{\sin((1-t)\rho)}{\sin(\rho)} \cdot \left[aP_i + bP_{i+1}\right] + \frac{\sin(t\rho)}{\sin(\rho)} \cdot \left[cP_i + dP_{i+1}\right] \\
&= \frac{\sin(2t(1-t)^2\omega)}{\sin(2t(1-t)\omega)} \left[\frac{\sin((1-t^2)\omega)}{\sin(\omega)} P_i + \frac{\sin(t^2\omega)}{\sin(\omega)} P_{i+1}\right] + \\
&\quad \frac{\sin(2t^2(1-t)\omega)}{\sin(2t(1-t)\omega)} \left[\frac{\sin((1-t)^2\omega)}{\sin(\omega)} P_i + \frac{\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin(\omega)} P_{i+1}\right] \\
&= \frac{\sin(2t(1-t)^2\omega)\sin((1-t^2)\omega) + \sin(2t^2(1-t)\omega)\sin((1-t)^2\omega)}{\sin(2t(1-t)\omega)\sin(\omega)} P_i + \\
&\quad \frac{\sin(2t(1-t)^2\omega)\sin(t^2\omega) + \sin(2t^2(1-t)\omega)\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin(2t(1-t)\omega)\sin(\omega)} P_{i+1} \\
&= A \cdot P_i + B \cdot P_{i+1}
\end{aligned}
$$

with

$$
A = \frac{\sin(2t(1-t)^2\omega)\sin((1-t^2)\omega) + \sin(2t^2(1-t)\omega)\sin((1-t)^2\omega)}{\sin(2t(1-t)\omega)\sin(\omega)}
$$

and

$$
B = \frac{\sin(2t(1-t)^2\omega)\sin(t^2\omega) + \sin(2t^2(1-t)\omega)\sin\left(\left[1-(1-t)^2\right]\omega\right)}{\sin(2t(1-t)\omega)\sin(\omega)}.
$$

The last step is to further simplify the quantities A and B.

For A we have

$$\begin{aligned}
A &= \frac{\sin(2t^3\omega - 4t^2\omega + 2t\omega)\sin(\omega - t^2\omega) + \sin(2t^2\omega - 2t^3\omega)\sin(\omega - 2t\omega + t^2\omega)}{\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cancel{\cos(2t^3\omega - 3t^2\omega + 2t\omega - \omega)} - \cos(2t^3\omega - 5t^2\omega + 2t\omega + \omega)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&\quad + \frac{\cos(-2t^3\omega + t^2\omega + 2t\omega - \omega) \cancel{- \cos(-2t^3\omega + 3t^2\omega - 2t\omega + \omega)}}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cos(-2t^3\omega + t^2\omega + 2t\omega - \omega) - \cos(2t^3\omega - 5t^2\omega + 2t\omega + \omega)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{-2\sin\left(\frac{-2t^3\omega + t^2\omega + 2t\omega - \omega + 2t^3\omega - 5t^2\omega + 2t\omega + \omega}{2}\right)\sin\left(\frac{-2t^3\omega + t^2\omega + 2t\omega - \omega - (2t^3\omega - 5t^2\omega + 2t\omega + \omega)}{2}\right)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{-2\sin\left(\frac{-4t^2\omega + 4t\omega}{2}\right)\sin\left(\frac{-2t^3\omega + t^2\omega + 2t\omega - \omega - 2t^3\omega + 5t^2\omega - 2t\omega - \omega)}{2}\right)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cancel{-2\sin(-2t^2\omega + 2t\omega)}\sin\left(\frac{-4t^3\omega + 6t^2\omega - 2\omega}{2}\right)}{\cancel{2\sin(2t\omega - 2t^2\omega)}\sin(\omega)} \\
&= \frac{-\sin(-2t^3\omega + 3t^2\omega - \omega)}{\sin(\omega)} \\
&= \frac{\sin(2t^3\omega - 3t^2\omega + \omega)}{\sin(\omega)} \\
&= \frac{\sin\left((1-t)^2(2t+1)\omega\right)}{\sin(\omega)}.
\end{aligned}$$

For B, the calculations go as follows.

$$\begin{aligned}
B &= \frac{\sin(2t^3\omega - 4t^2\omega + 2t\omega)\sin(t^2\omega) + \sin(2t^2\omega - 2t^3\omega)\sin(2t\omega - t^2\omega)}{\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cos(2t^3\omega - 4t^2\omega + 2t\omega - t^2\omega) - \cos(2t^3\omega - 4t^2\omega + 2t\omega + t^2\omega)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&\quad + \frac{\cos(2t^2\omega - 2t^3\omega - (2t\omega - t^2\omega)) - \cos(2t^2\omega - 2t^3\omega + 2t\omega - t^2\omega)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cos(2t^3\omega - 5t^2\omega + 2t\omega) - \cancel{\cos(2t^3\omega - 3t^2\omega + 2t\omega)}}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&\quad + \frac{\cancel{\cos(-2t^3\omega + 3t^2\omega - 2t\omega)} - \cos(-2t^3\omega + t^2\omega + 2t\omega)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cos(2t^3\omega - 5t^2\omega + 2t\omega) - \cos(-2t^3\omega + t^2\omega + 2t\omega)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{-2\sin\left(\frac{2t^3\omega - 5t^2\omega + 2t\omega - 2t^3\omega + t^2\omega + 2t\omega}{2}\right)\sin\left(\frac{2t^3\omega - 5t^2\omega + 2t\omega - (-2t^3\omega + t^2\omega + 2t\omega)}{2}\right)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{-2\sin\left(\frac{-4t^2\omega + 4t\omega}{2}\right)\sin\left(\frac{4t^3\omega - 6t^2\omega}{2}\right)}{2\sin(2t\omega - 2t^2\omega)\sin(\omega)} \\
&= \frac{\cancel{-2\sin(-2t^2\omega + 2t\omega)}\sin(2t^3\omega - 3t^2\omega)}{\cancel{2\sin(2t\omega - 2t^2\omega)}\sin(\omega)} \\
&= \frac{-\sin(2t^3\omega - 3t^2\omega)}{\sin(\omega)} \\
&= \frac{\sin(-2t^3\omega + 3t^2\omega)}{\sin(\omega)} \\
&= \frac{\sin\left(t^2(2(1-t)+1)\omega\right)}{\sin(\omega)}.
\end{aligned}$$

After all this work, (3.25) takes the amazingly beautiful form

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(t) = \frac{\sin\left((1-t)^2(2t+1)\omega\right)}{\sin(\omega)} \cdot P_i + \frac{\sin\left(t^2(2(1-t)+1)\omega\right)}{\sin(\omega)} \cdot P_{i+1} \tag{3.26}$$

which can be rewritten as

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(t) = \Phi(1-t;\omega) \cdot P_i + \Phi(t;\omega) \cdot P_{i+1} \tag{3.27}$$

with

$$\Phi(t;\omega) = \frac{\sin\left(t^2(2(1-t)+1)\omega\right)}{\sin(\omega)}.$$

Recall that we are working in the interval $[P_i, P_{i+1}],\ i = 1, 2, ..., n-2$ with $t \in [0, 1]$ being the "local" parameter.

It is easily verified for the above expression that

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(0) = P_i \quad \text{and} \quad \lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} S(1) = P_{i+1}.$$

hence expression (3.26) still describes an interpolant, as expected. However, it is clear that the interpolant is now defined by only two control points, which means that it is indeed a form of geodesic interpolant. The parameterization of the curve is different from the parameterization used in the linear geodesic interpolation, but for $t \in [0, 1]$ the above expression describes a point on the geodesic between P_i and P_{i+1}. One way to perceive this is relevant to the manner in which we explained C- and G-continuity. If we suppose that $S(t)$ describes the motion of a rigid body from point P_i to point P_{i+1}, the geodesic provides the minimum path between P_i and P_{i+1}, and moreover the body will be moving with constant velocity for the whole duration of the motion. Considering the limit of $S(t)$ provides a different parameterization, meaning that the velocity of the body will not be constant along the path, but will change depending on how far from each endpoint it is. The *trace* of the body, however, will be the same, being the geodesic from P_i to P_{i+1}.

Now, regarding the limit of the derivatives $\dot{S}(t)$ and $\ddot{S}(t)$, we will only present the resulting expressions without reference to the extensive calculations. We employ the limits for the angles ϑ, φ and ρ previously proven, and also exploit the limits for the quantities L_{i+1} and R_i. Replacing everything in the explicit form of the first derivative of $S(t)$ from (3.19) yields

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \dot{S}(t) = \Psi(1-t;\omega) \cdot P_i - \Psi(t;\omega) \cdot P_{i+1}$$

where

$$\begin{aligned}\Psi(t;\omega) = {} & \frac{2\,t(1-t)\omega\,\cos\left(2\,t^2(1-t)\omega\right)\sin(t(1-t)\omega) - (1-t)\omega\,\sin\left(2\,t^2(1-t)\omega\right)\cos(t(1-t)\omega)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)} - \\ & \frac{\left(-2\,t(1-t)\omega\,\cos\left(2\,t^2(1-t)\omega\right)\sin\left((1-t)^2\omega\right) + (1-t)\omega\,\sin\left(2\,t^2(1-t)\omega\right)\cos\left((1-t)^2\omega\right)\right)\sin(t\omega)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} - \\ & \frac{2\,t(1-t)\omega\,\cos\left(2\,(1-t)^2 t\omega\right)\sin\left(t^2\omega\right) - t\omega\,\sin\left(2\,(1-t)^2 t\omega\right)\cos\left(t^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin(\omega)} - \\ & \frac{\sin\left(2\,t^2(1-t)\omega\right)\sin\left((1-t)^2\omega\right)\omega\,\cos((1-t)\omega)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} - \\ & \frac{\sin\left(2\,(1-t)^2 t\omega\right)\sin\left(t^2\omega\right)\omega\,\cos((1-t)\omega)}{\sin(2\,t(1-t)\omega)\sin(t\omega)\sin(\omega)}\end{aligned}$$

It may not be immediately evident, but it can be verified that

$$\lim_{t\to 0} \Psi(t;\omega) = 0 = \lim_{t\to 1} \Psi(t;\omega),$$

which, in turn, implies that

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \dot{S}(0) = 0 = \lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \dot{S}(1).$$

Performing the same steps for the second derivative, we arrive at a similar expression. Namely, we have

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \ddot{S}(t) = \Xi(1-t;\omega)\cdot P_i - \Xi(t;\omega)\cdot P_{i+1}$$

where $\Xi(t;\omega)$ is defined on the (entire) next page.

$$
\begin{aligned}
\Xi(t;\omega) = {} & \frac{\sin\left(2\,(1-t)^2 t\omega\right)\sin\left(t^2\omega\right)\omega^2\cos((1-t)\omega)\cos(t\omega)}{(\sin(t\omega))^2\sin(2\,t(1-t)\omega)\sin(\omega)} + \frac{2\,t(1-t)\omega^2\sin\left((1-t)^2\omega\right)\cos\left(2\,t^2(1-t)\omega\right)\cos(t\omega)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} - \frac{(1-t)\omega^2\sin\left(2\,t^2(1-t)\omega\right)\cos\left((1-t)^2\omega\right)\cos(t\omega)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} - \\
& \frac{2\,t\omega^2\sin\left(2\,(1-t)^2 t\omega\right)\cos((1-t)\omega)\cos\left(t^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin(t\omega)\sin(\omega)} - \frac{t\omega\,\sin\left(2\,(1-t)^2 t\omega\right)(2\,(1-t)\omega-2\,t\omega)\cos(2\,t(1-t)\omega)\cos\left(t^2\omega\right)}{(\sin(2\,t(1-t)\omega))^2\sin(\omega)} + \\
& \frac{\left(-4\,t^2(1-t)\omega^2+t\omega\,(-2\,t(1-t)\omega+(1-t)(2\,(1-t)\omega-2\,t\omega))\right)\cos\left(2\,(1-t)^2 t\omega\right)\cos\left(t^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin(\omega)} + \frac{\omega\,\sin\left(2\,(1-t)^2 t\omega\right)\cos\left(t^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin(\omega)} - \\
& \frac{\sin\left(2\,t^2(1-t)\omega\right)\sin\left((1-t)^2\omega\right)\omega^2(\cos((1-t)\omega))^2}{(\sin((1-t)\omega))^2\sin(2\,t(1-t)\omega)\sin(\omega)} + \frac{\cos\left((1-t)^2\omega\right)\cos((1-t)\omega)}{\sin(\omega)}\left(-\frac{(1-t)\omega^2\sin\left(2\,t^2(1-t)\omega\right)\sin(t\omega)}{\sin(2\,t(1-t)\omega)(\sin((1-t)\omega))^2} + \frac{2\,(1-t)\omega^2\sin\left(2\,t^2(1-t)\omega\right)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)}\right) + \\
& \frac{\cos(2\,t(1-t)\omega)\cos((1-t)\omega)}{\sin(\omega)}\left(\frac{\sin\left(2\,t^2(1-t)\omega\right)(2\,(1-t)\omega-2\,t\omega)\sin\left((1-t)^2\omega\right)\omega}{(\sin(2\,t(1-t)\omega))^2\sin((1-t)\omega)} + \frac{\sin\left(2\,(1-t)^2 t\omega\right)(2\,(1-t)\omega-2\,t\omega)\sin\left(t^2\omega\right)\omega}{(\sin(2\,t(1-t)\omega))^2\sin(t\omega)}\right) + \\
& \left(\frac{2\,t(1-t)\omega^2\sin(t(1-t)\omega)}{\sin(2\,t(1-t)\omega)(\sin((1-t)\omega))^2} + \frac{1}{\sin(\omega)}\left(\frac{2\,t(1-t)\omega^2\sin\left((1-t)^2\omega\right)\sin(t\omega)}{\sin(2\,t(1-t)\omega)(\sin((1-t)\omega))^2} - \frac{(2\,t(1-t)\omega+t(2\,(1-t)\omega-2\,t\omega))\sin\left((1-t)^2\omega\right)\omega}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)}\right)\right)\cos\left(2\,t^2(1-t)\omega\right)\cos((1-t)\omega) - \\
& \frac{(-2\,t(1-t)\omega+(1-t)(2\,(1-t)\omega-2\,t\omega))\omega\,\sin\left(t^2\omega\right)\cos\left(2\,(1-t)^2 t\omega\right)\cos((1-t)\omega)}{\sin(2\,t(1-t)\omega)\sin(t\omega)\sin(\omega)} + \frac{(1-t)\omega^2\sin\left(2\,t^2(1-t)\omega\right)\cos(t(1-t)\omega)\cos((1-t)\omega)}{\sin(2\,t(1-t)\omega)(\sin((1-t)\omega))^2} + \\
& \frac{(1-t)\omega\,\sin\left(2\,t^2(1-t)\omega\right)(2\,(1-t)\omega-2\,t\omega)\sin(t\omega)\cos(2\,t(1-t)\omega)\cos\left((1-t)^2\omega\right)}{(\sin(2\,t(1-t)\omega))^2\sin((1-t)\omega)\sin(\omega)} + \\
& \frac{\left(-4\,t(1-t)^2\omega^2-(1-t)\omega\,(2\,t(1-t)\omega+t(2\,(1-t)\omega-2\,t\omega))\right)\sin(t\omega)\cos\left(2\,t^2(1-t)\omega\right)\cos\left((1-t)^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} + \frac{\omega\,\sin\left(2\,t^2(1-t)\omega\right)\sin(t\omega)\cos\left((1-t)^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} + \\
& \left(\frac{-2\,t(1-t)\omega\,(2\,(1-t)\omega-2\,t\omega)\sin\left((1-t)^2\omega\right)\sin(t\omega)}{(\sin(2\,t(1-t)\omega))^2\sin((1-t)\omega)\sin(\omega)} - \frac{2\,t(1-t)\omega\,(2\,(1-t)\omega-2\,t\omega)\sin(t(1-t)\omega)}{(\sin(2\,t(1-t)\omega))^2\sin((1-t)\omega)}\right)\cos\left(2\,t^2(1-t)\omega\right)\cos(2\,t(1-t)\omega) - \\
& \frac{(1-t)\omega\,\sin\left(2\,t^2(1-t)\omega\right)(2\,(1-t)\omega-2\,t\omega)\cos(t(1-t)\omega)\cos(2\,t(1-t)\omega)}{(\sin(2\,t(1-t)\omega))^2\sin((1-t)\omega)} + \frac{2\,t(1-t)\omega\,\cos\left(2\,(1-t)^2 t\omega\right)(2\,(1-t)\omega-2\,t\omega)\sin\left(t^2\omega\right)\cos(2\,t(1-t)\omega)}{(\sin(2\,t(1-t)\omega))^2\sin(\omega)} + \\
& \frac{(2\,(1-t)\omega\,\sin(t(1-t)\omega)-2\,t\sin(t(1-t)\omega)\omega+2\,t(1-t)\omega\,((1-t)\omega\,\cos(t(1-t)\omega)-t\cos(t(1-t)\omega)\omega)+(1-t)\omega\,(2\,t(1-t)\omega+t(2\,(1-t)\omega-2\,t\omega))\cos(t(1-t)\omega))\cos\left(2\,t^2(1-t)\omega\right)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)} + \\
& \frac{(2\,(1-t)\omega-2\,t\omega)\sin\left((1-t)^2\omega\right)\sin(t\omega)\cos\left(2\,t^2(1-t)\omega\right)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} + \frac{(-2\,(1-t)\omega+2\,t\omega)\sin\left(t^2\omega\right)\cos\left(2\,(1-t)^2 t\omega\right)}{\sin(2\,t(1-t)\omega)\sin(\omega)} - \\
& \frac{(2\,t(1-t)\omega\,(-2\,t(1-t)\omega-t(2\,(1-t)\omega-2\,t\omega))\sin(t(1-t)\omega)-\omega\,\cos(t(1-t)\omega)+(1-t)\omega\,(-(1-t)\omega\,\sin(t(1-t)\omega)+t\sin(t(1-t)\omega)\omega))\sin\left(2\,t^2(1-t)\omega\right)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)} \\
& \frac{\left(2\,t(1-t)\omega\,(-2\,t(1-t)\omega-t(2\,(1-t)\omega-2\,t\omega))-2\,(1-t)^2\omega^2\right)\sin\left(2\,t^2(1-t)\omega\right)\sin\left((1-t)^2\omega\right)\sin(t\omega)}{\sin(2\,t(1-t)\omega)\sin((1-t)\omega)\sin(\omega)} + \\
& \frac{\left(2\,t(1-t)\omega\,(-2\,t(1-t)\omega+(1-t)(2\,(1-t)\omega-2\,t\omega))-2\,t^2\omega^2\right)\sin\left(2\,(1-t)^2 t\omega\right)\sin\left(t^2\omega\right)}{\sin(2\,t(1-t)\omega)\sin(\omega)} - \frac{\sin\left(2\,t^2(1-t)\omega\right)\sin\left((1-t)^2\omega\right)\omega^2}{\sin(2\,t(1-t)\omega)\sin(\omega)} - \frac{\sin\left(2\,(1-t)^2 t\omega\right)\sin\left(t^2\omega\right)\omega^2\sin((1-t)\omega)}{\sin(2\,t(1-t)\omega)\sin(t\omega)\sin(\omega)}
\end{aligned}
$$

Now, we wish to examine the behavior of the cross-product of the first and second derivative of the spline, as the tension values go to infinity. Let us consider

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \dot{S}(t) = \alpha P_i - \beta P_{i+1} \quad \text{and} \quad \lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \ddot{S}(t) = \gamma P_i + \delta P_{i+1}$$

with

$$\begin{aligned}
\alpha &:= \Psi(1-t,\omega),\\
\beta &:= \Psi(t,\omega),\\
\gamma &:= \Xi(1-t,\omega),\\
\delta &:= \Xi(t,\omega).
\end{aligned}$$

We use the knowledge that the cross-product is distributive, meaning that

$$\kappa \times (\tau + \sigma) = (\kappa \times \tau) + (\kappa \times \sigma) \quad \text{and} \quad (\kappa + \tau) \times \sigma = (\kappa \times \sigma) + (\tau \times \sigma).$$

Thus we have

$$\begin{aligned}
(\alpha P_i - \beta P_{i+1}) \times (\gamma P_i + \delta P_{i+1}) &= (\alpha P_i - \beta P_{i+1}) \times \gamma P_i + (\alpha P_i - \beta P_{i+1}) \times \delta P_{i+1}\\
&= \alpha\gamma(P_i \times P_i) - \beta\gamma(P_{i+1} \times P_i) + \alpha\delta(P_i \times P_{i+1}) - \beta\delta(P_{i+1} \times P_{i+1})\\
&= \alpha\delta(P_i \times P_{i+1}) + \beta\gamma(P_i \times P_{i+1})\\
&= (\alpha\delta + \beta\gamma)(P_i \times P_{i+1}).
\end{aligned}$$

Hence, we conclude that

$$\begin{aligned}
\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \left[\dot{S}(t) \times \ddot{S}(t)\right] &= \left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \dot{S}(t)\right] \times \left[\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \ddot{S}(t)\right]\\
&= (\alpha\delta + \beta\gamma)(P_i \times P_{i+1})\\
&= \left[\Psi(1-t,\omega)\,\Xi(t,\omega) + \Psi(t,\omega)\,\Xi(1-t,\omega)\right](P_i \times P_{i+1}).
\end{aligned}$$

Substituting

$$\Lambda(t;\omega) = \Psi(1-t,\omega)\,\Xi(t,\omega) + \Psi(t,\omega)\,\Xi(1-t,\omega)$$

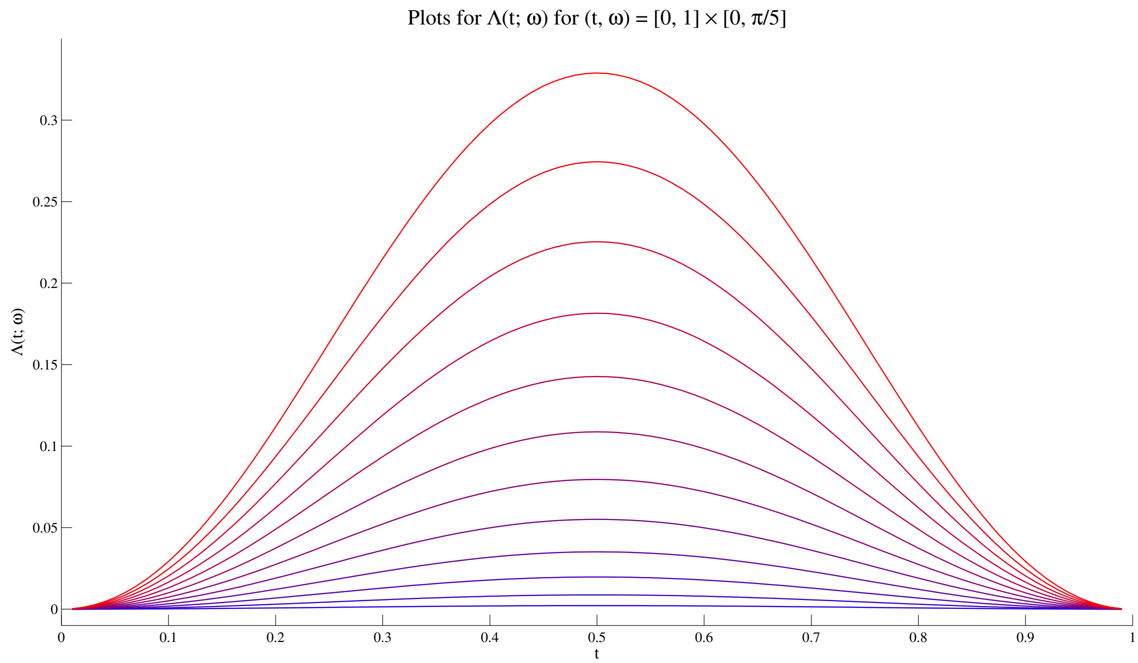
allows us to write

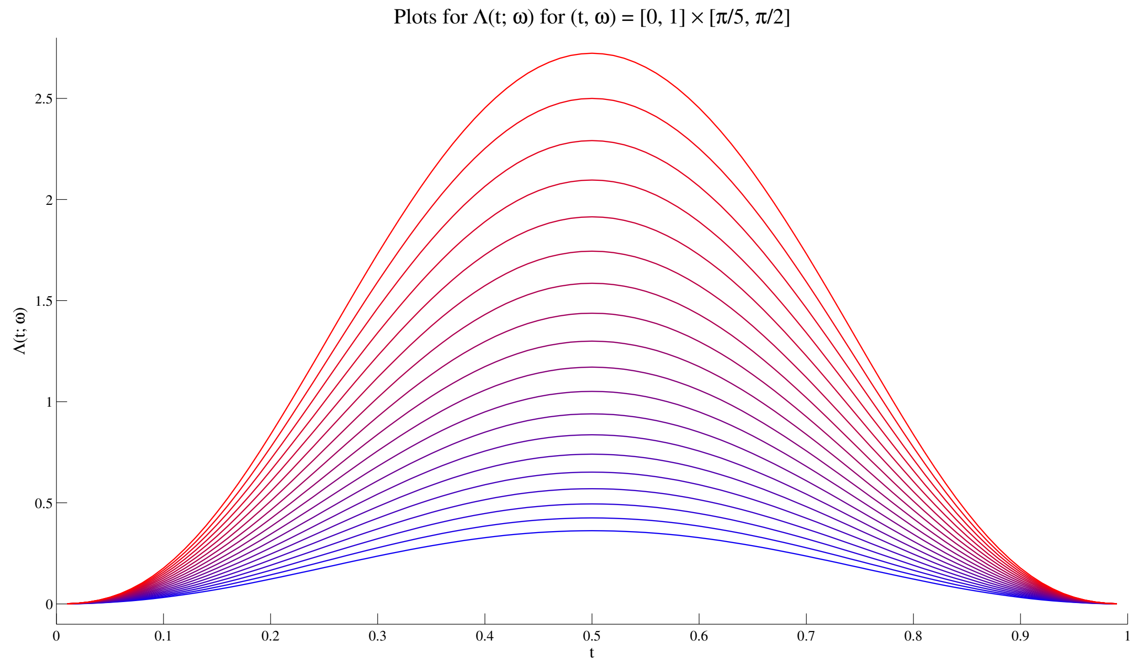
$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty} \left[\dot{S}(t) \times \ddot{S}(t)\right] = \Lambda(t;\omega)\left(P_i \times P_{i+1}\right). \tag{3.28}$$

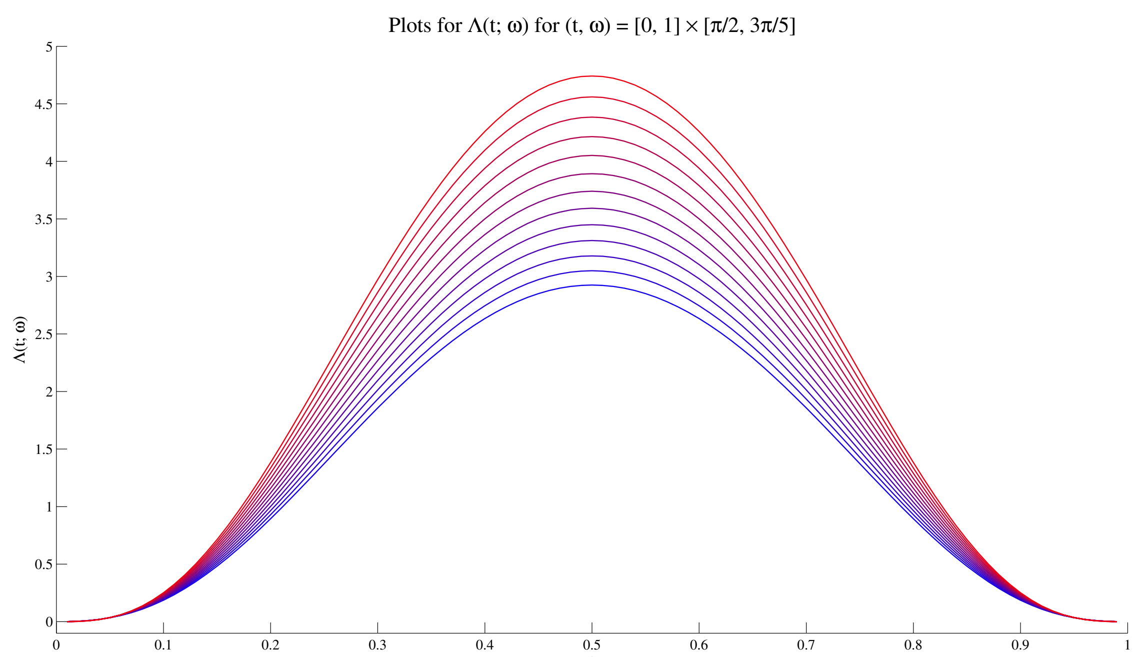
We said previously that the limit of the quantity Ψ goes to zero at both $t = 0$ and $t = 1$, thus we gather that

$$\lim_{t\to 0} \Lambda(t;\omega) = 0 = \lim_{t\to 1} \Lambda(t;\omega).$$

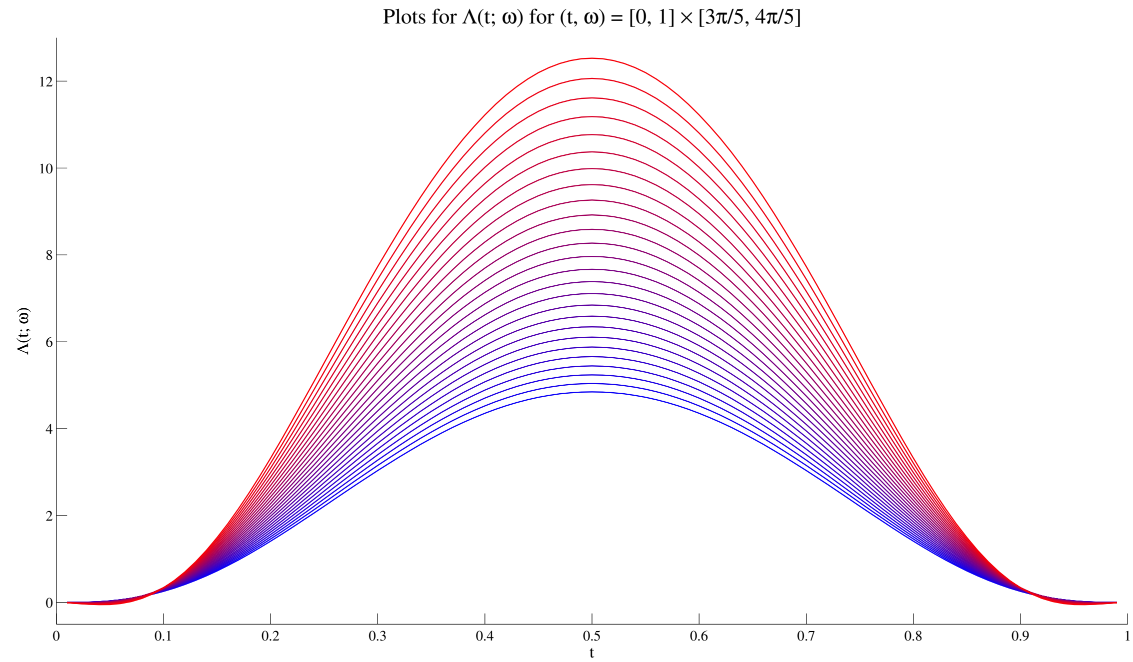
It is not easily proved analytically, but the quantity $\Lambda(t;\omega)$ has been found computationally to be non-negative for $(t,\omega) \in [0,1] \times [0, 3\pi/5]$. These numerical results can be misleading, however, and we confine our domain of trust to be for $(t,\omega) \in [0,1] \times [0,\pi/2]$. We include the plots for $\Lambda(t;\omega)$ for $t \in [0,1]$ and with ω varying from 0 to π. In the following three figures, $\Lambda(t;\omega)$ is non-negative for all $t \in [0,1]$. We have broken down the plots in various intervals for ω in order to be able to show the progression of the quantity at all stages – it would be difficult to make out anything if we plotted everything together. Cooler colors (blue) are used for lower values of ω, while warmer colors (red) represent higher values for ω.

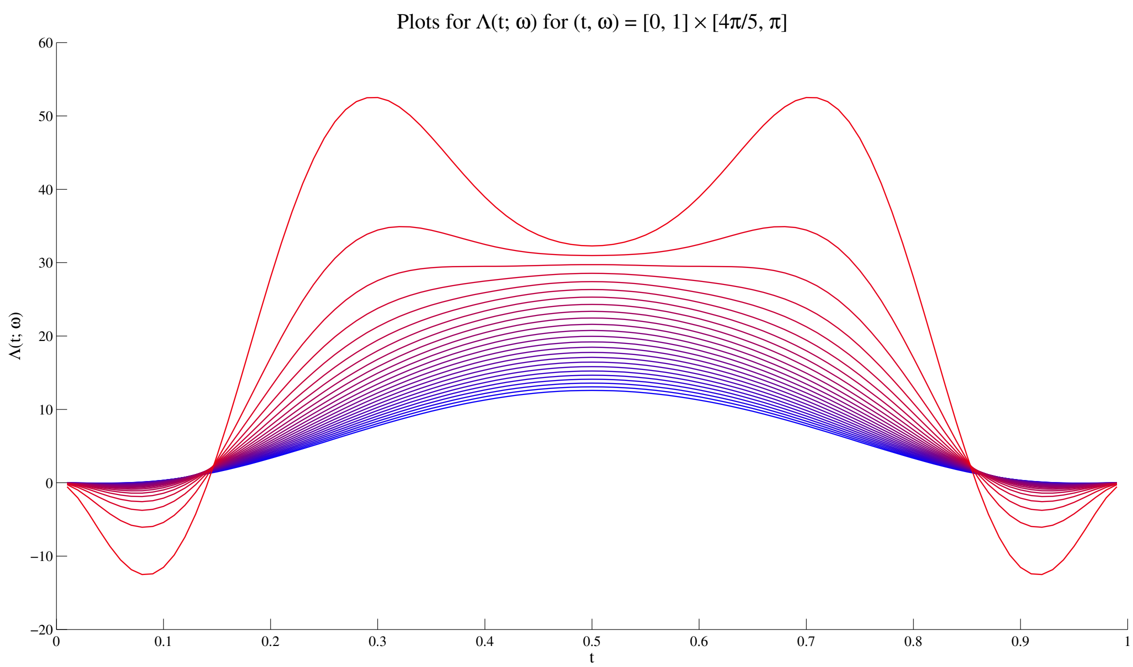






The final two plots show exclusively values for ω for which $\Lambda(t;\omega)$ becomes negative at some $t \in [0,1]$. It is not observable in the first figure, but near the ends ($t = 0$ and $t = 1$) the function becomes negative. This behavior is magnified for higher values of ω.





Based on the premise that the maximum distance between consecutive input points is $\pi/2$, which is one of our main requirements, we can see that the sign of the cross-product $\dot{S}(t)\times\ddot{S}(t)$ agrees with the sign of $P_i \times P_{i+1}$ when the quadruple of tension values $\{\nu_{i-1}, \nu_i, \nu_{i+1}, \nu_{i+2}\}$ goes to infinity.

Moreover, we can see that

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\left[\dot{S}(0)\times\ddot{S}(0)\right]=0=\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\left[\dot{S}(1)\times\ddot{S}(1)\right].$$

Bearing these results in mind, let us examine now the behavior of the geodesic curvature $\kappa_g(t;S_i)$ as the tension values affecting S_i tend to infinity. This is exactly the limit

$$\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\kappa_g(t;S_i)=\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\frac{S_i(t)\cdot\left[\dot{S}_i(t)\times\ddot{S}_i(t)\right]}{\left\|\dot{S}_i(t)\right\|^3},\quad t\in[0,1] \tag{3.29}$$

The denominator in the last expression is a positive quantity for $t\in(0,1)$, thus the sign of the curvature as the tension values tend to infinity is defined only by the numerator. Considering only the top part of the fraction, we have

$$\begin{aligned}\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}S_i(t)\cdot\left[\dot{S}_i(t)\times\ddot{S}_i(t)\right]&=\left[\Phi(1-t;\omega)P_i+\Phi(t;\omega)P_{i+1}\right]\cdot\left[\Lambda(t;\omega)\left(P_i\times P_{i+1}\right)\right]\\&=\Phi(1-t;\omega)\,\Lambda(t;\omega)\left[P_i\cdot\left(P_i\times P_{i+1}\right)\right]\\&\quad+\Phi(t;\omega)\,\Lambda(t;\omega)\left[P_{i+1}\cdot\left(P_i\times P_{i+1}\right)\right]\end{aligned} \tag{3.30}$$

Relation (3.4.2) tell us that when the appropriate tension values go to infinity, the geodesic curvature becomes zero. The triple products in the last expression indeed amount to 0, and this result indicates that when the quadruple of tension values $\{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\}$ tend to infinity, then the curve in the segment i tends to coincide with the geodesic, thus having zero curvature. Let us not forget that the limits for the quantity Λ for $t\to 0$ and $t\to 1$ are 0, so we can conclude that this behavior is manifested for $t\in[0,1]$.

Thus, we arrive at the main results of our analysis which are summarized in the following theorem.

Theorem 3.2. *The behavior of the ν-spline in the segment $[P_i,P_{i+1}]$, $i=1,2,...,n-2$ is determined by the values of the tension parameters $\{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\}$. When these values tend to infinity, the following limits hold.*

$$\begin{aligned}\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}S(t)&=\Phi(1-t;\omega)P_i+\Phi(t;\omega)P_{i+1}, && t\in[0,1],\\\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\dot{S}(t)&=\Psi(1-t;\omega)P_i-\Psi(t;\omega)P_{i+1}, && t\in[0,1],\\\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\ddot{S}(t)&=\Xi(1-t;\omega)P_i+\Xi(t;\omega)P_{i+1}, && t\in[0,1],\\\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\left[\dot{S}(t)\times\ddot{S}(t)\right]&=\Lambda(t;\omega)\left(P_i\times P_{i+1}\right), && t\in[0,1],\\\lim_{\nu_{i-1},\nu_i,\nu_{i+1},\nu_{i+2}\to\infty}\kappa_g(t;S_i)&=0, && t\in[0,1]\end{aligned}$$

where the quantities $\Phi(t;\omega),\Psi(t;\omega),\Xi(t;\omega),\Lambda(t;\omega)$ are as previously defined.

The natural question to ask is, what do our results mean? Actually, the results from the asymptotic analysis are consistent with the brief remark in section 2.2: it is as if there were a winch at each interpolation node, and turning the winch causes the spline to come closer and closer to the geodesic between the point and its two immediate neighbors.

Formally, we witnessed the behavior of the quantity $\Lambda(t\,;\omega)$ which gives us information about the shape of the spline in a segment as the appropriate tension values tend to infinity. We concluded that if the distance between the input points does not exceed $3\pi/5$, then this quantity is positive, and the curve will follow the alignment of the geodesic in this segment. We have also seen that the geodesic curvature tends to become zero in this segment, and this result also fortifies our hypothesis that increasing the quadruple of tension values $\{\nu_{i-1}, \nu_i, \nu_{i+1}, \nu_i\}$ we will achieve the shape of the geodesics. Our belief is that eventually κ_g will indeed become 0, but it is essential that it does so from values whose sign agrees with the sign of the convexity indicators Q_i. The experimental results collected suggest that this is indeed the case, and numerical findings support our claim. As the tension values increase, the spline indeed takes the form of the piecewise geodesic interpolant, the control points d_i tend to coincide with the interpolation points P_i, and the auxiliary values L_i and R_i also tend to the same nodes. Therefore, as the curve approaches the poly-geodesic, the criteria which we have established in order to ensure shape preservation are satisfied in a straightforward manner:

- The co-circularity criterion for the input point $P_i,\ i = 1, 2, ..., n-1$ is eventually satisfied if we increase the tension values for the computation of the control point d_i, which means that in order to satisfy the co-circularity criterion, we need to increase the values of the triplet $\{\nu_{i-1}, \nu_i, \nu_{i+1}\}$.
- Nodal convexity for the input point $P_i,\ i = 1, 2, ..., n-1$ is achieved if the control point d_i is sufficiently close to P_i, hence yielding a curve which will "follow" the geodesic on both sides of P_i. This again translates to increasing the values of the triplet $\{\nu_{i-1}, \nu_i, \nu_{i+1}\}$.
- Based on our last discussion, segment convexity is satisfied by aligning the curve with the geodesic in the segment $[P_i, P_{i+1}]$. In order to achieve this, we have seen that the tension values to increase are the quadruple $\{\nu_{i-1}, \nu_i, \nu_{i+1}, \nu_{i+2}\}$.

3.5 Algorithm & Implementation

We have discussed the construction of spherical ν-splines via the method described in [2]. We have also described the shape-preservation criteria needed by our application, based on the notions introduced in [1]. We are now ready to give an iterative algorithm which, given a set of points $\mathcal{P}$ on the unit sphere, automatically determines the set of control points $d_i,\ i = 0, 1, ..., n$ and tension values $\nu_i,\ i = 0, 1, ..., n$ for which the resulting spherical ν-spline interpolates the point set $\mathcal{P}$ and satisfies the requirements for shape-preservation as formulated in $\boldsymbol{\mathcal{S}}$.

The input to the algorithm is summarized below.

$\mathcal{P}$	Set of points on the unit sphere
B_{tol}	Tolerance parameter for the Bisection method (see below)
$B_{N_{\max}}$	Maximum number of iterations for the Bisection method (see below)
S_{tol}	Tolerance parameter for the spline computation method (Nielson)
$S_{N_{\max}}$	Maximum number of iterations for the spline computation algorithm (Nielson)
ε	Tolerance parameter for the geodesic curvature (co-circularity)
h	Step size parameter for the finite difference method

With this input we summarize the algorithm in the following steps.

1. Preliminary steps: initialize auxiliary variables
   (a) Create a set of initial tension values $\nu_i^{(0)} = 0,\ i = 0, 1, ..., n$.
   (b) Create an index set for which to examine co-circularity, i.e.,
   $$\mathcal{C} = \left\{i : \left[Q_i = 0\right] \wedge \left[\theta(P_{i-1}, P_{i+1}) > \max\left\{\theta(P_{i-1}, P_i), \theta(P_i, P_{i+1})\right\}\right], i \in \{1, 2, ..., n-1\}\right\}$$
   (c) Create an index set for which to examine nodal convexity, i.e.,
   $$\mathcal{K} = \{i : Q_i \neq 0, \quad i \in \{1, 2, ..., n-1\}\}$$
   (d) Create an index set for which to examine segment convexity, i.e.,
   $$\mathcal{S} = \{i : Q_i\, Q_{i+1} > 0, \quad i \in \{1, 2, ..., n-2\}\}$$
2. Iterate
   (a) Assign the set of problematix indices $\mathcal{I} = \emptyset$.
   (b) Compute the interpolating spline with the current $\nu-$values by using the algorithm described in [2]. Employ the input parameters S_{tol} and $S_{N_{\max}}$.
   (c) For each $i \in \mathcal{C}$ verify that the co-circularity criterion holds. If the check fails, set
   $$\mathcal{I} = \mathcal{I} \cup \{i-1, i, i+1\}\,.$$
   (d) For each index $i \in \mathcal{K}$ verify that the nodal convexity criterion holds. If the check fails, set
   $$\mathcal{I} = \mathcal{I} \cup \{i-1, i, i+1\}\,.$$
   (e) For each index $i \in \mathcal{S}$, verify that segment convexity is satisfied. If the check fails, set
   $$\mathcal{I} = \mathcal{I} \cup \{i-1, i, i+1, i+2\}\,.$$
3. Verify convergence

(a) If the problematic indices set $\mathcal{I}$ is empty, exit and output the current values for the control points d_i and tension values $\nu_i,\ i = 0, 1, ..., n$.

(b) If the problematic indices set $\mathcal{I}$ is **not** empty, increase the tension values corresponding to these indices, and go to step 2. In other words,

$$\forall\, i \in \mathcal{I} \quad \nu_i^{(k+1)} = \begin{cases} f\left(\nu_i^{(k)}\right), & i \in \mathcal{I} \\ \nu_i^{(k)}, & i \notin \mathcal{I} \end{cases}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a user-defined increase function, for example $f(x) = 2x$ or $f(x) = x + 1$, and k indicates iteration index.

It should be noted that practical considerations introduce some details to the algorithm. By definition, the quantities $Q_i,\ i = 1, 2, ..., n-1$ can have values either 1, 0 or -1. This is perfectly logical, as they are essentially sign indicators. However, employing the formula for these quantities directly results in different results, varying from -1 to 1, due to numerical errors. For this reason, the value of the quantities Q_i is determined based on a more adaptive approach. We set

$$Q_i = \begin{cases} \operatorname{sign}(\sigma_i), & \Big| |\sigma_i| - 1 \Big| < Q_{tol}, \\ 0, & \text{otherwise}, \end{cases}$$

where Q_{tol} is a user-provided tolerance parameter, and

$$\sigma_i = \frac{P_i \cdot V_i}{\|V_i\|}, \quad i = 1, 2, ..., n-1.$$

Another consideration is the way in which we verify segment convexity. We require that the sign of the product of the curvature and both quantities Q_i and Q_{i+1} be positive in the segment i. Perhaps one of the easiest ways to do this, is to examine the behavior of the function defined by this product numerically. We define

$$C_n(t) = \kappa_g(t; S_i) Q_n, \quad n \in i, i+1, \quad t \in [0, 1]$$

for an index i at which segment convexity should be verified. In order to examine the behavior of $C_n(t)$, we will need to find a root of its derivative, which will give us an extremum for $C_n(t)$. The quantities Q_i are independent on the parameter t as they are defined with reference to the geodesics. Also, the denominator of the geodesic curvature is simply a regularizing positive quantity for $t \in (0, 1)$, hence we will examine only the numerator, for which we have

$$\frac{d}{dt}\left(S(t) \cdot \left[\dot{S}(t) \times \ddot{S}(t)\right]\right) = \dot{S}(t) \cdot \left[\dot{S}(t) \times \ddot{S}(t)\right] + S(t) \cdot \frac{d}{dt}\left[\dot{S}(t) \times \ddot{S}(t)\right] \tag{3.31}$$

The first term amounts to 0, while for the derivative of the cross-product we employ the relative identity and have

$$\begin{aligned} \frac{d}{dt}\left[\dot{S}(t) \times \ddot{S}(t)\right] &= \ddot{S}(t) \times \ddot{S}(t) + \dot{S}(t) \times \dddot{S}(t) \\ &= \dot{S}(t) \times \dddot{S}(t) \end{aligned} \tag{3.32}$$

thus finally

$$\frac{d}{dt}\left(S(t)\cdot\left[\dot{S}(t)\times\ddot{S}(t)\right]\right)=S(t)\cdot\left[\dot{S}(t)\times\dddot{S}(t)\right]. \tag{3.33}$$

Therefore, in order to find an extremum of $C_n(t)$, we need to find a root for the function

$$\dot{C}_n(t)=\left(S(t)\cdot\left[\dot{S}(t)\times\dddot{S}(t)\right]\right)Q_n, \quad n\in\{i,i+1\}, \quad t\in[0,1].$$

We do this by implementing a simple bisection method, and compute an approximation of the root t_* of $\dot{C}_n(t)$. Assuming that the function $C_n(t)$ does not change its monotony on either side of t_*, we compare the signs of $C_n(t)$ at $t=0, t=1$ and $t=t_*$. If the function is positive at all of these parametric values, segment convexity is considered to be satisfied.

Another point which is worth mentioning is that the derivative of the spline curve ought to be continuous in any case. Empirically, this has been found to be false when we use directly the approximation given by the finite difference method. The reason is that the parameter t is essentially arc length, hence the *magnitude* of the vectors computed by the finite difference scheme depends on the length of the segment in which it is computed. The alignment of these vectors is consistent along the path of $S(t)$, but their magnitude is not. A simple solution is to consider one of the segments (for instance, the first) to have a reference (unit) length, and consider the factors

$$N_i^d=\frac{\ell_{ref}}{\ell_i^d}$$

where $\ell_i=\theta(P_i,P_{i+1})$ is the length of the i-th segment, ℓ_{ref} is the length of the reference segment, and d is the order of the derivative computed. Hence, the first derivative of the curve $S(t)$ in the segment i should be

$$\dot{S}_i(t)=N_i^1\cdot \mathit{FiniteDifferenceApproximation},$$

while the second derivative should be

$$\ddot{S}_i(t)=N_i^2\cdot \mathit{FiniteDifferenceApproximation}.$$

Chapter 4

Conclusions

In the previous chapter we have discussed and described a method for computing in an automated way the control points and tension values which produce a piecewise cubic spherical spline that interpolates a given set of points on the sphere and satisfies the criteria for shape preservation as stated in section 3.3. We have seen that when the appropriate tension values increase, the resulting spline satisfies our demands, but there are limitations to what we can do.

We need to draw attention to the fact that in subsection 3.4.2 we saw that the asymptotic limit for the quantity $\dot{S}(t) \times \ddot{S}(t)$ as the corresponding tension values go to infinity, is null at both $t = 0$ and $t = 1$. Since t is our local parameter, the meaning of this limit is that at each interpolation node, for large enough tension values, the above cross-product vanishes, and this is a shortcoming of our method. Ideally, the limit for this quantity would be a non-zero constant, something that could perhaps be achieved by considering a more complex expression, involving normalizing terms which would lead to the desired result. This is a potential course of future enquiry, however it needs to be underlined that in the tests conducted so far the anticipated asymptotic behavior has not been manifested. In all cases presented in the next chapter, convergence was achieved before the quantities of interest could become small enough to be considered nil.

Another aspect we must consider is the formulation of the criteria for shape preservation. The introduction of the geodesic curvature $\kappa_g(t; S)$ allows us to properly formulate the requirements to capture convexity on curved surfaces. In the co-circularity criterion, we need to note that while the criterion is verified when the nodes are not given in the suitable order, the requirement implicitly satisfied the nodal convexity criterion, which is a desired trait of the algorithm. Of more interest is the convexity criterion regarding a whole segment, where again geodesic curvature plays a major role. It has also helped us see that when the tension values affecting the spline in the segment $[P_i, P_{i+1}]$ increase, the spline in this segment approaches the geodesic arc between the nodes P_i and P_{i+1}. These and other observations we will see in the results of the next section.

Chapter 5

Results

The code implementing the algorithm described in the previous section has been tested for 20 input cases. Note that the limitation that the maximum geodesic distance between consecutive nodes must not exceed $\pi/2$ is not respected in all cases, for illustrative purposes. In the following pages we present the results for every test case. The parameters used for all cases are the same, excluding the set of input points. The values of the parameters used are the following.

Parameter	**Value**
B_{tol}	`1e-10`
$B_{N_{\max}}$	`100`
S_{tol}	`1e-10`
$S_{N_{\max}}$	`1000`
ε	`1e-3`
h	`1e-3`

Also, for every case the tension value increase function has been defined as

$$f(\nu_i) = \nu_i + 10$$

For each one of the test cases, we have included the input points passed to the program, as well as plots representing the unit sphere, the geodesics between the input points (in red dashed curves), and the spline resulting from the algorithm (purple solid curve). For every case, the binormal vectors $V_i,\ i = 1, 2, ..., n-1$ are also included in the figures. In each figure, the starting node P_0 is marked with a black x-mark in order to facilitate the reader's orientation on the curve. Whenever the segment convexity criterion needs to be verified, the "fan" of vectors $\dot{S}(t) \times \ddot{S}(t),\ t \in [0, 1]$ is included in the figure (red when the criterion is not satisfied in the corresponding segment, green when the criterion is satisfied). In these cases, we have also included plots which demonstrate the evolution of the quantities $\left[\dot{S}(t) \times \ddot{S}(t)\right] \cdot Q_\kappa,\ \kappa \in \{i, i+1\},\ t \in [0, 1]$ for the intervals of interest. The segment

$[P_i, P_{i+1}]$, $i = 0, 1, ..., n-1$ is called for brevity segment i.

The following table summarizes the problems exhibited by each case. The increase of the appropriate tension values has led to the satisfaction of the requirements which were not met initially (with all tension values equal to 0). In some cases, all requirements were fulfilled with the initial tension values all set to 0. In these cases we have not only G^2, but C^2-continuity at the interpolation nodes. We draw attention to the particular interest of each case in the corresponding section.

Case examined	**Co-circularity**	**Nodal convexity**	**Segment convexity**
Case 1	•		
Case 2			
Case 3		•	•
Case 4	•		
Case 5			
Case 6			
Case 7		•	•
Case 8			
Case 9			
Case 10			
Case 11		•	•
Case 12			
Case 13			
Case 14		•	
Case 15			
Case 16			
Case 17		•	
Case 18			
Case 19		•	
Case 20			

5.1 Case 1

This case requires co-circularity to be satisfied at nodes P_1 and P_3, hence the tension values increased are $\{0, 1, 2, 3, 4\}$. As we can see from the final tension values, co-circularity is validated for node P_1 before it is validated for node P_3. Also, note than the last and first node coincide, however this does not affect the form of the spline.

X	Y	Z
1.000000,	0.000000,	0.000000
0.707100,	0.707100,	0.000000
0.000000,	1.000000,	0.000000
0.000000,	0.000000,	1.000000
0.000000,	-0.879700,	0.475500
1.000000,	0.000000,	0.000000

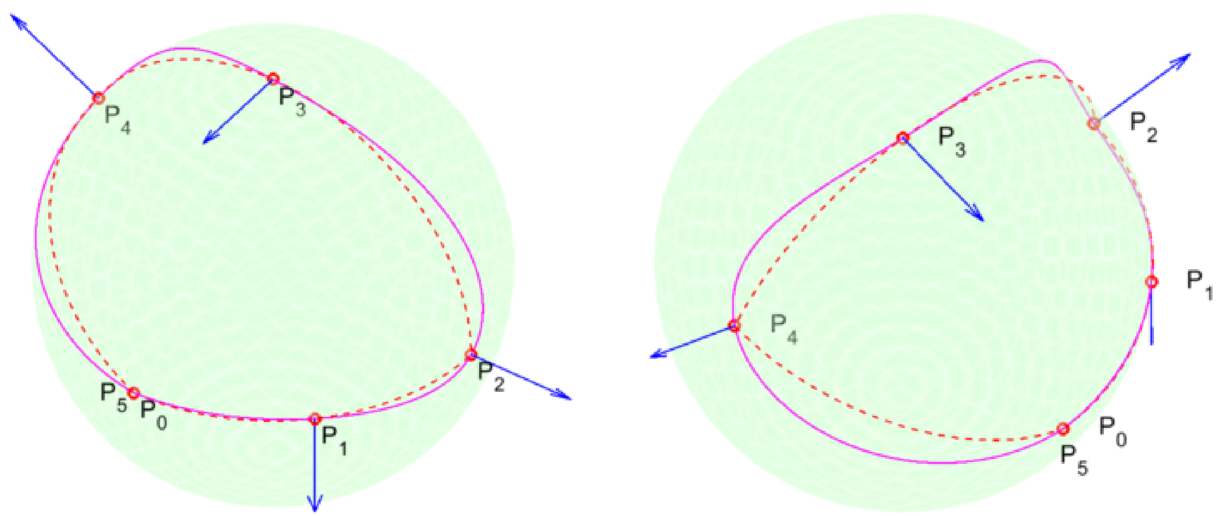


Figure 5.1: *Case 1 initial setting on the sphere from different viewpoints. All tension values are equal to zero.*

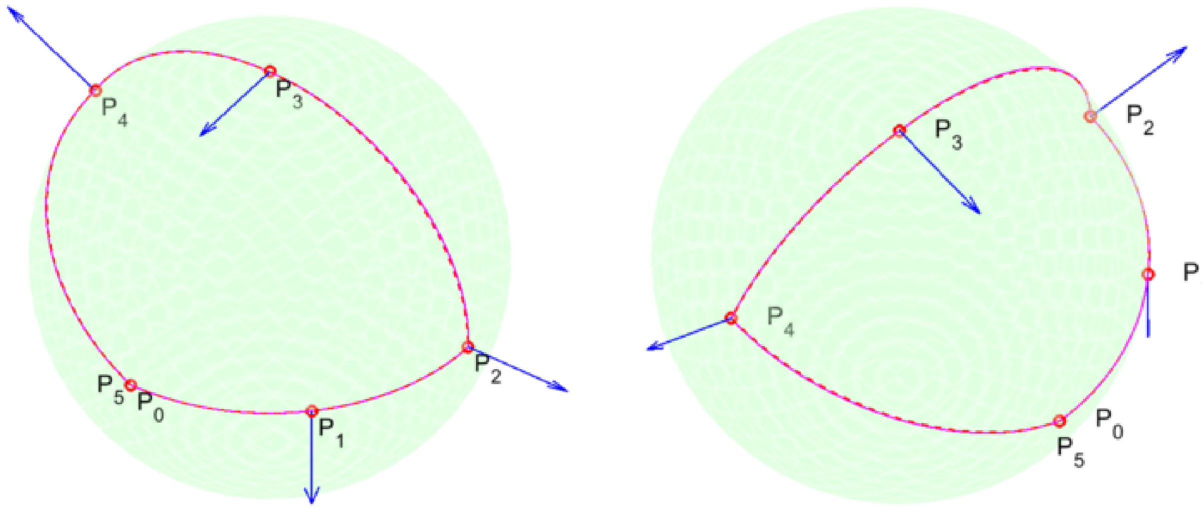
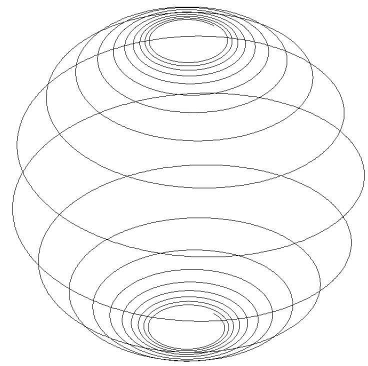


Figure 5.2: *Case 1 final setting on the sphere from different viewpoints. Tension values are* $\{220, 220, 260, 260, 260, 0\}$.

5.2 Case 2

The input nodes to this case are points (sparsely) sampled from the Spherical Spiral. The parametric equations for the Spherical Spiral are



$$
\begin{aligned}
x &= \cos(t)\cos(c) \\
y &= \sin(t)\cos(c) \\
z &= -\sin(c)
\end{aligned}
$$

with $c = \tan^{-1}(\alpha t)$, where α is a given constant. This curve describes the trajectory of a ship traveling from the south to the north pole of a sphere while maintaining a fixed (but not right) angle with the meridians. An example is given on the figure to the right with $\alpha = 0.075$.

```
     X           Y           Z
-------------------------------
 0.433084,  0.200366,  0.878801
-0.678528,  0.228623,  0.698092
 0.213169, -0.767765,  0.604232
 0.569245,  0.670167,  0.476274
-0.945461, -0.102825, -0.309079
 0.569245, -0.670167, -0.476274
 0.213169,  0.767765, -0.604232
-0.678528, -0.228623, -0.698092
 0.511748, -0.389021, -0.766014
```

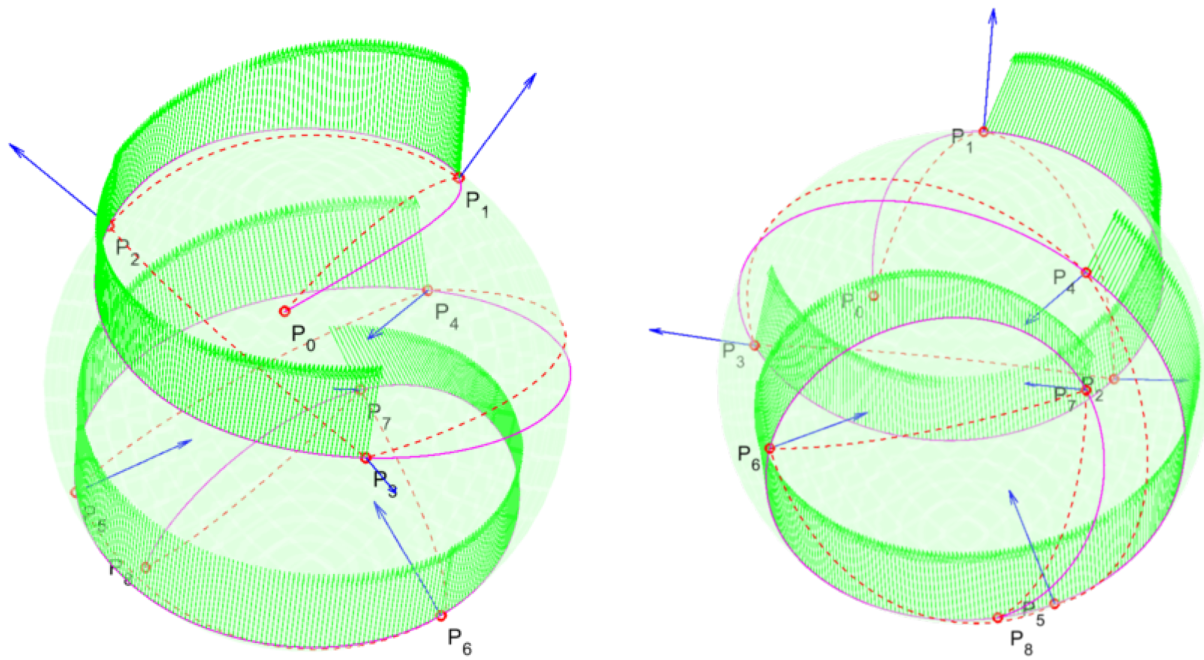
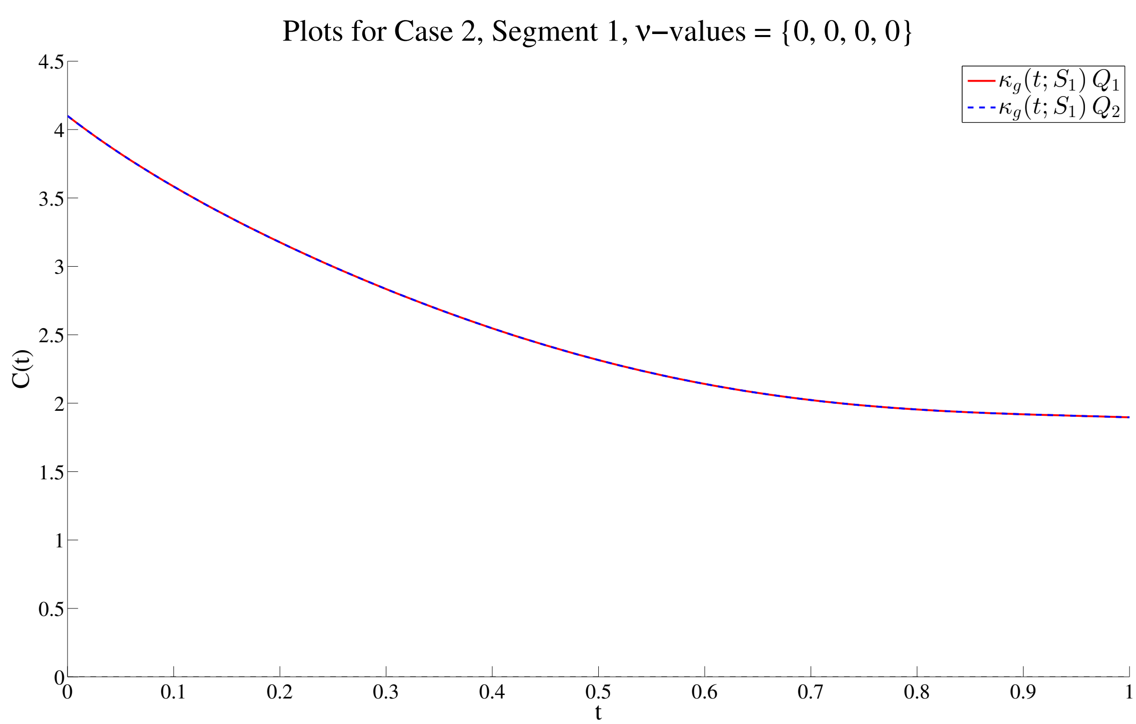
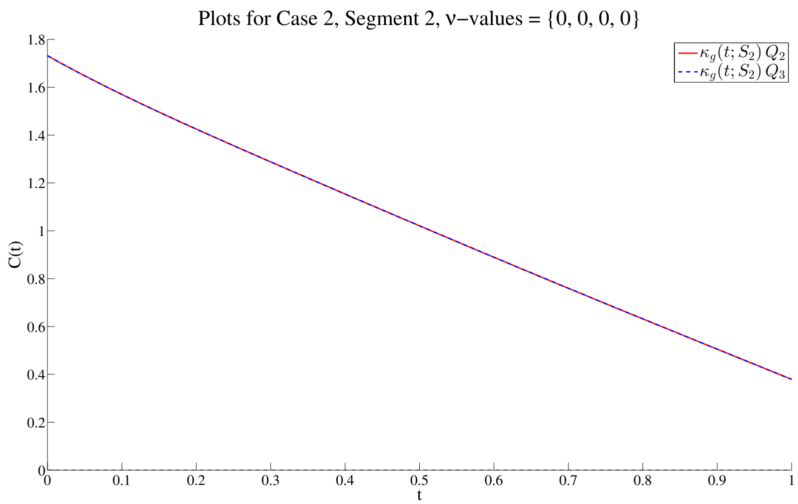
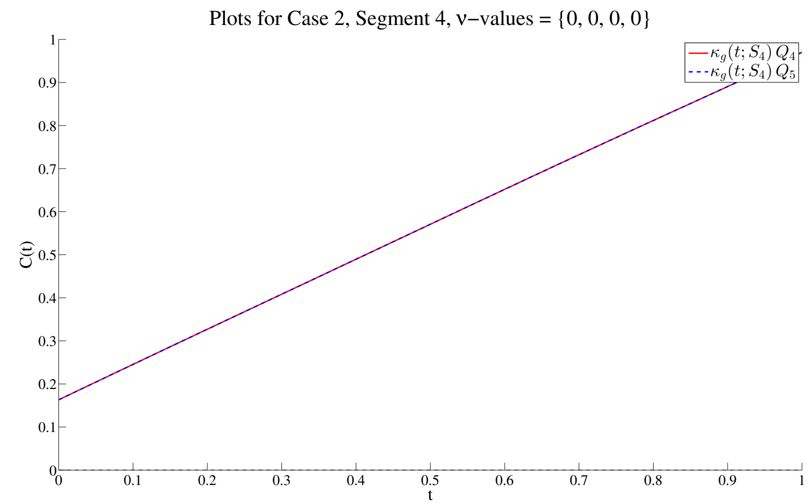
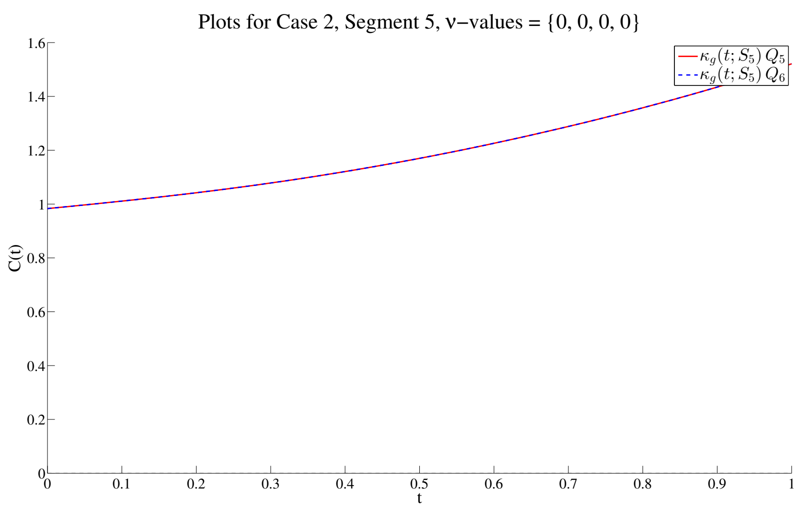


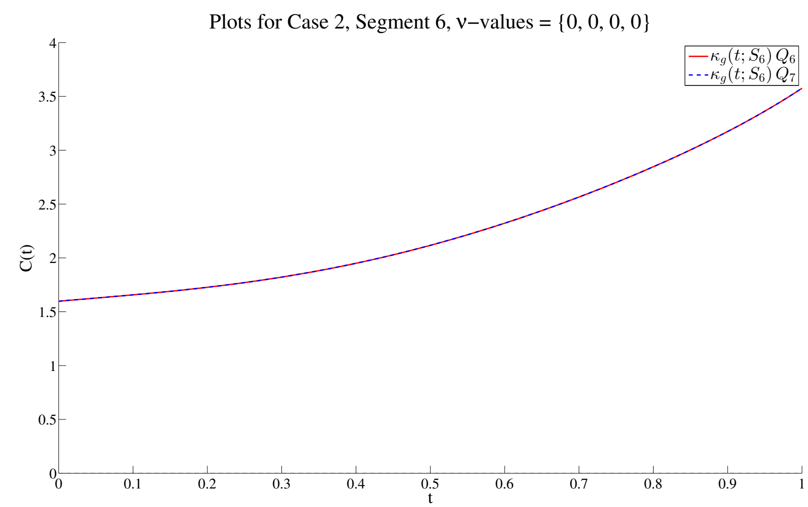
Figure 5.3: *Case 2 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*




Plots for Case 2, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_2)\, Q_2$
$\kappa_g(t; S_2)\, Q_3$
C(t)
t


Plots for Case 2, Segment 4, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_4)\, Q_4$
$\kappa_g(t; S_4)\, Q_5$
C(t)
t


Plots for Case 2, Segment 5, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_5)\, Q_5$
$\kappa_g(t; S_5)\, Q_6$
C(t)
t


Plots for Case 2, Segment 6, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_6)\, Q_6$
$\kappa_g(t; S_6)\, Q_7$
C(t)
t

5.3 Case 3

This case requires segment convexity to be verified on two neighboring segments. We can see that in the initial setting the fan of the vectors of the cross-product are not aligned with the binormal at P_3, but when the appropriate tension values reach high enough values, segment convexity is satisfied.

```
      X          Y          Z
--------------------------------
 1.000000,  0.000000,  0.000000
 0.653600,  0.270600,  0.706800
 0.500400,  0.500000,  0.706800
 0.000400,  0.500200,  0.865900
-0.924000,  0.001500,  0.382500
 0.000800, -0.980800,  0.195000
```

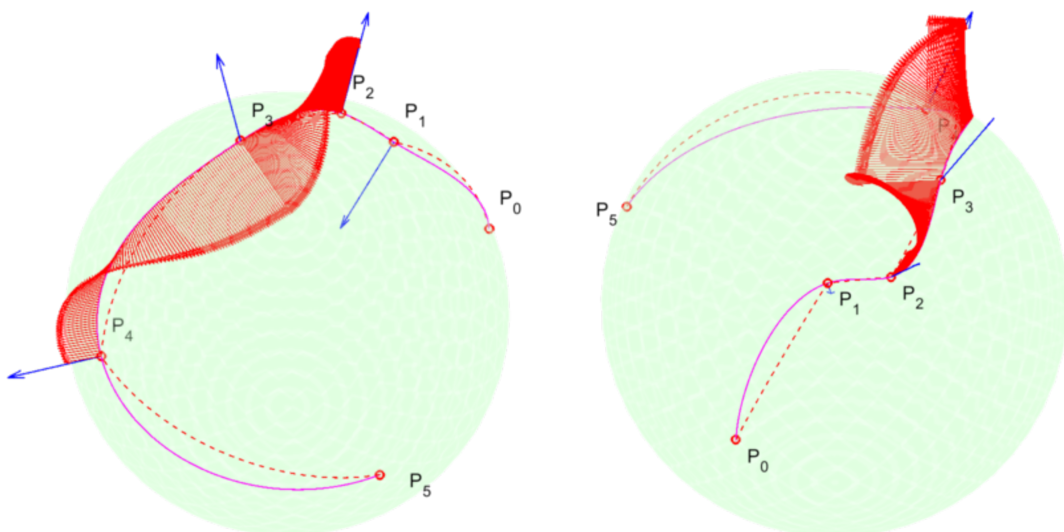


Figure 5.4: *Case 3 initial setting on the sphere from different viewpoints. All tension values are equal to 0.*

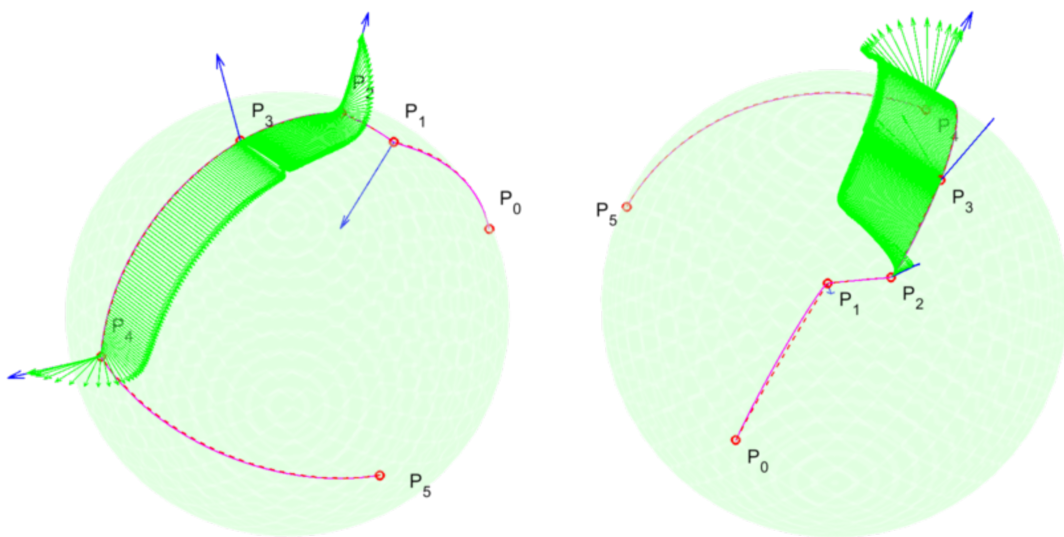
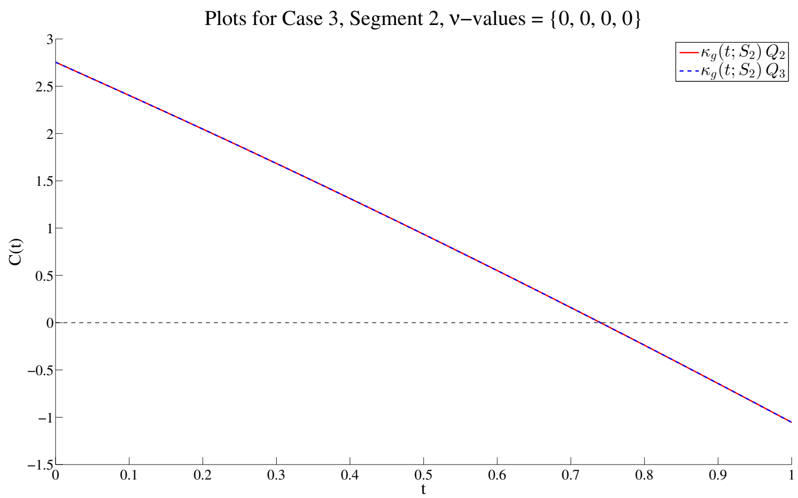
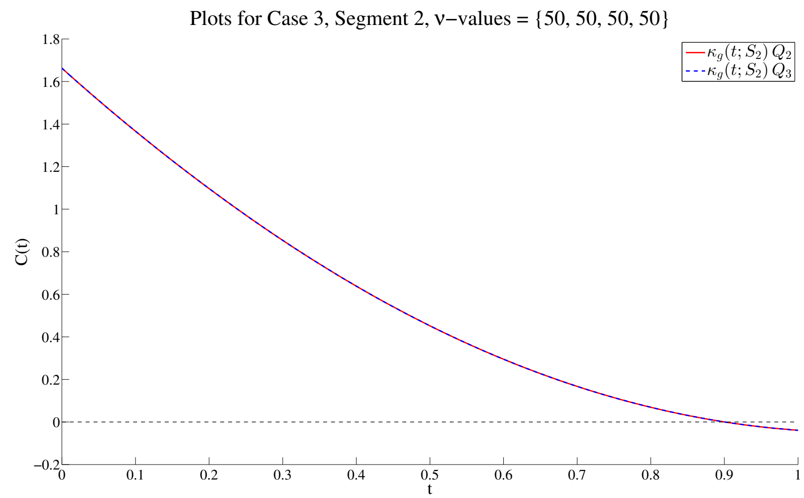
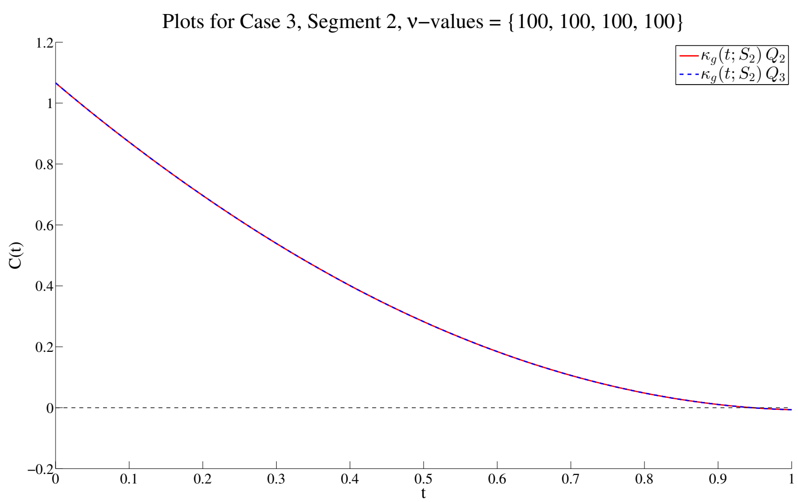
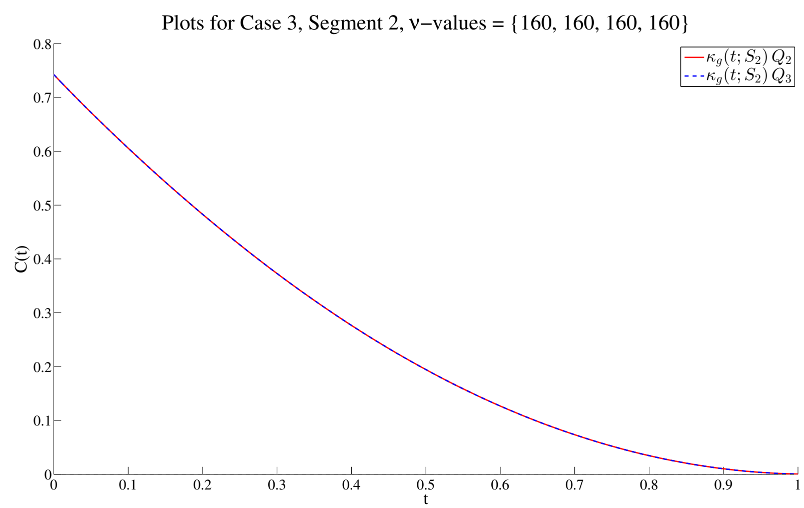


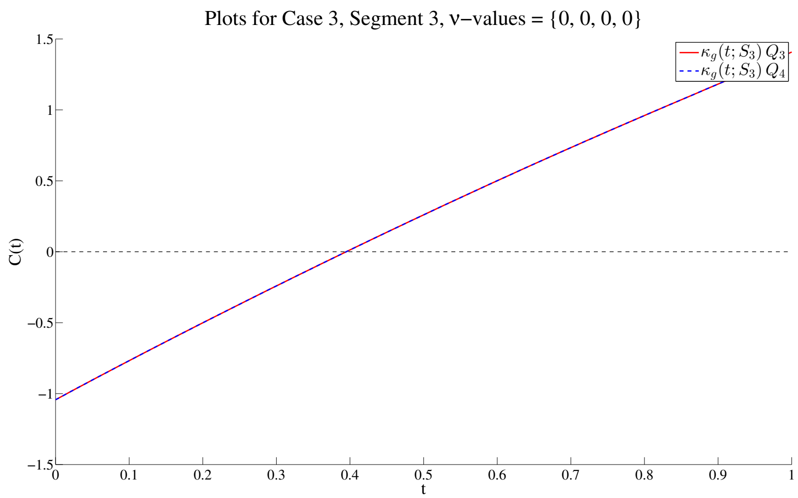
Figure 5.5: *Case 3 final setting on the sphere from different viewpoints. Tension values are* $\{0, 160, 160, 160, 160, 160\}$.

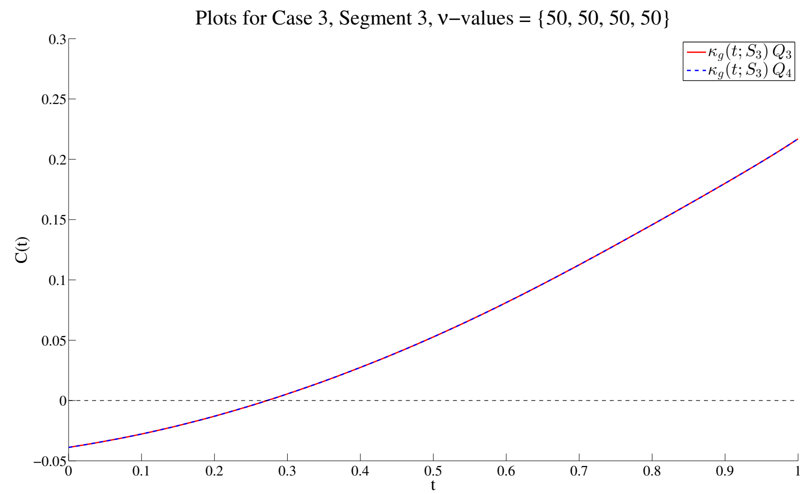

Plots for Case 3, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_2)\,Q_2$
$\kappa_g(t;S_2)\,Q_3$
C(t)
t

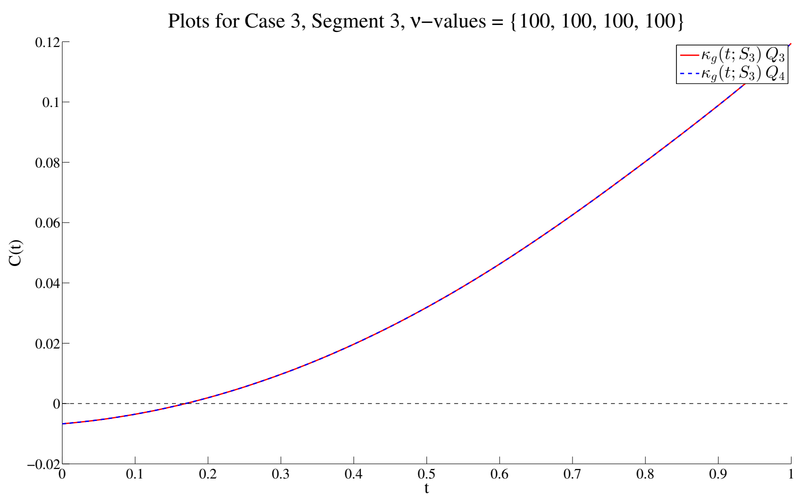

Plots for Case 3, Segment 2, ν−values = {50, 50, 50, 50}
$\kappa_g(t;S_2)\,Q_2$
$\kappa_g(t;S_2)\,Q_3$
C(t)
t

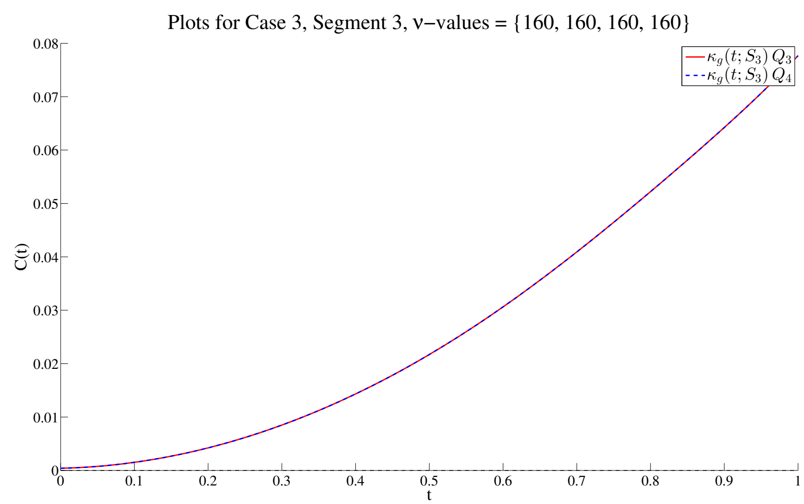

Plots for Case 3, Segment 2, ν−values = {100, 100, 100, 100}
$\kappa_g(t;S_2)\,Q_2$
$\kappa_g(t;S_2)\,Q_3$
C(t)
t


Plots for Case 3, Segment 2, ν−values = {160, 160, 160, 160}
$\kappa_g(t;S_2)\,Q_2$
$\kappa_g(t;S_2)\,Q_3$
C(t)
t


Plots for Case 3, Segment 3, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_3)\,Q_3$
$\kappa_g(t;S_3)\,Q_4$
C(t)
t


Plots for Case 3, Segment 3, ν−values = {50, 50, 50, 50}
$\kappa_g(t;S_3)\,Q_3$
$\kappa_g(t;S_3)\,Q_4$
C(t)
t


Plots for Case 3, Segment 3, ν−values = {100, 100, 100, 100}
$\kappa_g(t;S_3)\,Q_3$
$\kappa_g(t;S_3)\,Q_4$
C(t)
t


Plots for Case 3, Segment 3, ν−values = {160, 160, 160, 160}
$\kappa_g(t;S_3)\,Q_3$
$\kappa_g(t;S_3)\,Q_4$
C(t)
t

5.4 Case 4

In this case, segment convexity is required for Segment 2, and co-circularity is requested ad node P_1. Within three iterations, the requirements regarding co-circularity are satisfied – convexity is valid with zero tension values.

```
    X          Y          Z
-------------------------------
 0.587785,  0.000000, -0.809017
 0.809017, -0.000000, -0.587785
 0.809017, -0.000000,  0.587785
-0.654509,  0.475528,  0.587785
 0.250000,  0.181636, -0.951057
```

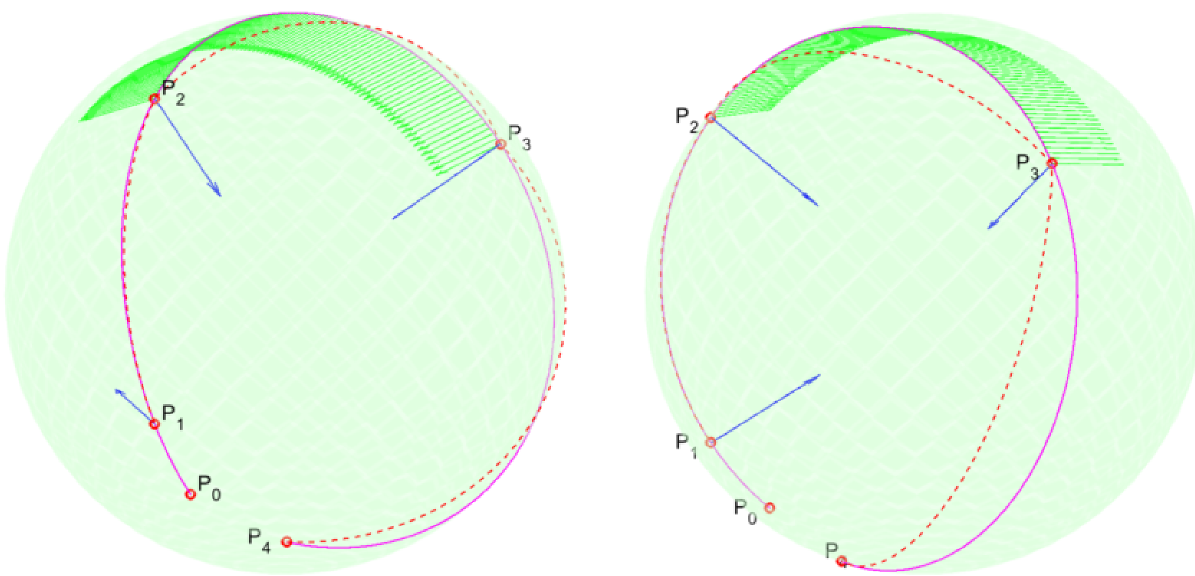


Figure 5.6: *Case 3 initial setting on the sphere from different viewpoints. All tension values are equal to 0.*

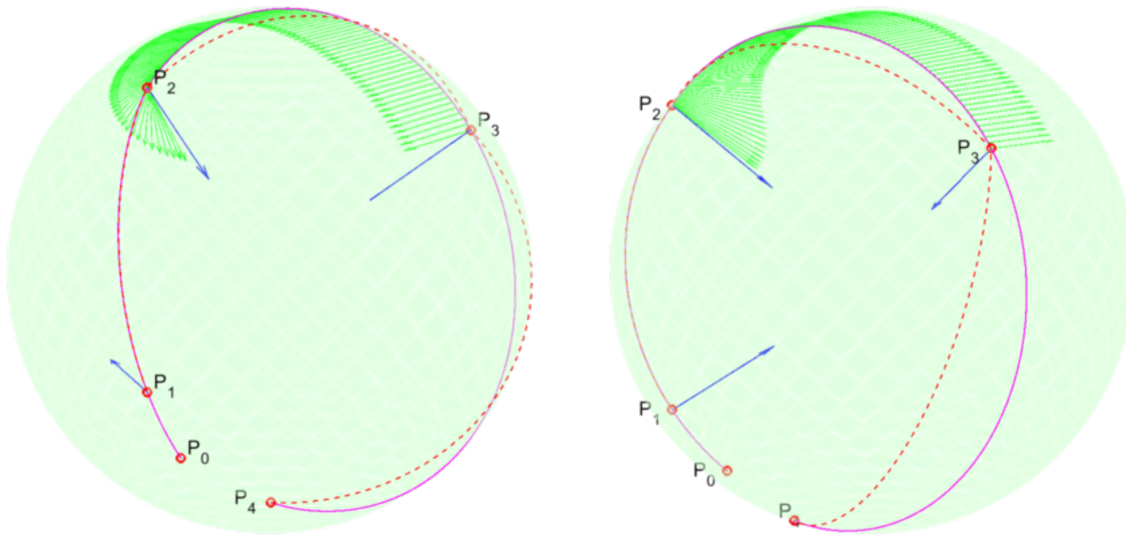
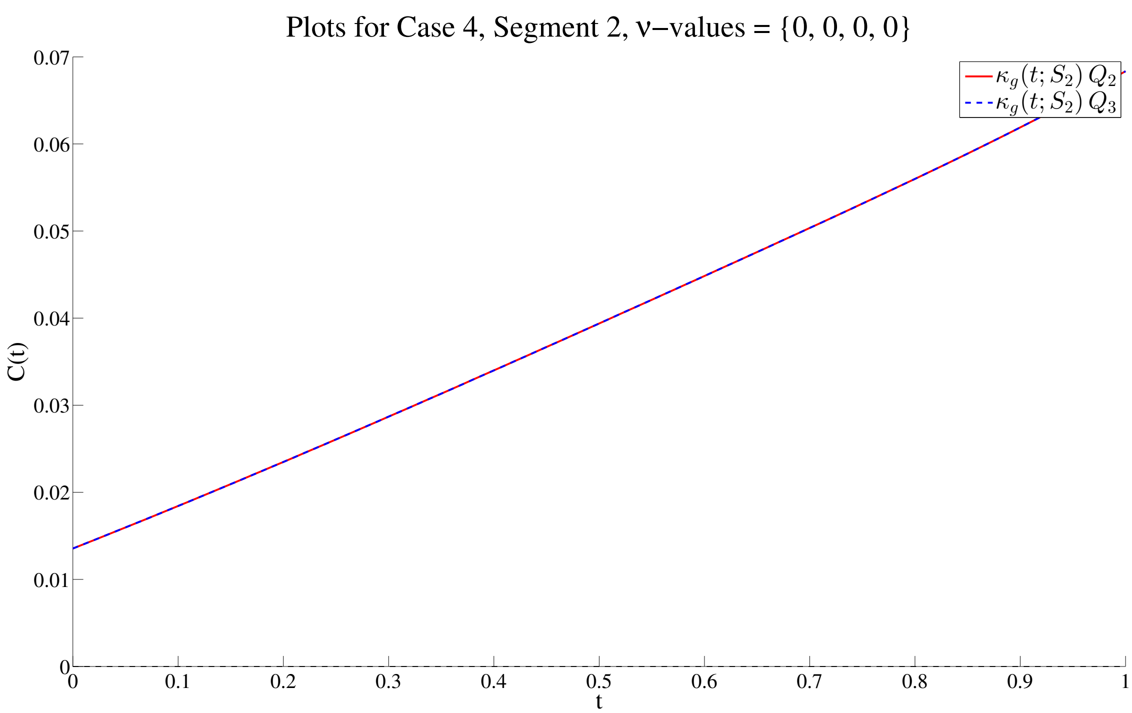
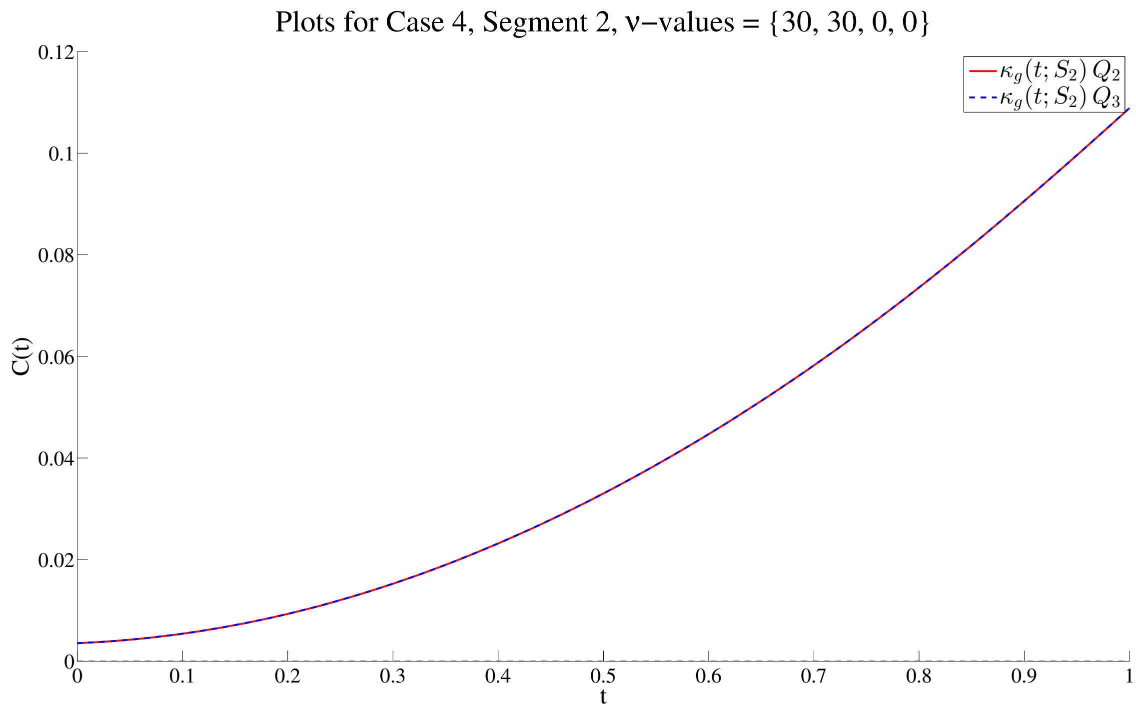


Figure 5.7: *Case 4 final setting on the sphere from different viewpoints. Tension values are* $\{30, 30, 30, 0, 0\}$.





Plots for Case 4, Segment 2, ν−values = {30, 30, 0, 0}
$\kappa_g(t; S_2)\, Q_2$
$\kappa_g(t; S_2)\, Q_3$
C(t)
t
0.12
0.1
0.08
0.06
0.04
0.02
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

5.5 Case 5

This is another example of points sampled from the Spherical Spiral. In this case, however, we have sampled points closer to the equator, avoiding the poles. As can be seen from the figure, many segments require convexity to be verified, and the criterion is indeed satisfied for every segment. Note how the points appear collinear, but are actually not – they do not lie on *geodesics*.

```
     X          Y          Z
------------------------------
 0.828900,  0.215100,  0.516300
 0.555000,  0.683100,  0.474800
 0.043000,  0.901900,  0.429800
-0.512100,  0.769700,  0.381400
-0.887200,  0.322900,  0.329600
-0.922600, -0.270900,  0.274600
-0.591200, -0.776800,  0.216800
-0.015700, -0.987500,  0.156700
 0.577400, -0.810900,  0.094800
 0.949600, -0.311900,  0.031700
 0.949600,  0.311900, -0.031700
 0.577400,  0.810900, -0.094800
-0.015700,  0.987500, -0.156700
-0.591200,  0.776800, -0.216800
-0.922600,  0.270900, -0.274600
-0.887200, -0.322900, -0.329600
-0.512100, -0.769700, -0.381400
 0.043000, -0.901900, -0.429800
 0.555000, -0.683100, -0.474800
 0.828900, -0.215100, -0.516300
```

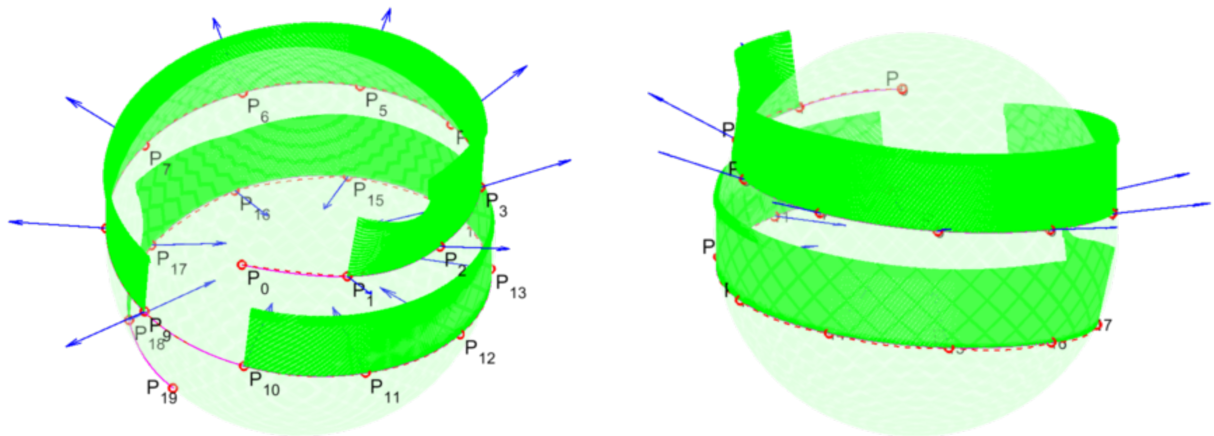
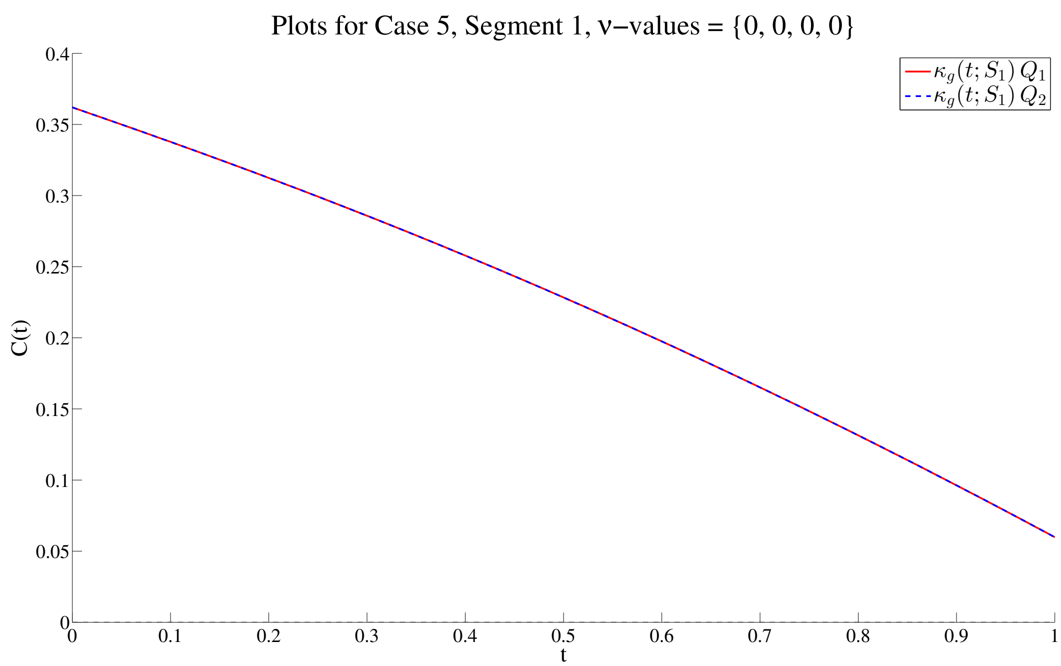
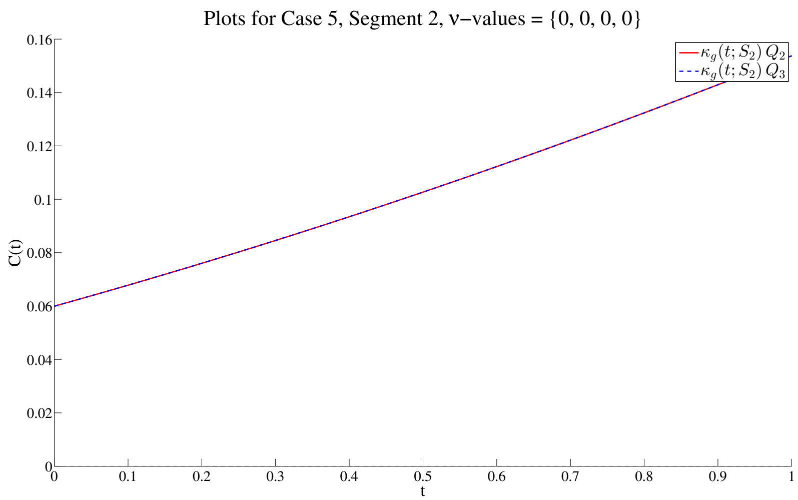
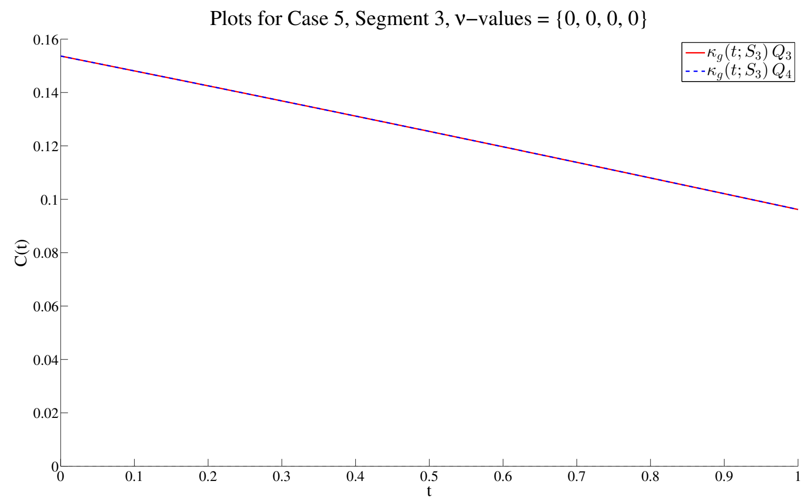
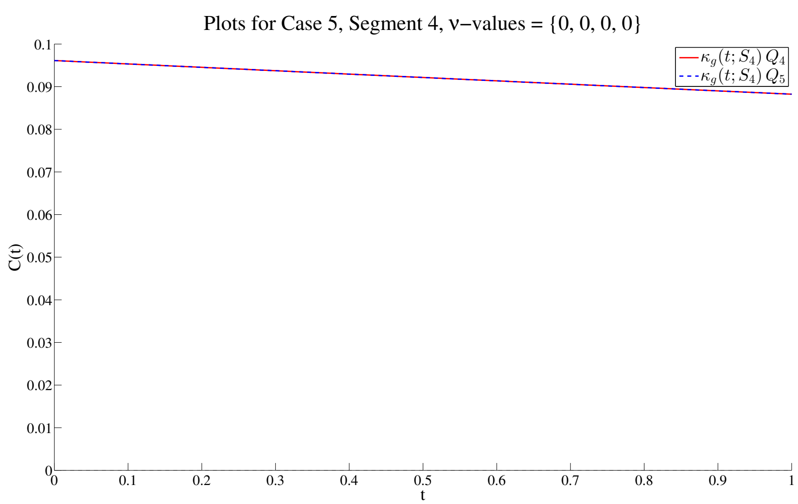


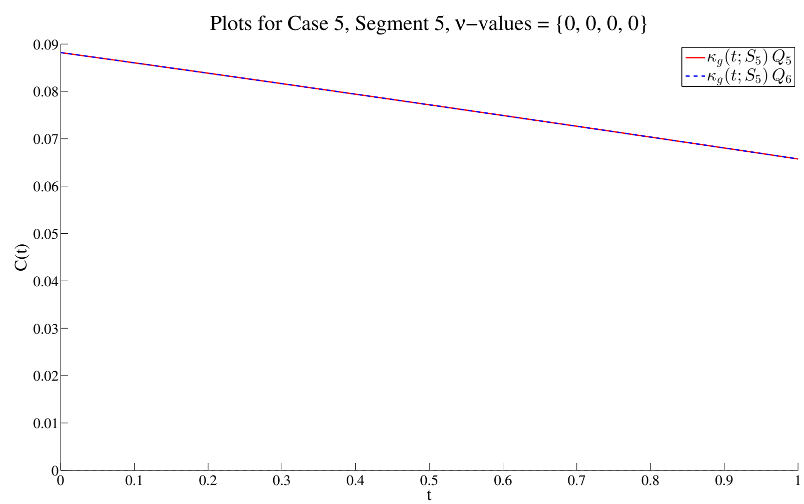
Figure 5.8: *Case 5 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*

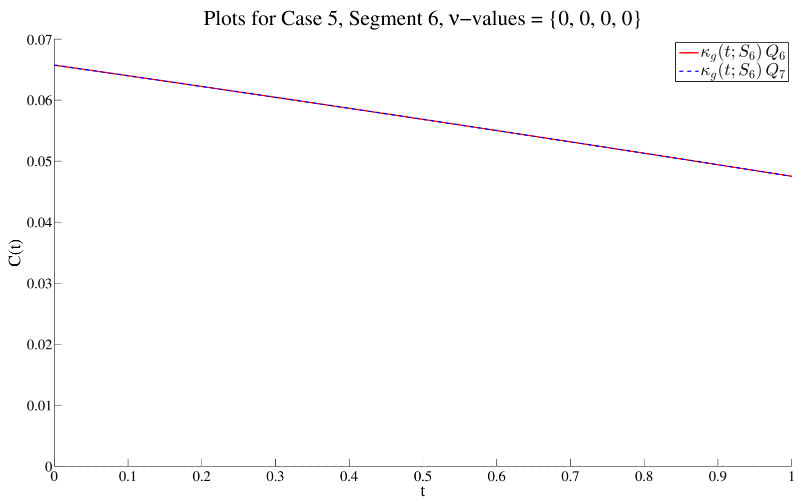


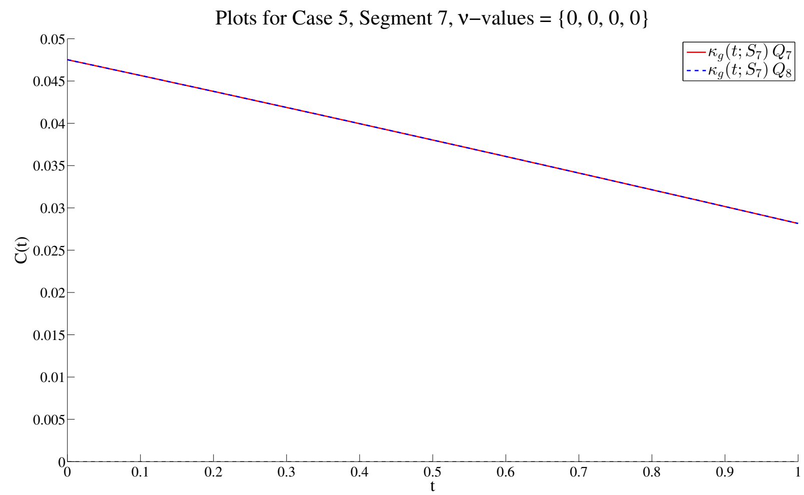

Plots for Case 5, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_2)\,Q_2$
$\kappa_g(t;S_2)\,Q_3$
C(t)
t

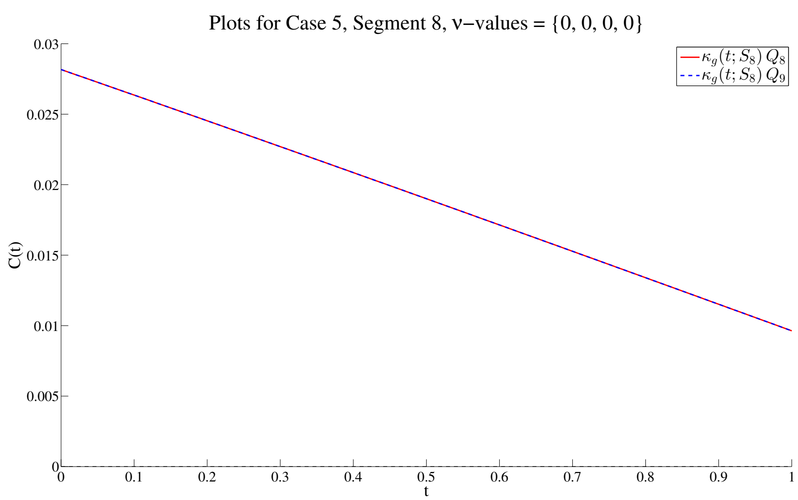

Plots for Case 5, Segment 3, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_3)\,Q_3$
$\kappa_g(t;S_3)\,Q_4$
C(t)
t

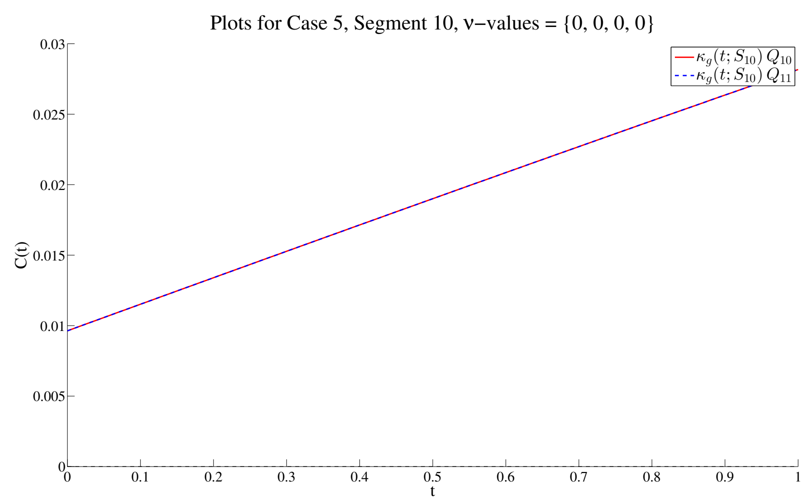

Plots for Case 5, Segment 4, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_4)\,Q_4$
$\kappa_g(t;S_4)\,Q_5$
C(t)
t

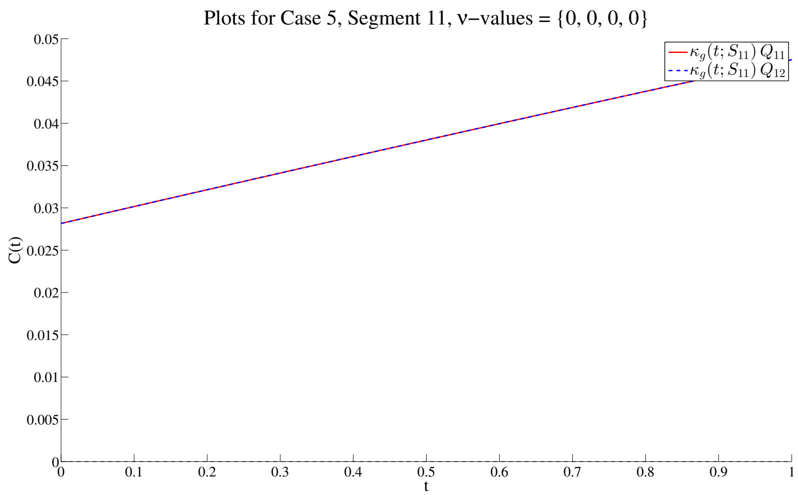

Plots for Case 5, Segment 5, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_5)\,Q_5$
$\kappa_g(t;S_5)\,Q_6$
C(t)
t

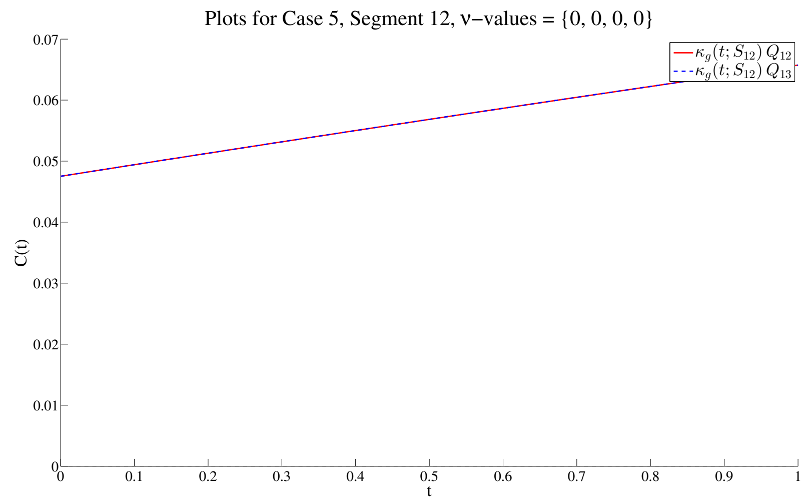

Plots for Case 5, Segment 6, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_6)\,Q_6$
$\kappa_g(t;S_6)\,Q_7$
C(t)
t

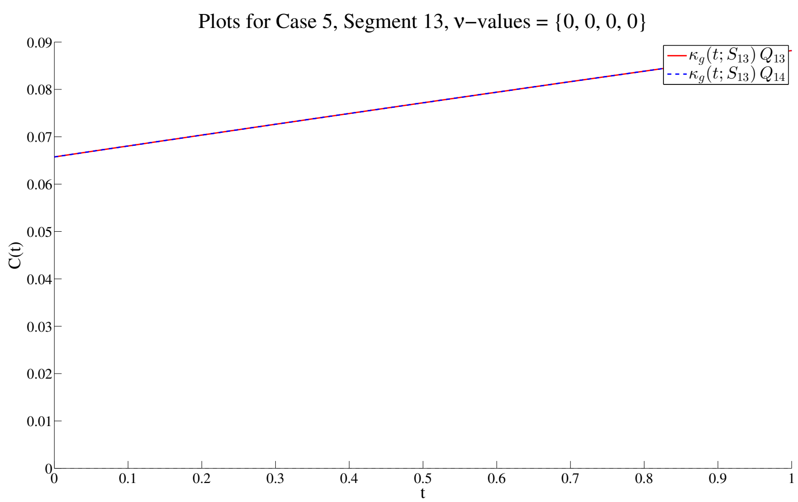

Plots for Case 5, Segment 7, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_7)\,Q_7$
$\kappa_g(t;S_7)\,Q_8$
C(t)
t

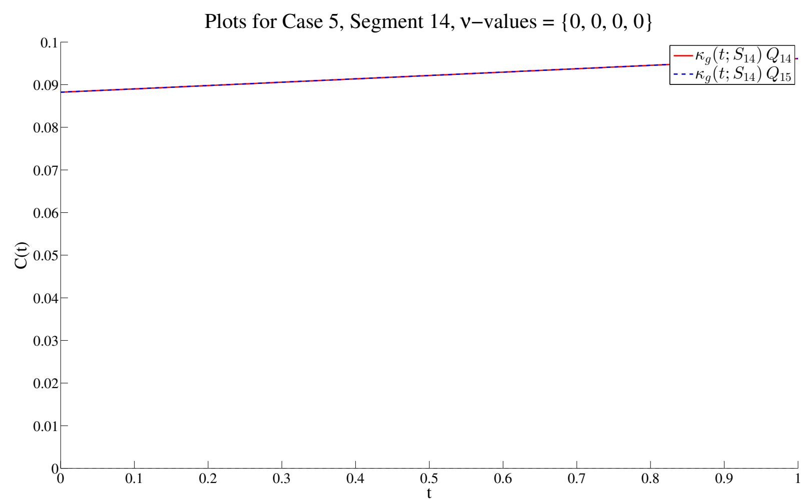

Plots for Case 5, Segment 8, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_8)\,Q_8$
$\kappa_g(t;S_8)\,Q_9$
C(t)
t

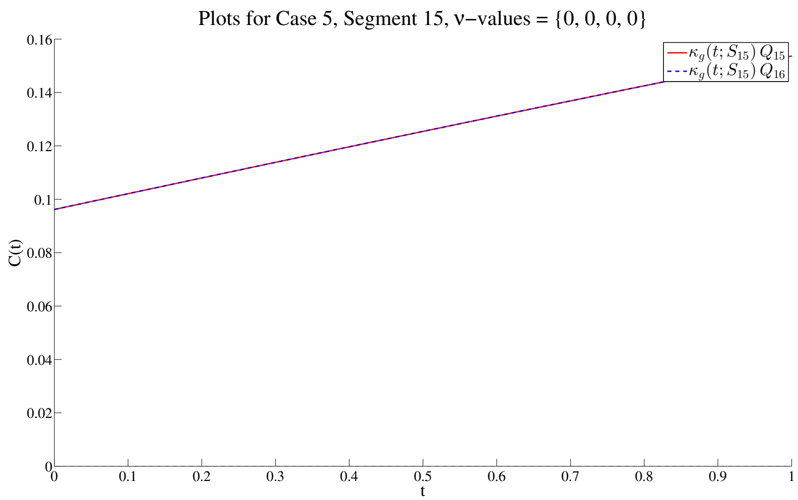

Plots for Case 5, Segment 10, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_{10})\,Q_{10}$
$\kappa_g(t;S_{10})\,Q_{11}$
C(t)
t

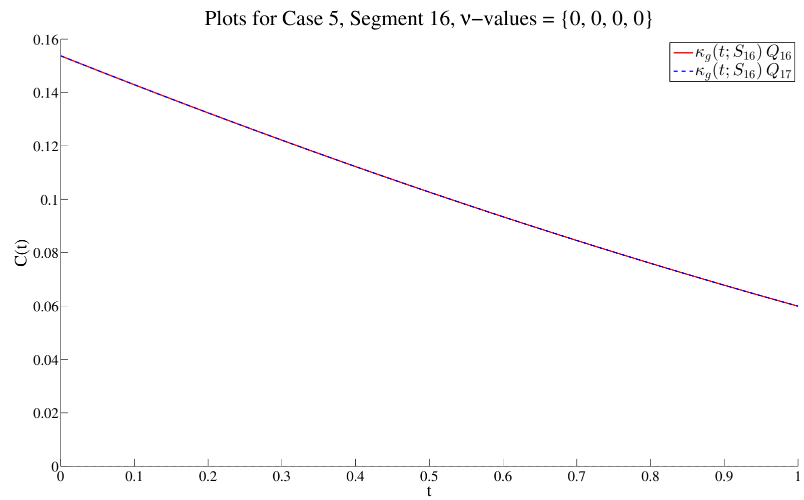

Plots for Case 5, Segment 11, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_{11})\, Q_{11}$
$\kappa_g(t; S_{11})\, Q_{12}$
C(t)
t

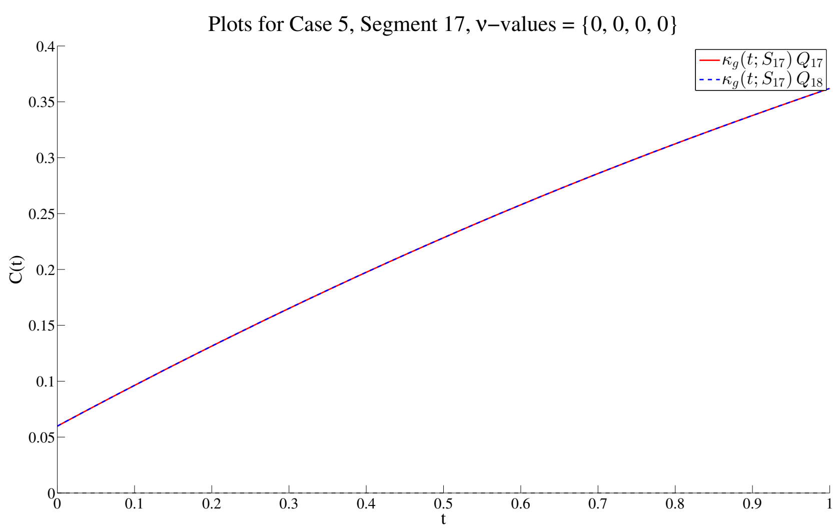

Plots for Case 5, Segment 12, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_{12})\, Q_{12}$
$\kappa_g(t; S_{12})\, Q_{13}$
C(t)
t


Plots for Case 5, Segment 13, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_{13})\, Q_{13}$
$\kappa_g(t; S_{13})\, Q_{14}$
C(t)
t


Plots for Case 5, Segment 14, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_{14})\, Q_{14}$
$\kappa_g(t; S_{14})\, Q_{15}$
C(t)
t


Plots for Case 5, Segment 15, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_{15})\,Q_{15}$
$\kappa_g(t;S_{15})\,Q_{16}$
C(t)
t
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Plots for Case 5, Segment 16, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_{16})\,Q_{16}$
$\kappa_g(t;S_{16})\,Q_{17}$
C(t)
t
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Plots for Case 5, Segment 17, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_{17})\,Q_{17}$
$\kappa_g(t;S_{17})\,Q_{18}$
C(t)
t
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

5.6 Case 6

In this generic case we can see that all criteria are satisfied with all tension values set to 0, hence the resulting curve is C^2-continuous. Note how the default behavior of the ν-spline satisfies the node convexity criterion at nodes P_2 and P_4. Also note that no two consecutive binormals have the same sign, thus eliminating the segment convexity requirement.

```
    X          Y          Z
-------------------------------
 0.721375,  0.366154, -0.587835
 0.538720,  0.540781, -0.646016
-0.813290,  0.015269, -0.581658
-0.177499,  0.940630,  0.289325
 0.373575, -0.314647,  0.872605
 0.597205,  0.057249, -0.800043
```

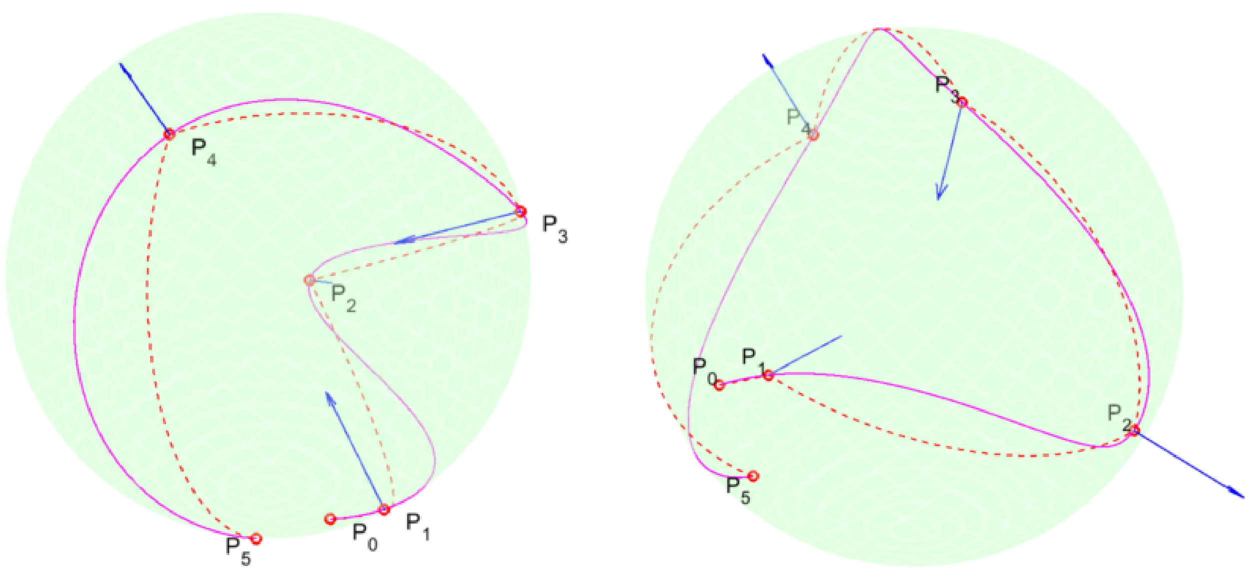


Figure 5.9: *Case 6 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*

5.7 Case 7

In this case we can see that segment convexity is requested for four consecutive segments. It is verified for two of them, but not for the other two. A single increase of the appropriate tension values is sufficient to cause the spline to bend, satisfying the convexity requirements for the problematic segments.

```
     X          Y          Z
-------------------------------
 0.561370,  0.357729,  0.746253
 0.455491,  0.888784, -0.050908
-0.631673,  0.774457, -0.034716
 0.394154, -0.617118,  0.681034
 0.094836,  0.902925,  0.419204
-0.634718,  0.352093,  0.687869
-0.540772, -0.078641,  0.837485
```

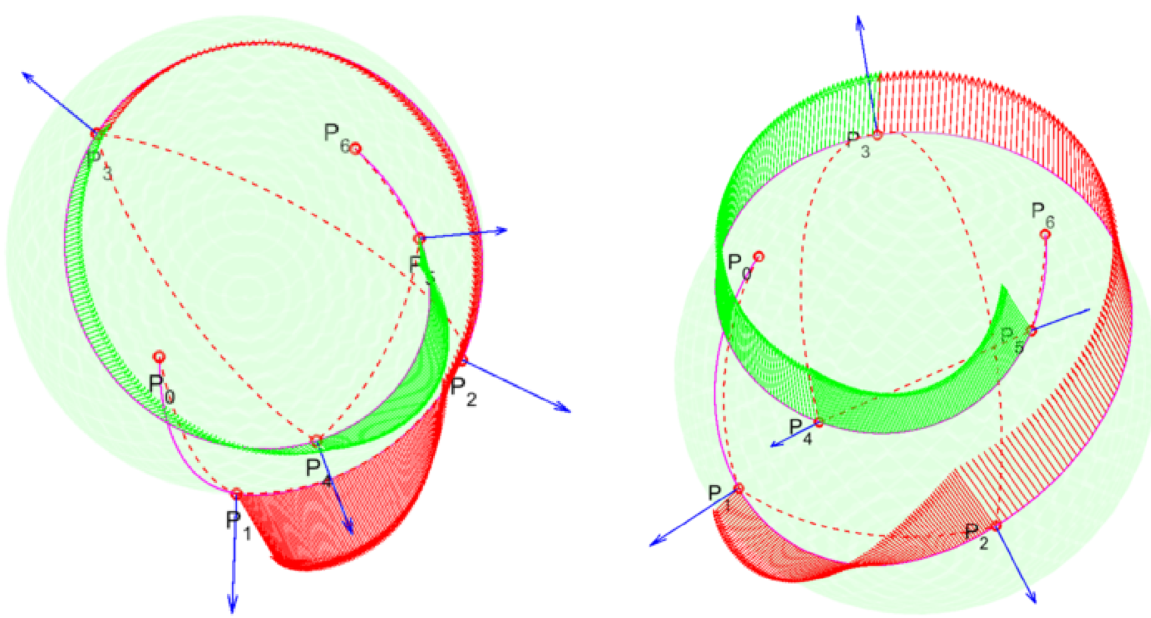


Figure 5.10: *Case initial setting on the sphere from different viewpoints. All tension values are equal to zero.*

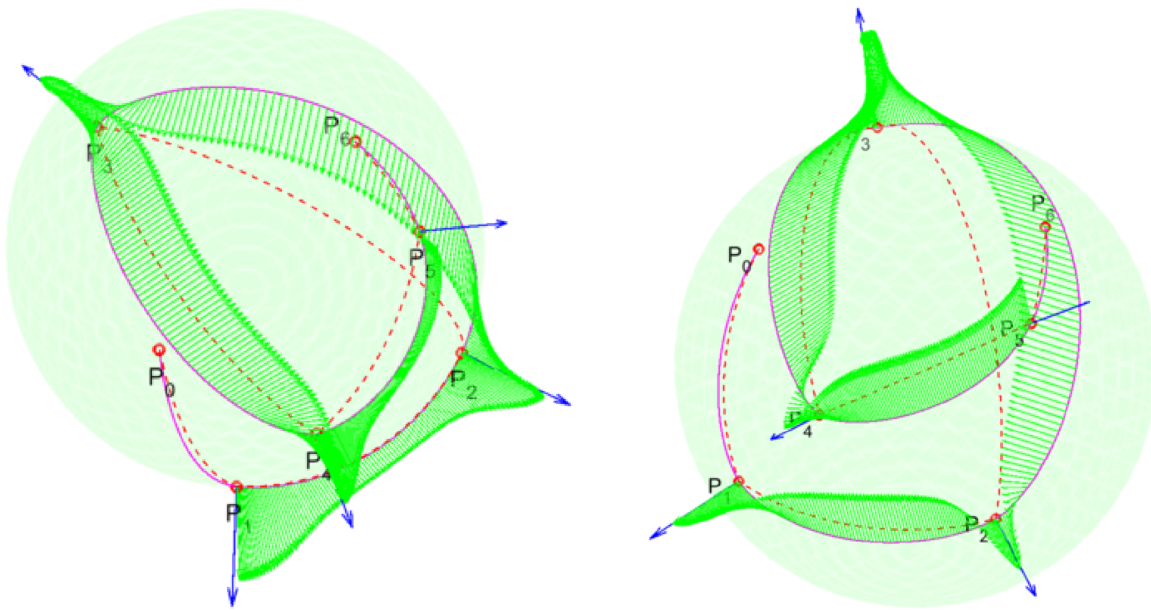
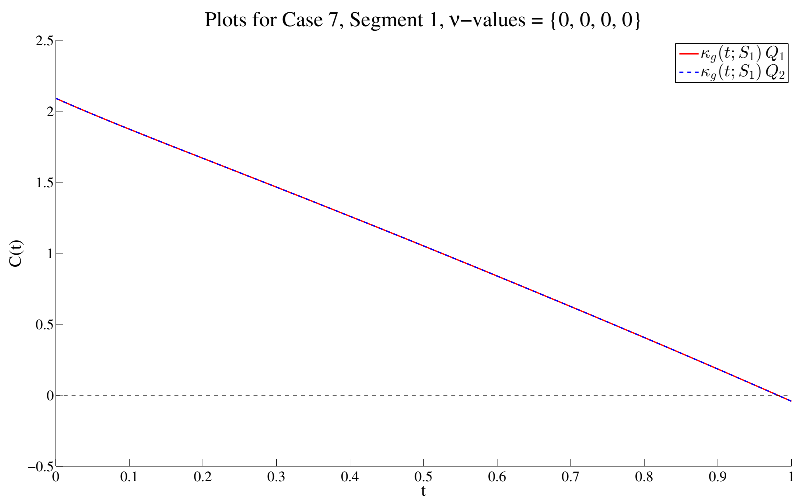
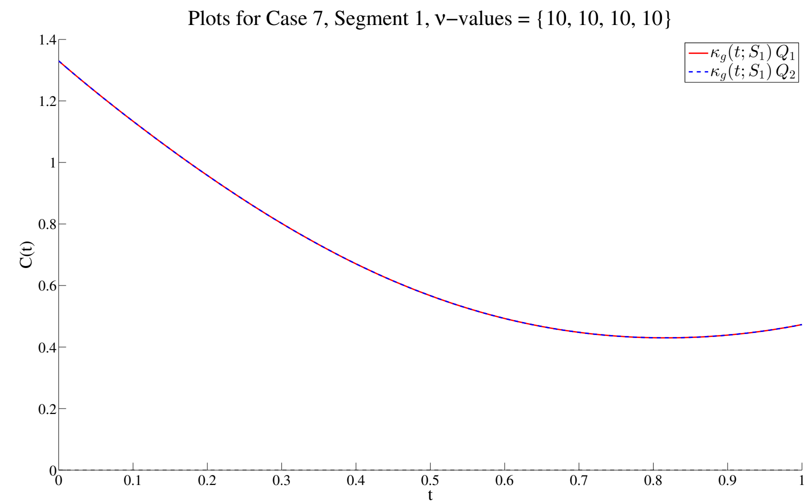
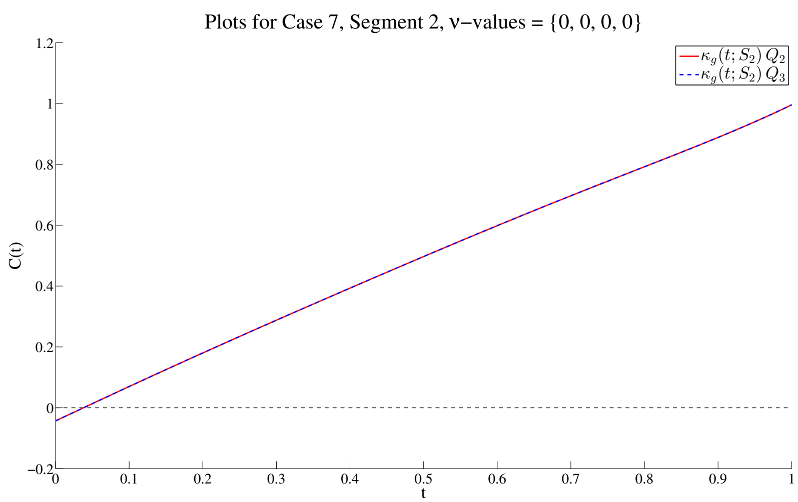
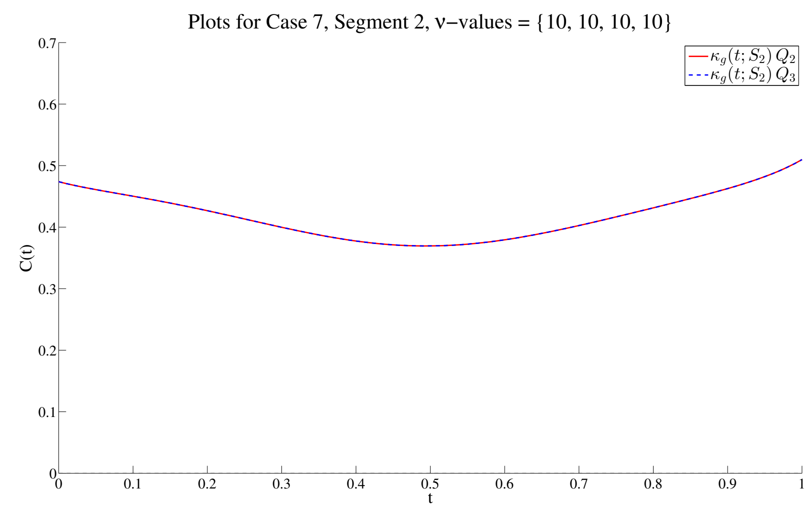


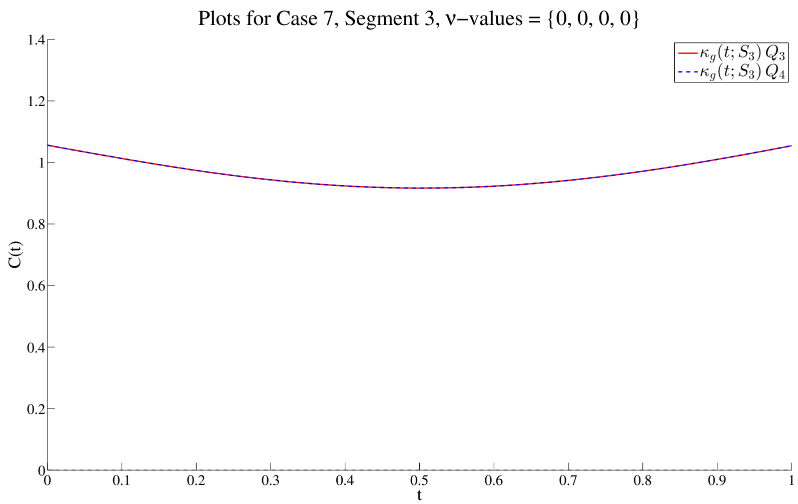
Figure 5.11: *Case 7 final setting on the sphere from different viewpoints. Tension values are* $\{10, 10, 10, 10, 10, 0, 0\}$.

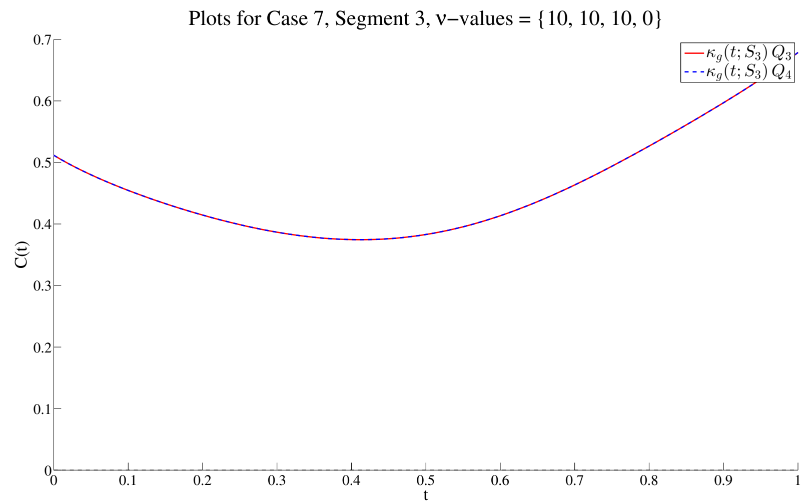

Plots for Case 7, Segment 1, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_1)\,Q_1$
$\kappa_g(t; S_1)\,Q_2$
C(t)
t

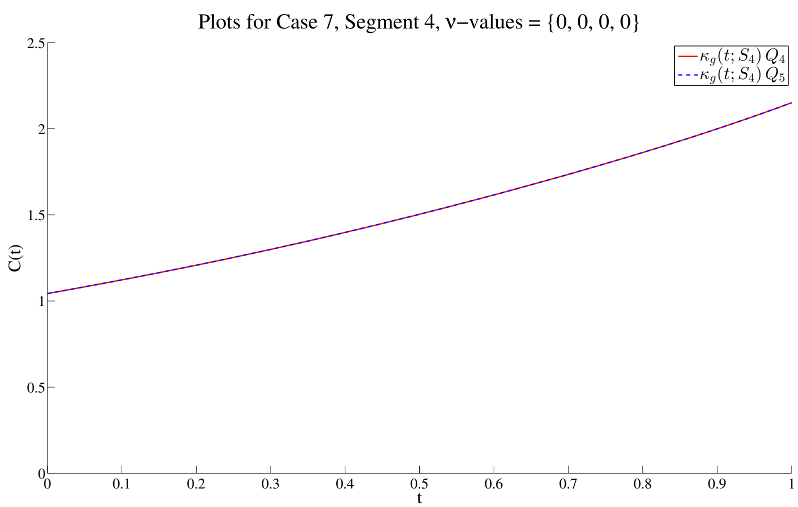

Plots for Case 7, Segment 1, ν−values = {10, 10, 10, 10}
$\kappa_g(t; S_1)\,Q_1$
$\kappa_g(t; S_1)\,Q_2$
C(t)
t

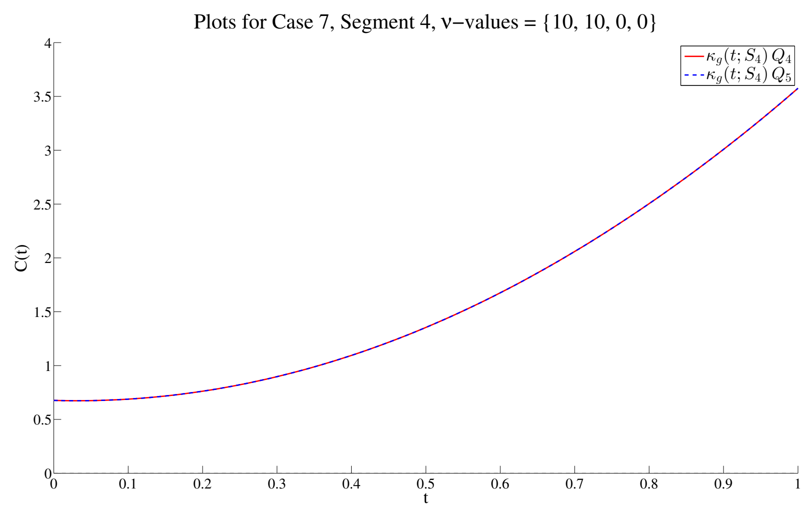

Plots for Case 7, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_2)\,Q_2$
$\kappa_g(t; S_2)\,Q_3$
C(t)
t


Plots for Case 7, Segment 2, ν−values = {10, 10, 10, 10}
$\kappa_g(t; S_2)\,Q_2$
$\kappa_g(t; S_2)\,Q_3$
C(t)
t


Plots for Case 7, Segment 3, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_3)\, Q_3$
$\kappa_g(t; S_3)\, Q_4$
C(t)
t


Plots for Case 7, Segment 3, ν−values = {10, 10, 10, 0}
$\kappa_g(t; S_3)\, Q_3$
$\kappa_g(t; S_3)\, Q_4$
C(t)
t


Plots for Case 7, Segment 4, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_4)\, Q_4$
$\kappa_g(t; S_4)\, Q_5$
C(t)
t


Plots for Case 7, Segment 4, ν−values = {10, 10, 0, 0}
$\kappa_g(t; S_4)\, Q_4$
$\kappa_g(t; S_4)\, Q_5$
C(t)
t

5.8 Case 8

This is a case demonstrating another example where a C^2-continuous curve satisfies all shape-preserving criteria. We begin from node P_0 which is the "outermost" here, and proceed spiraling inwards, thus creating an inherently convex setting. In the three internal segments for which convexity is required, we can see that zero-tension parameters produce a suitable curve.

```
     X          Y          Z
------------------------------
 0.963358, -0.137129,  0.230513
 0.919701, -0.382540,  0.088395
 0.488113, -0.755353, -0.437250
-0.851766, -0.455639, -0.258627
-0.023080, -0.071187,  0.997196
 0.852590, -0.296186,  0.430540
-0.375458, -0.920964,  0.104198
```

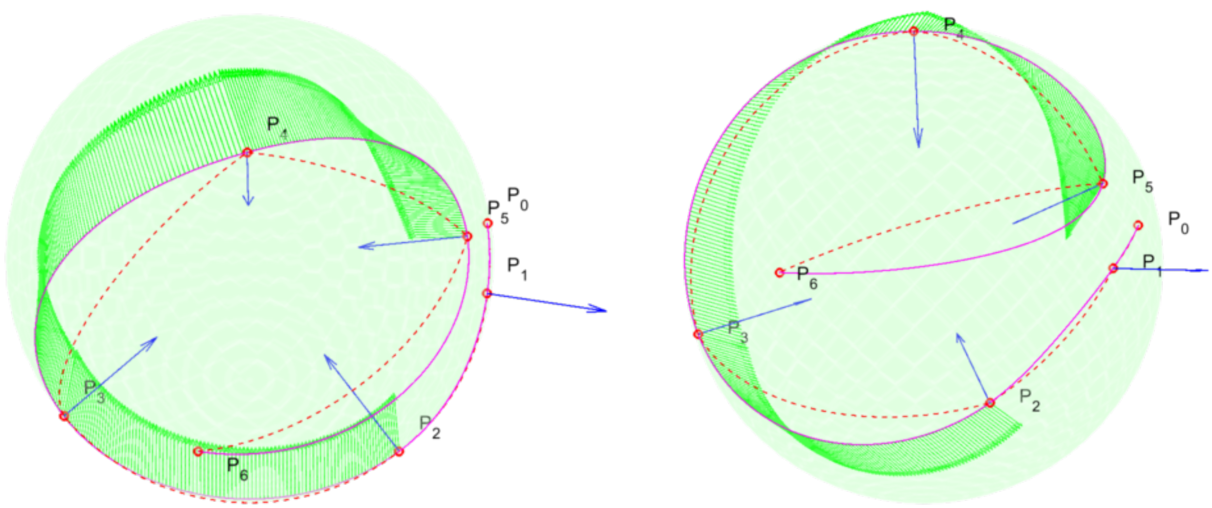
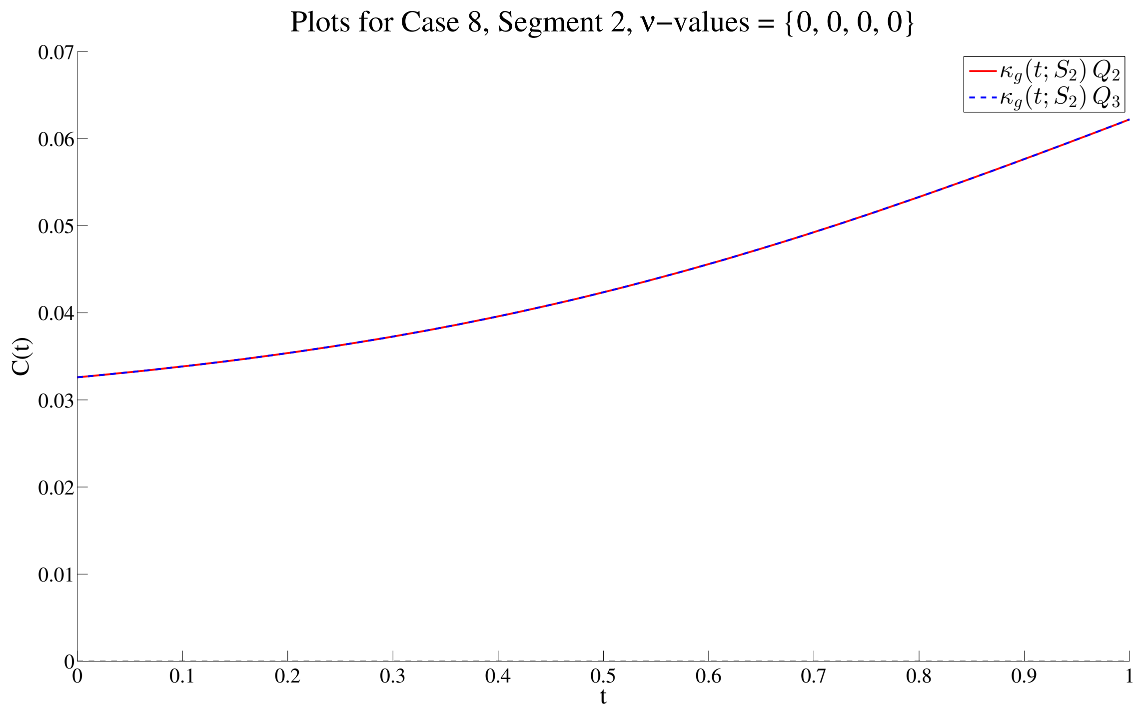
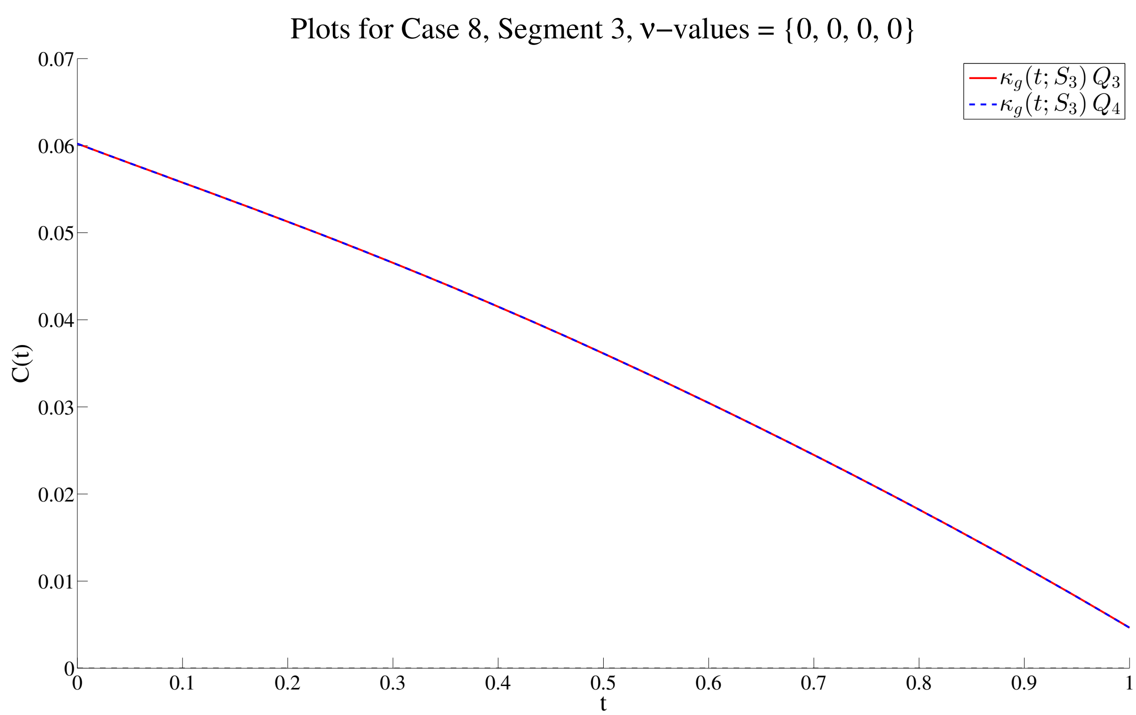
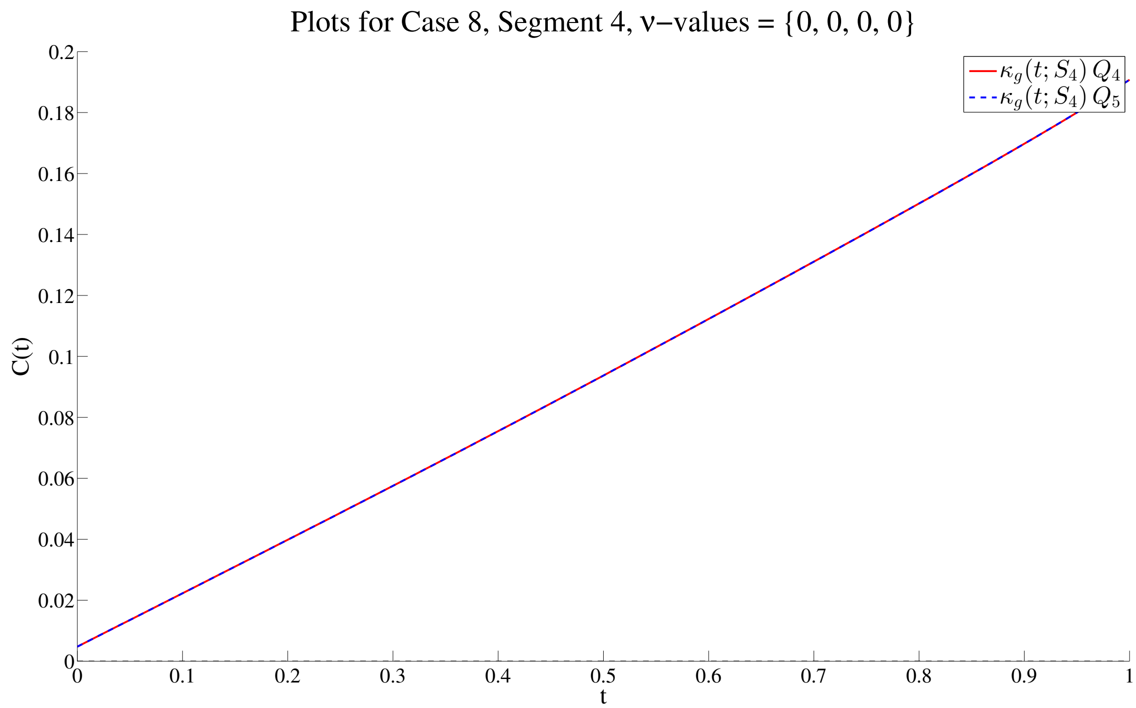


Figure 5.12: *Case 8 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*


Plots for Case 8, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_2)\, Q_2$
$\kappa_g(t; S_2)\, Q_3$
C(t)
t
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Plots for Case 8, Segment 3, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_3)\, Q_3$
$\kappa_g(t; S_3)\, Q_4$
C(t)
t
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



Plots for Case 8, Segment 4, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_4)\,Q_4$
$\kappa_g(t; S_4)\,Q_5$
C(t)
t
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

5.9 Case 9

Another example where all shape-reserving criteria are satisfied with zero-tension values.

```
    X          Y          Z
-------------------------------
-0.273240,  0.423877,  0.863521
-0.015828,  0.833876, -0.551724
 0.404872,  0.384930, -0.829402
-0.415558,  0.206476, -0.885821
-0.476242,  0.754058, -0.452316
```

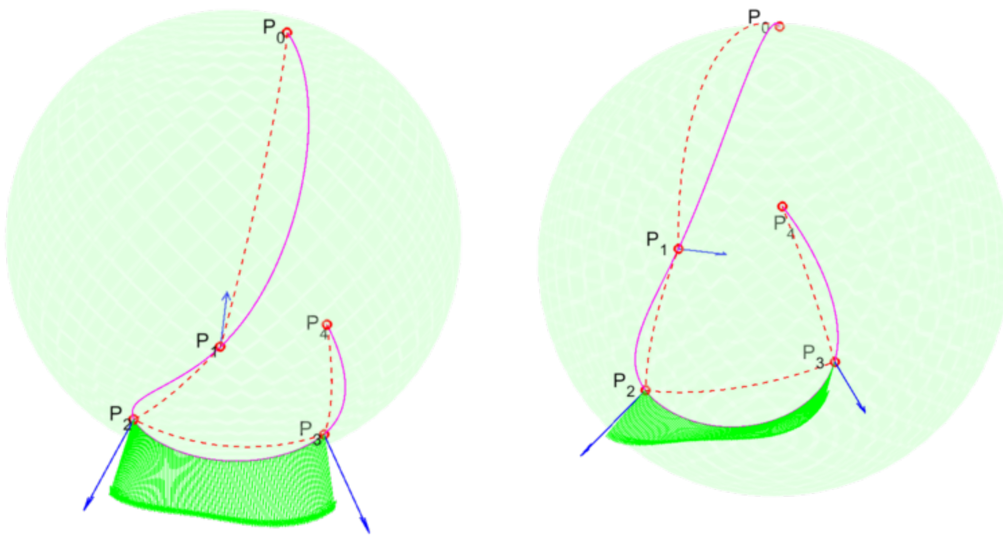
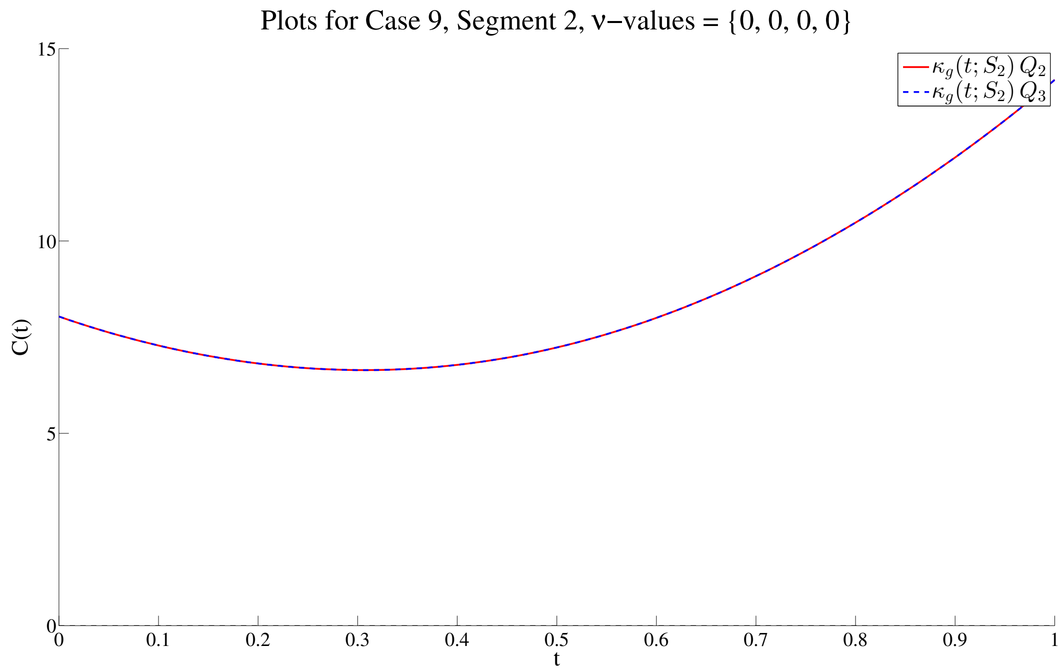


Figure 5.13: *Case 9 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*


Plots for Case 9, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_2)\,Q_2$
$\kappa_g(t;S_2)\,Q_3$
C(t)
t
15
10
5
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

5.10 Case 10

Here again shape-preservation is verified with all tension values equal to 0. It is interesting to note that node convexity is satisfied at node P_3. Here again no two consecutive binormals have the same sign, thus segment convexity is not required.

X	Y	Z
0.422968,	0.293534,	0.857284
0.261673,	0.422327,	0.867852
-0.673178,	0.295242,	0.677985
-0.576687,	0.815392,	0.050673
0.162503,	0.330596,	-0.929677

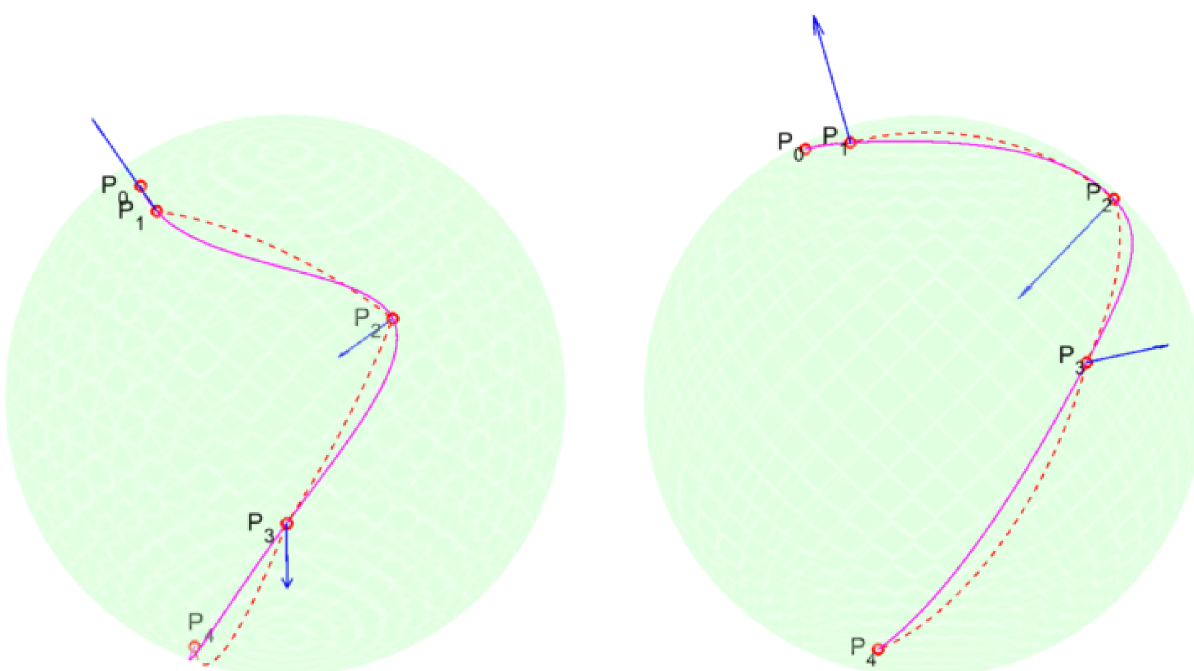


Figure 5.14: *Case 10 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*

5.11 Case 11

In this case, similarly to Case 7, a single increase step for the appropriate tension values is sufficient to satisfy the segment convexity criterion. We can see that in Segments 2 and 3 the curve does not satisfy our requirements with zero tension values, while augmenting only once bends the curve sufficiently to cause it to satisfy the convexity criteria.

```
    X           Y           Z
--------------------------------
 0.821695, -0.297540, -0.486094
-0.003780,  0.480555,  0.876956
 0.236220,  0.649826,  0.722444
-0.059150,  0.998160, -0.013354
-0.628260, -0.222031, -0.745649
-0.223151, -0.973184, -0.055827
```

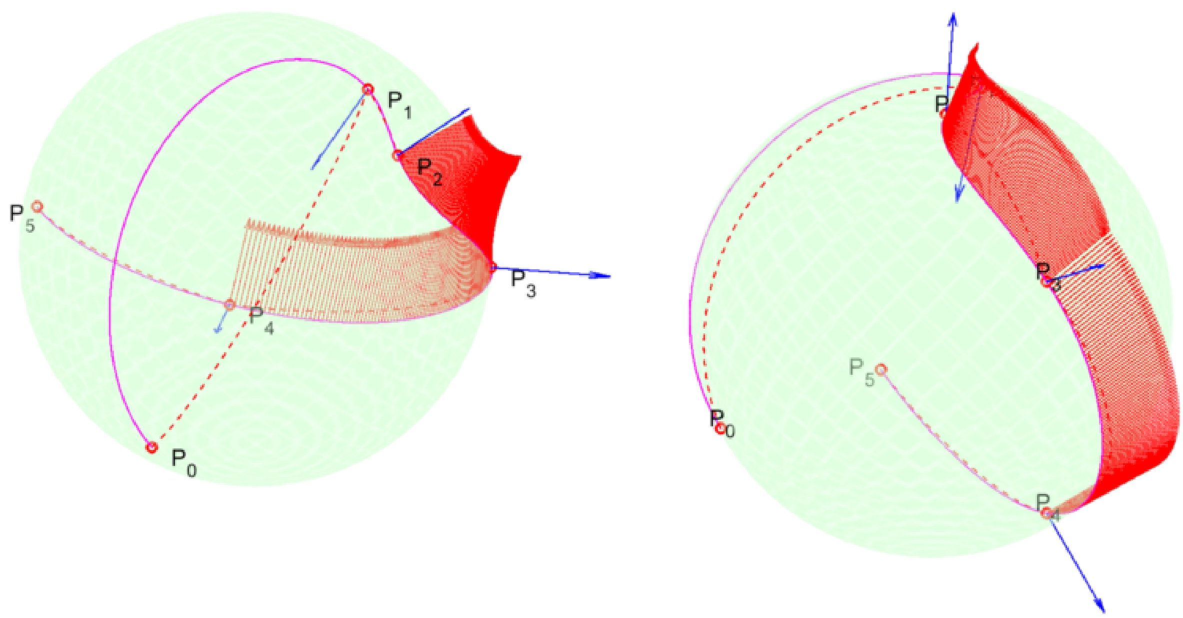


Figure 5.15: *Case 11 initial setting on the sphere from different viewpoints. All tension values are equal to zero.*

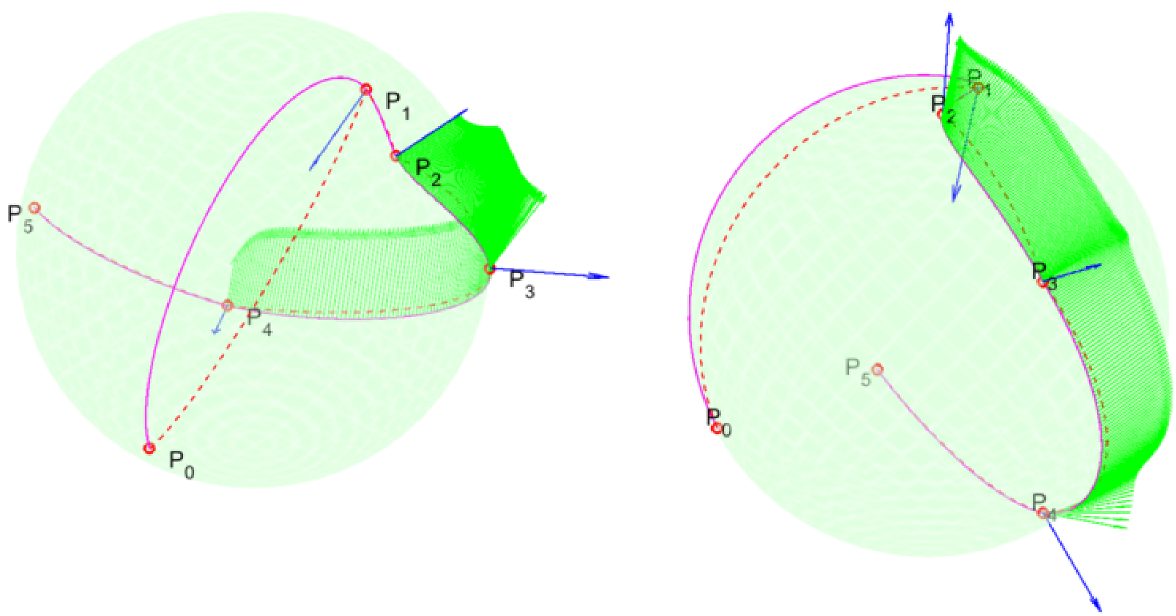
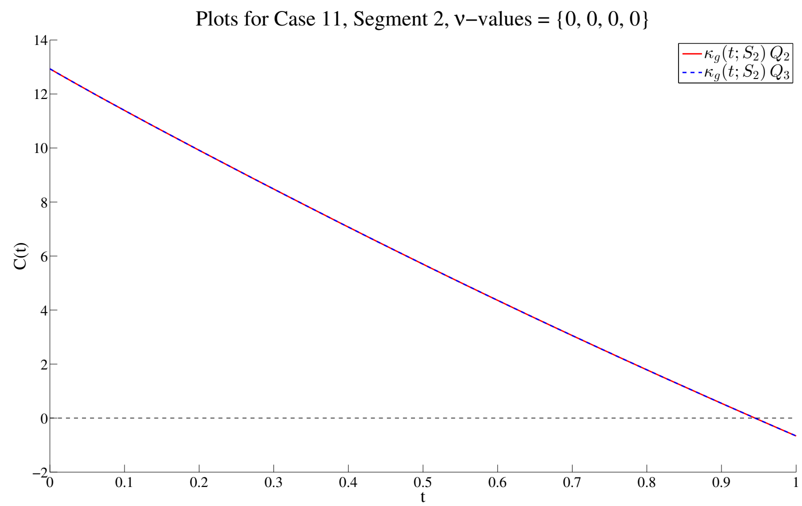
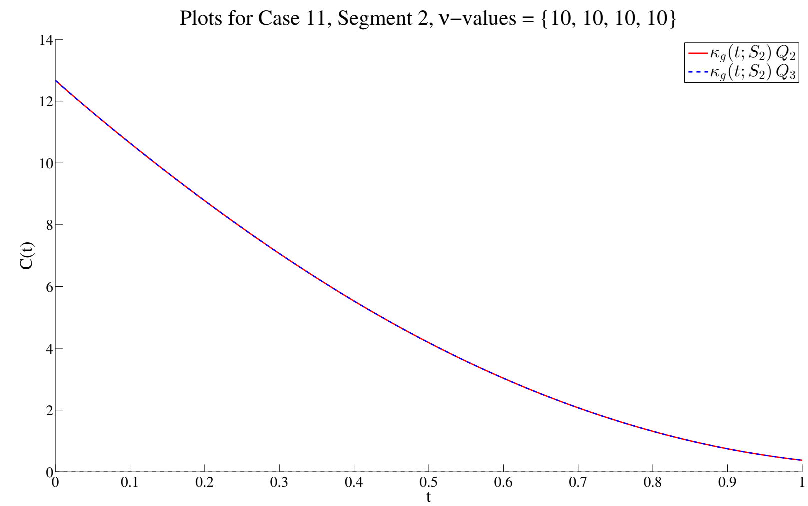
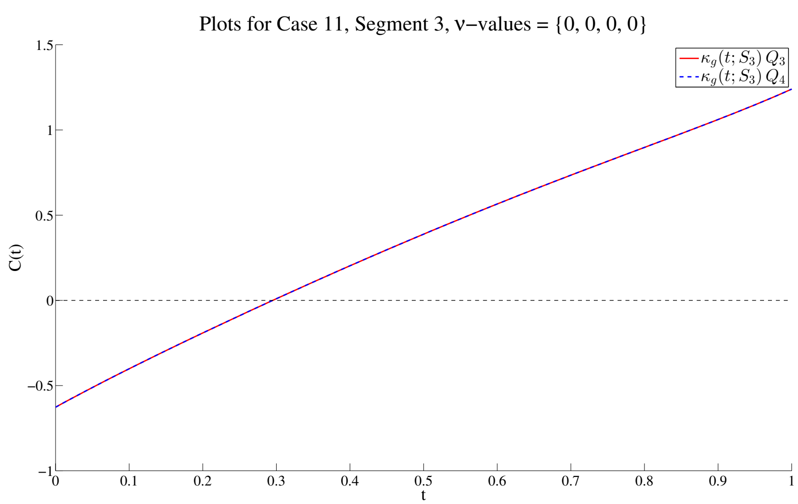
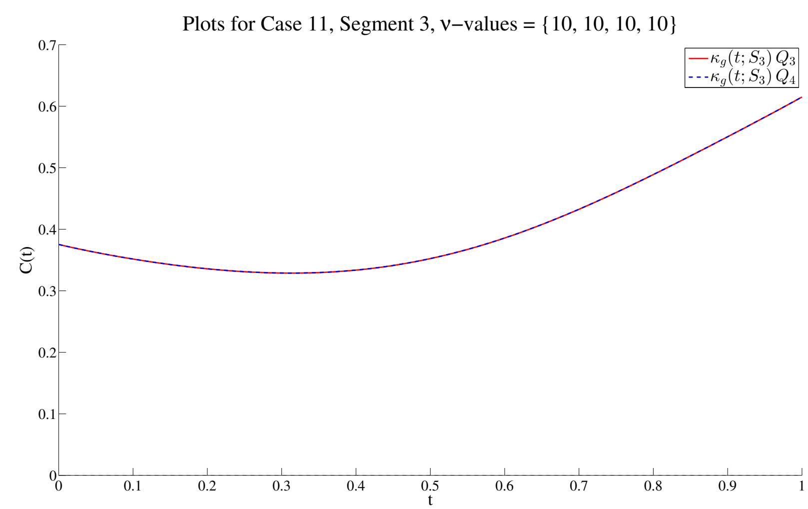


Figure 5.16: *Case 11 final setting on the sphere from different viewpoints. Tension values are* $\{0, 10, 10, 10, 10, 10\}$.


Plots for Case 11, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_2)\, Q_2$
$\kappa_g(t; S_2)\, Q_3$
C(t)
t


Plots for Case 11, Segment 2, ν−values = {10, 10, 10, 10}
$\kappa_g(t; S_2)\, Q_2$
$\kappa_g(t; S_2)\, Q_3$
C(t)
t


Plots for Case 11, Segment 3, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_3)\, Q_3$
$\kappa_g(t; S_3)\, Q_4$
C(t)
t


Plots for Case 11, Segment 3, ν−values = {10, 10, 10, 10}
$\kappa_g(t; S_3)\, Q_3$
$\kappa_g(t; S_3)\, Q_4$
C(t)
t

5.12 Case 12

This is another case where shape-preservation is satisfied with zero tension parameters. Note the behavior of the curve in Segment 1 – for both P_1 and P_2 nodal convexity is required and granted.

X	Y	Z
0.667140,	0.645016,	0.372665
0.695001,	0.176847,	-0.696921
0.076007,	0.663362,	-0.744428
-0.271177,	-0.456538,	-0.847370

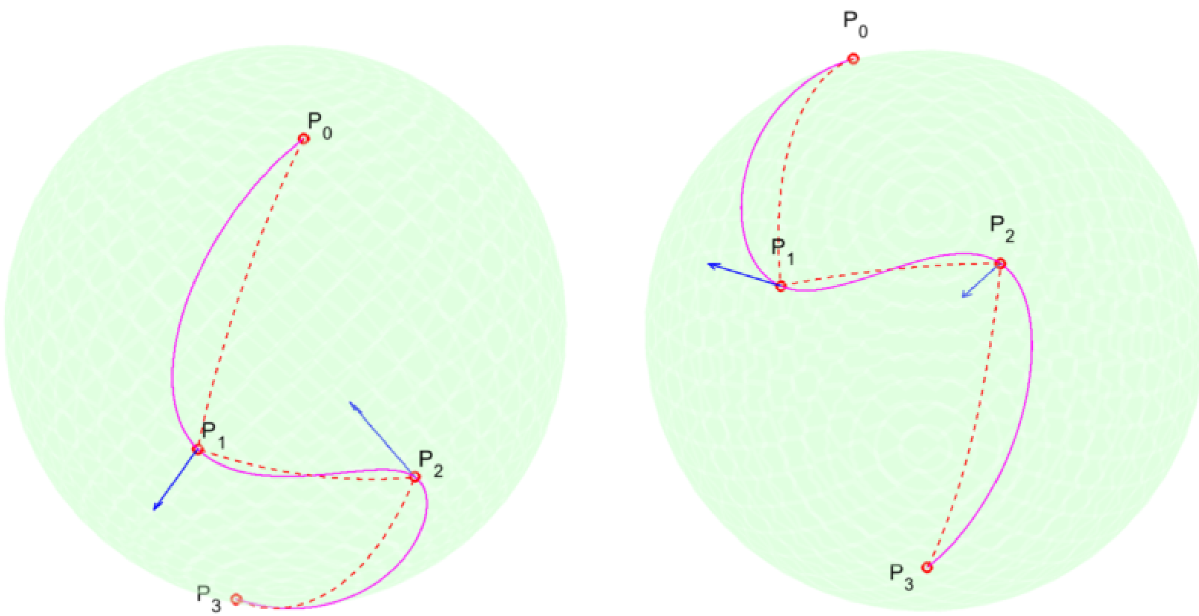


Figure 5.17: *Case 12 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*

5.13 Case 13

This example also shows the fulfillment of all shape-preserving criteria with zero tension values. Segment convexity is required for Segment 1, and is satisfied with zero tension values.

```
     X           Y           Z
-------------------------------
 0.349298, -0.244607,  0.904521
-0.299771,  0.896771,  0.325483
-0.928258, -0.371670,  0.014084
-0.333118, -0.868790,  0.366383
```

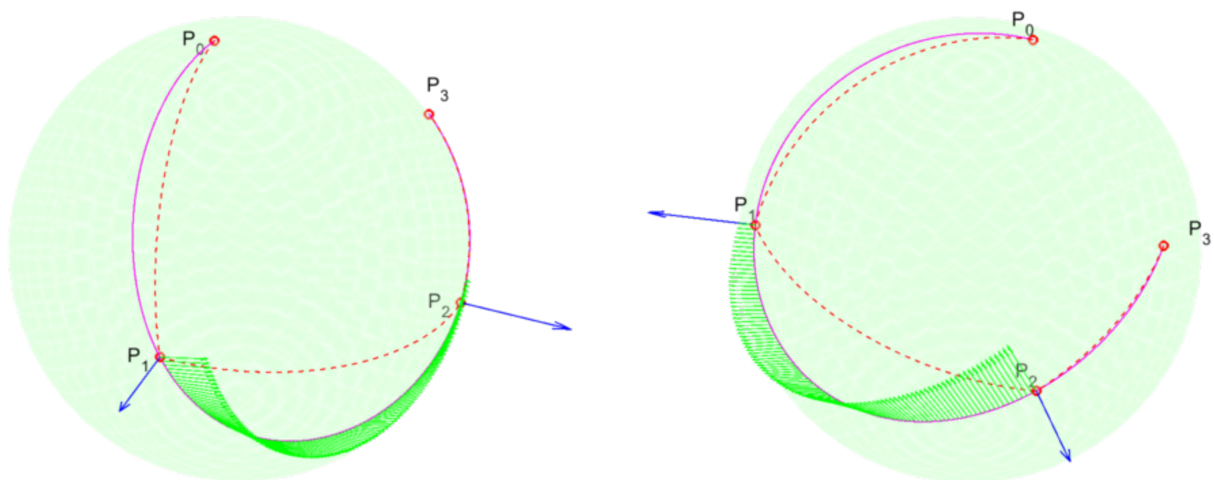
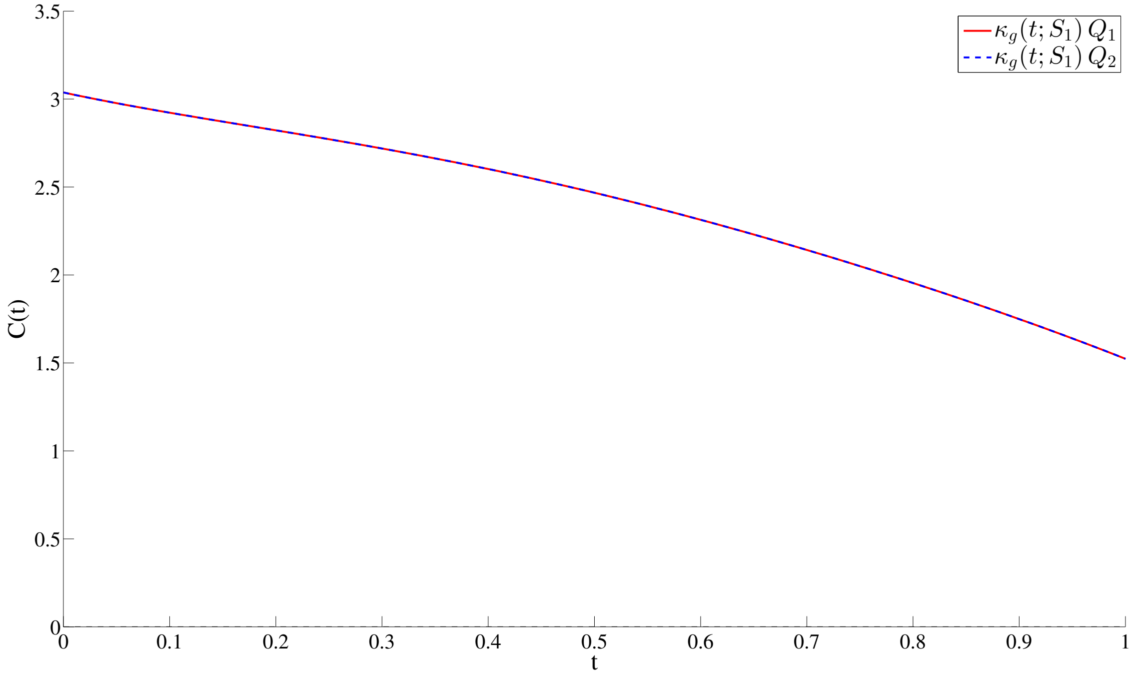


Figure 5.18: *Case 13 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*

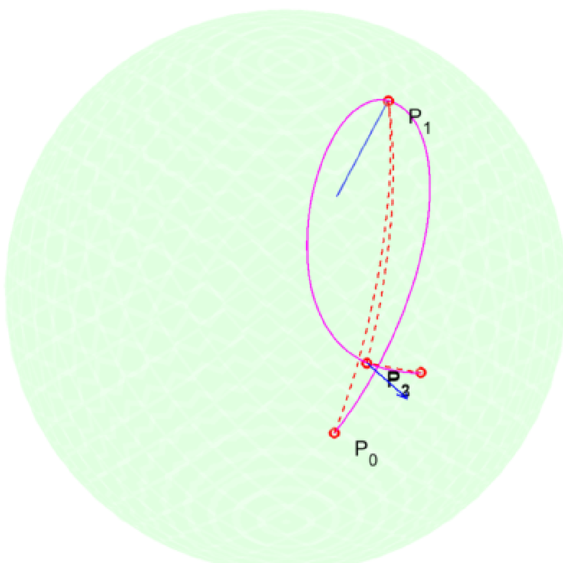
Plots for Case 13, Segment 1, ν−values = {0, 0, 0, 0}



5.14 Case 14

This is an interesting case where node convexity is not verified for node P_1 initially, as can be seen from the figures. Increasing the tension values causes the curve to “tighten” and fall in the correct half-space on both sides of P_1.

```
     X          Y          Z
--------------------------------
 0.904370,  0.420098,  0.075045
 0.041134,  0.385725,  0.921697
 0.797571,  0.510032,  0.322099
 0.692873,  0.680188,  0.239313
```



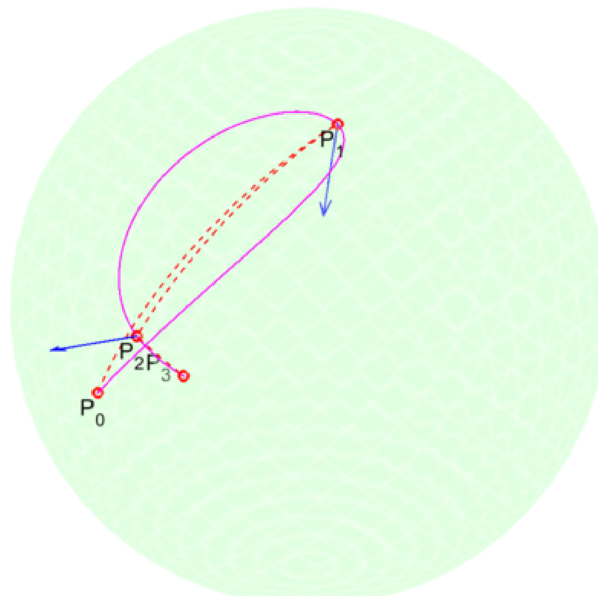


Figure 5.19: *Case 14 initial setting on the sphere from different viewpoints. All tension values are equal to zero.*

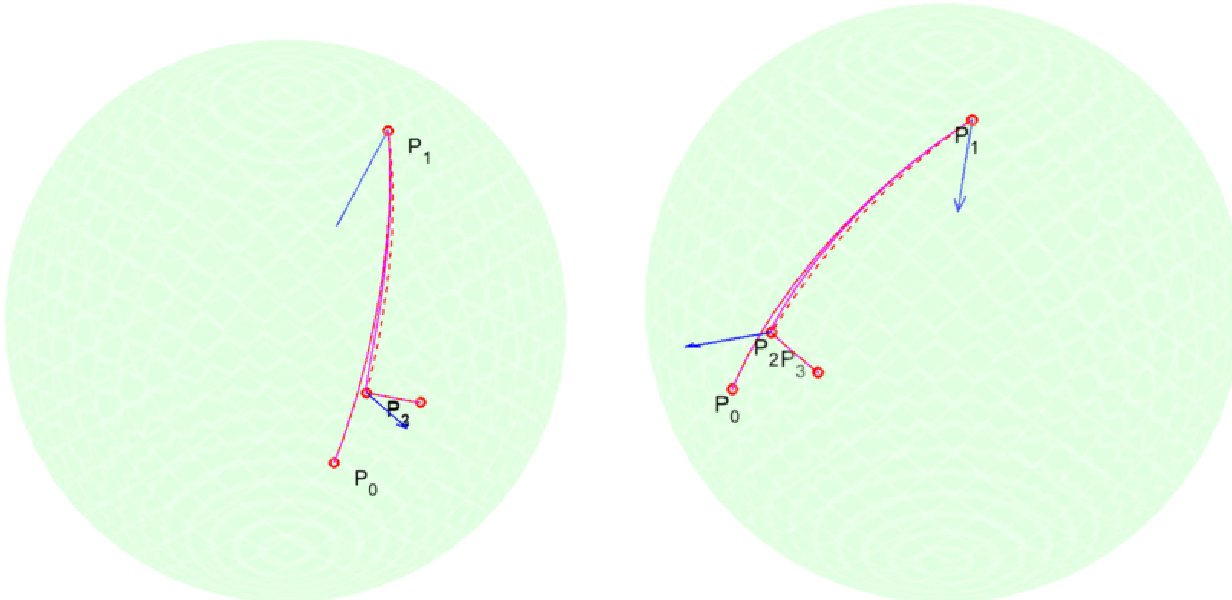


Figure 5.20: *Case 14 final setting on the sphere from different viewpoints. Tension values are* $\{280, 280, 280, 0\}$.

5.15 Case 15

A simple setup forming a “knot” with just four input points. Segment convexity is required, and as we can see it is found to be valid with the initial zero-tension values.

```
     X          Y          Z
-------------------------------
 0.774478,  0.077635,  0.627819
 0.405894,  0.664839,  0.627087
 0.669358,  0.517036,  0.533510
 0.524525,  0.310779,  0.792647
```

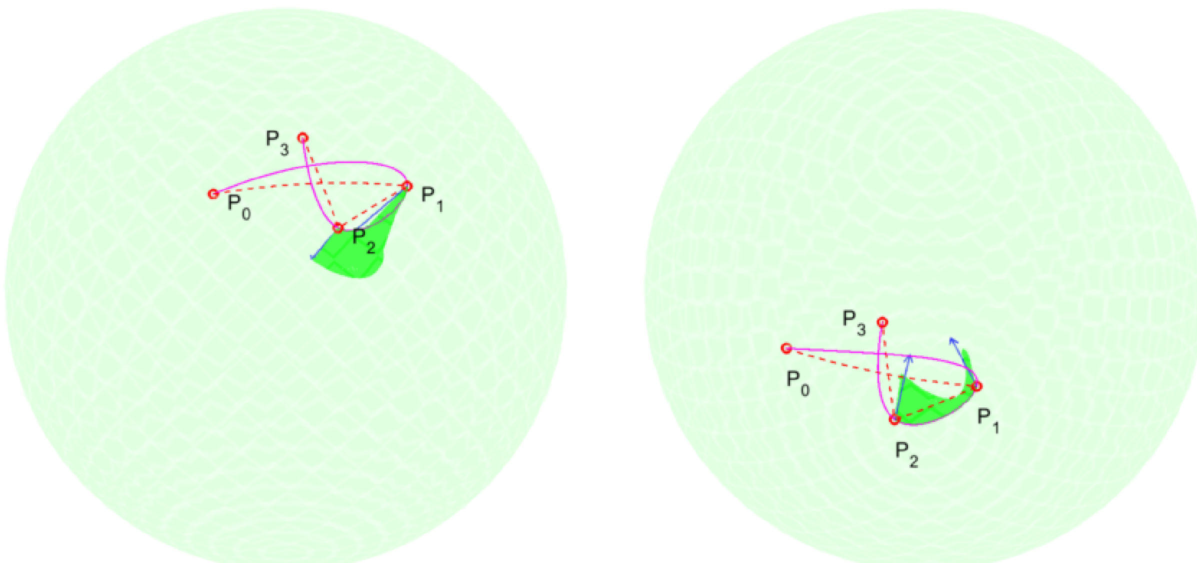
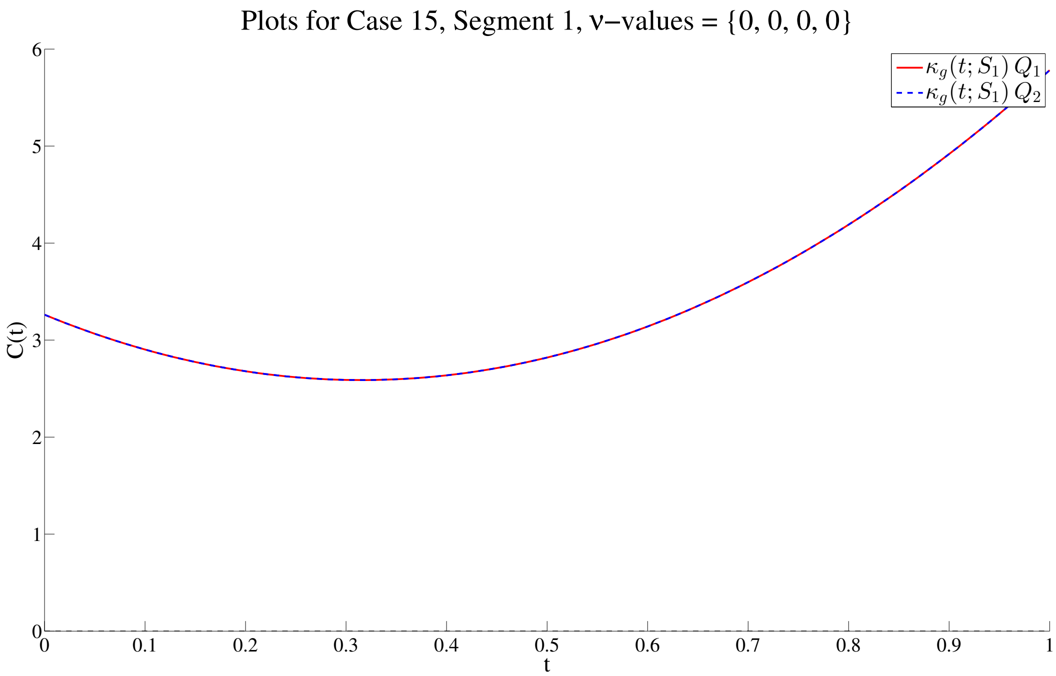


Figure 5.21: *Case 15 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*



Plots for Case 15, Segment 1, ν−values = {0, 0, 0, 0}
6
5
4
3
2
1
0
C(t)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
t
$\kappa_g(t; S_1)\, Q_1$
$\kappa_g(t; S_1)\, Q_2$

5.16 Case 16

Here we can see another convex setup which satisfies both nodal and segment requirements.

X	Y	Z
0.974665,	0.197237,	0.105474
0.872768,	0.037515,	0.486692
0.429167,	0.886710,	0.171932
0.783579,	0.500632,	0.367928

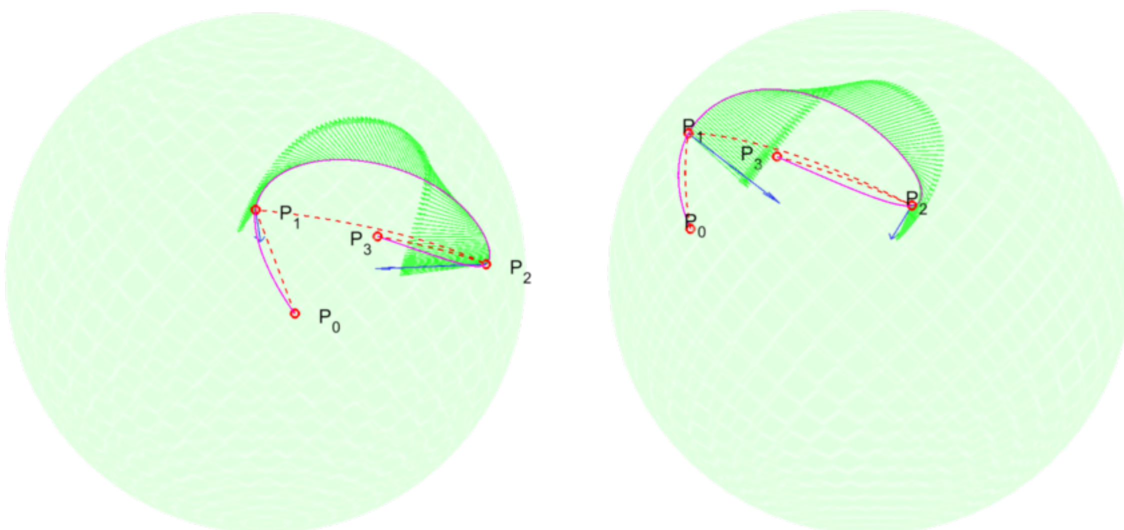
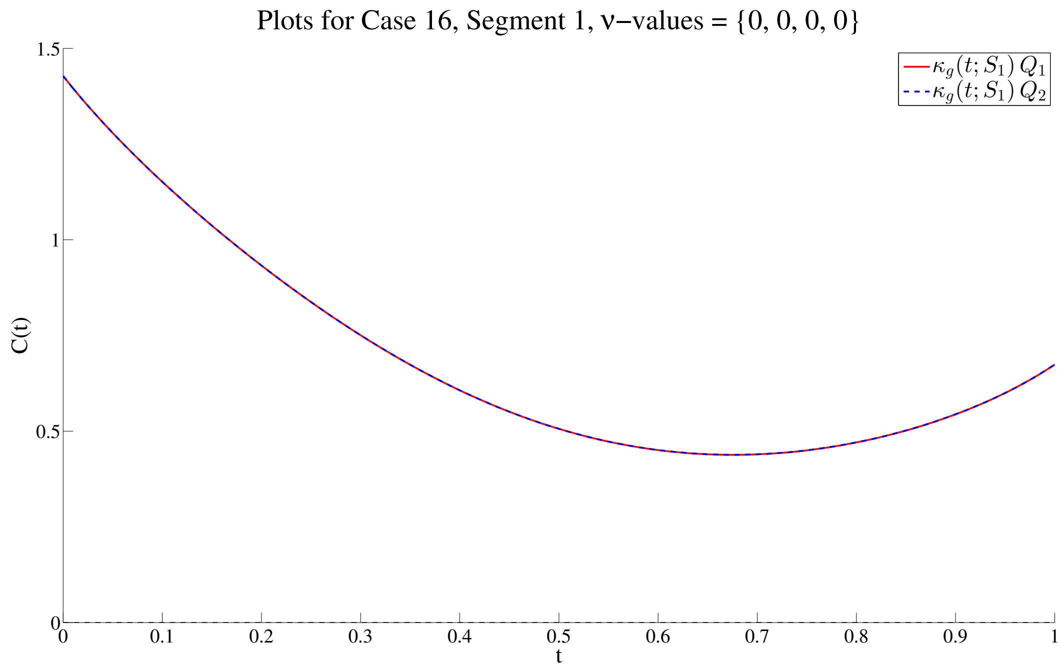


Figure 5.22: *Case 16 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*

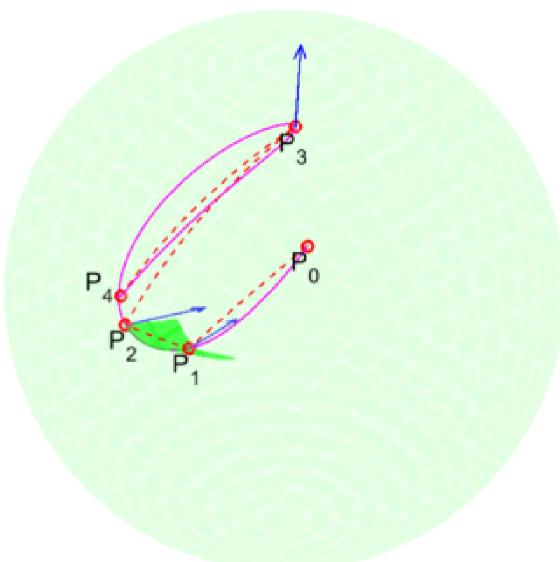


Plots for Case 16, Segment 1, ν−values = {0, 0, 0, 0}
$\kappa_g(t;S_1)\,Q_1$
$\kappa_g(t;S_1)\,Q_2$
C(t)
t
1.5
1
0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

5.17 Case 17

In this setup nodal convexity is not satisfied at node P_3, but by incrementing the corresponding tension values the criterion is satisfied. Note that the procedure does not have an adverse effect on the convexity of Segment 1.

```
      X           Y           Z
--------------------------------
 0.168300,  0.699821,  0.694208
 0.617717,  0.700415,  0.357553
 0.783579,  0.500632,  0.367928
 0.077453,  0.326002,  0.942191
 0.779081,  0.437652,  0.448880
```



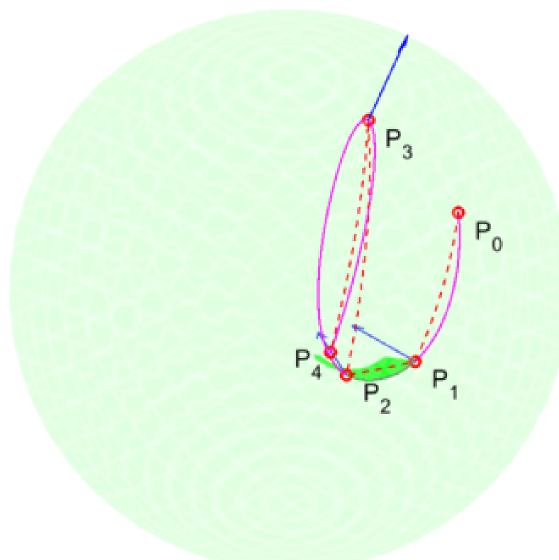
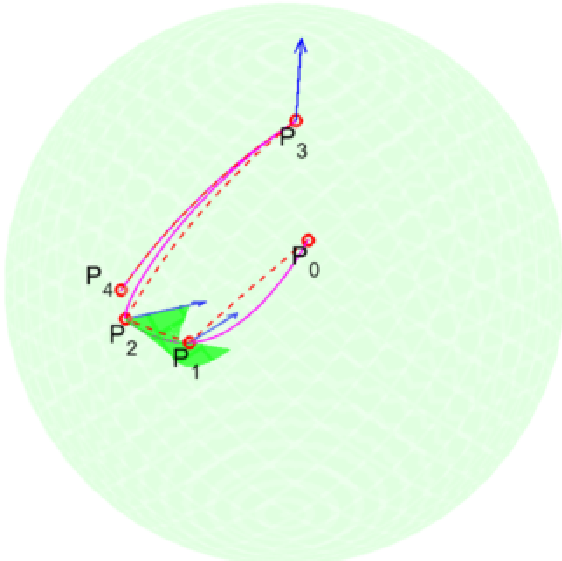


Figure 5.23: *Case 17 initial setting on the sphere from different viewpoints. All tension values are equal to zero.*



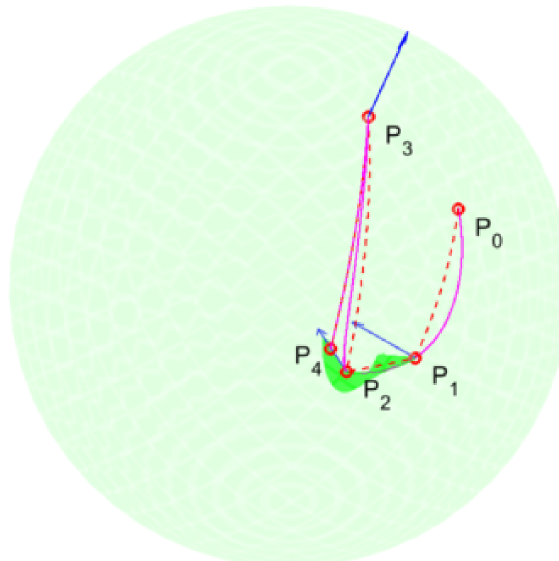
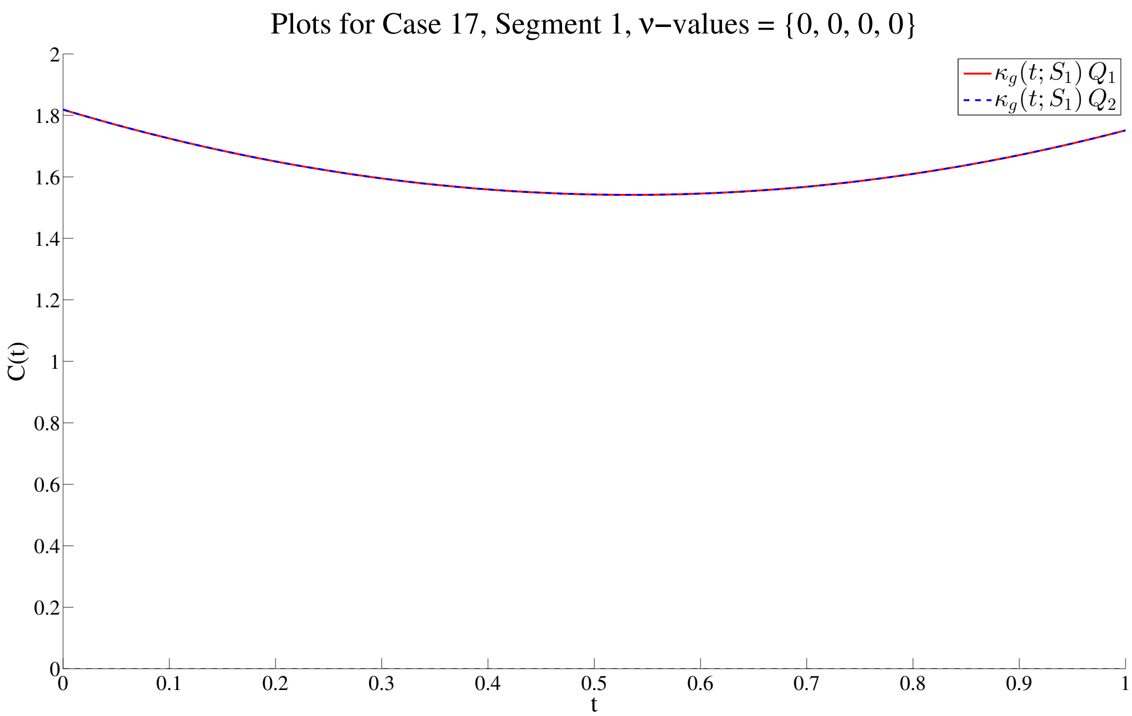
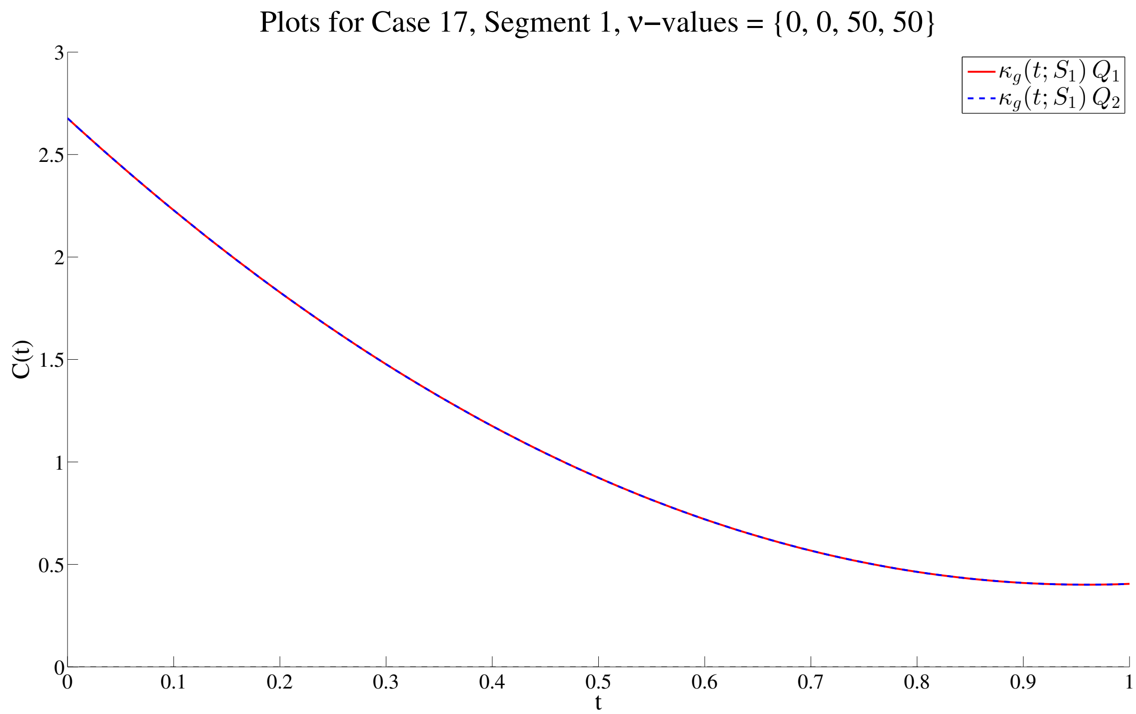


Figure 5.24: *Case 17 final setting on the sphere from different viewpoints. Tension values are* $\{0, 0, 50, 50, 50\}$.




Plots for Case 17, Segment 1, ν−values = {0, 0, 50, 50}
$\kappa_g(t; S_1)\, Q_1$
$\kappa_g(t; S_1)\, Q_2$
C(t)
t
0
0.5
1
1.5
2
2.5
3
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

5.18 Case 18

Another interesting case where segment convexity is verified with zero tension values.

X	Y	Z
0.678714,	0.572556,	0.459921
0.287136,	0.410225,	0.865603
0.187111,	0.673429,	0.715180
0.039320,	0.043193,	0.998293
0.779081,	0.437652,	0.448880

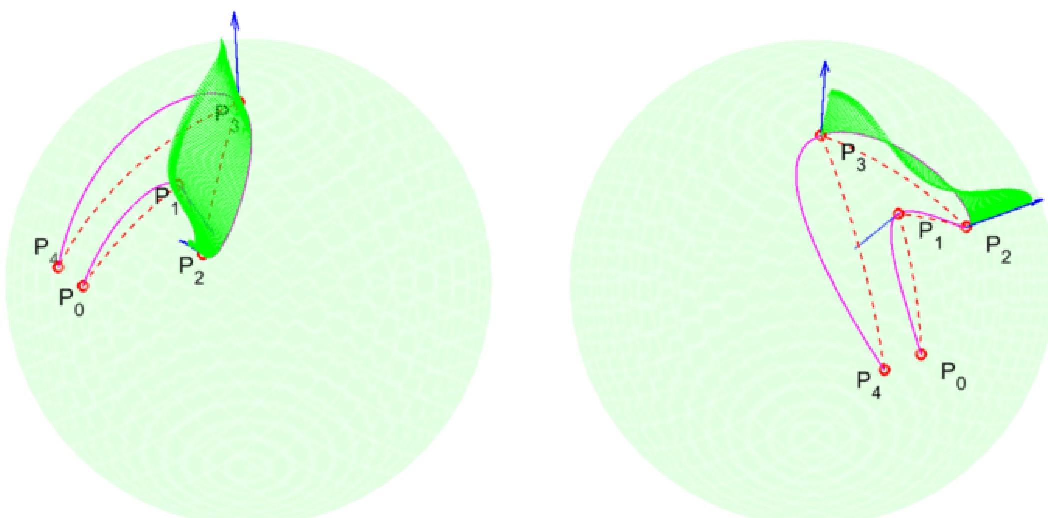
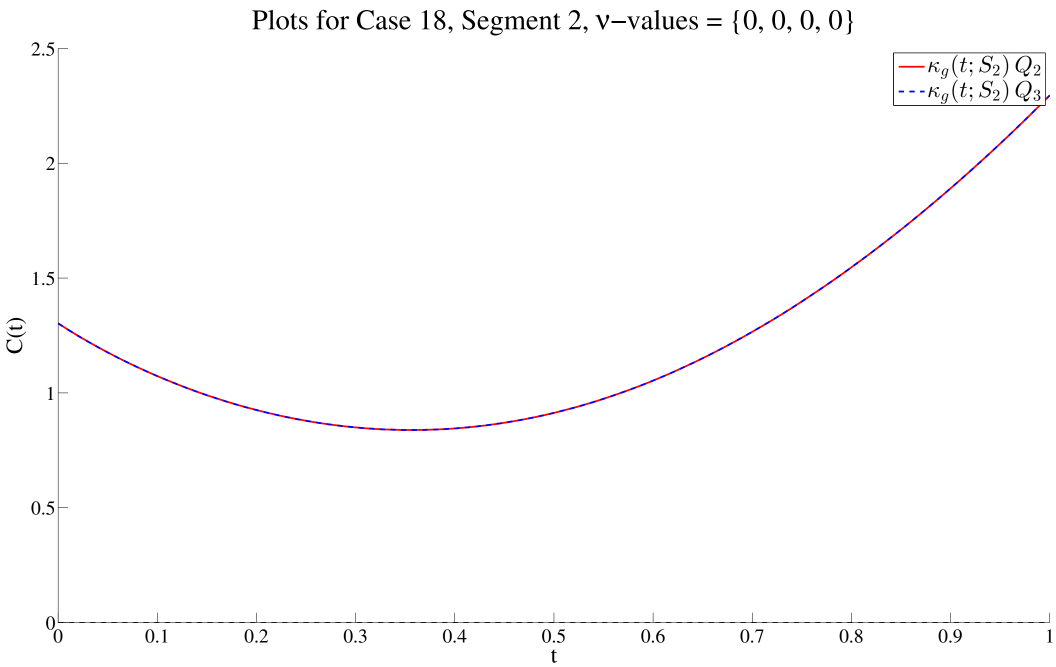


Figure 5.25: *Case 18 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*



Plots for Case 18, Segment 2, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_2)\, Q_2$
$\kappa_g(t; S_2)\, Q_3$
C(t)
t
0
0.5
1
1.5
2
2.5
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

5.19 Case 19

This is a case similar to Case 17, where node convexity is not initially verified but is achieved within 2 increment steps of the appropriate tension values. We can see again that the convexity of Segment 1 is not affected negatively.

```
     X          Y          Z
-------------------------------
 0.441525,  0.864293,  0.240941
 0.512342,  0.453462,  0.729299
 0.875010,  0.472468,  0.105504
 0.768660,  0.555493,  0.317157
 0.876014,  0.480475,  0.041743
 0.212173,  0.965682,  0.149801
```

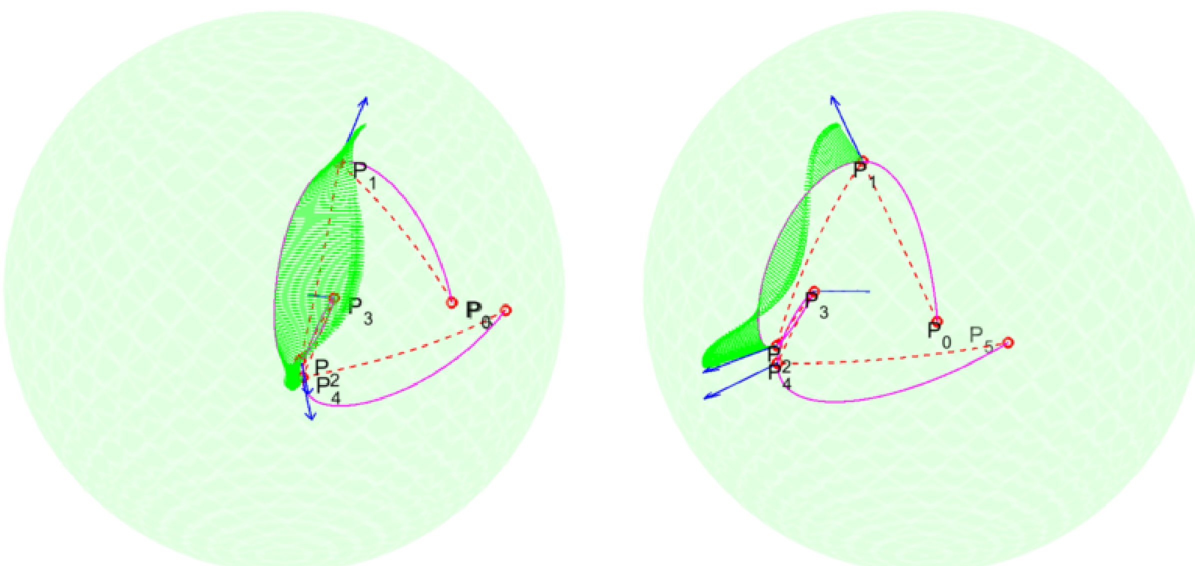


Figure 5.26: *Case 19 initial setting on the sphere from different viewpoints. All tension values are equal to zero.*

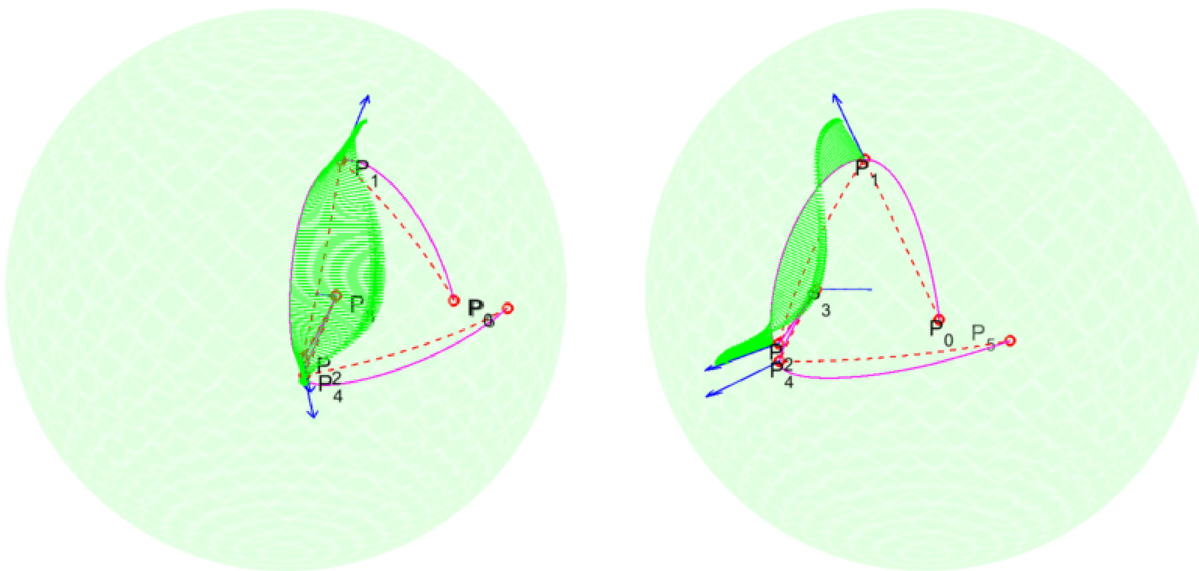
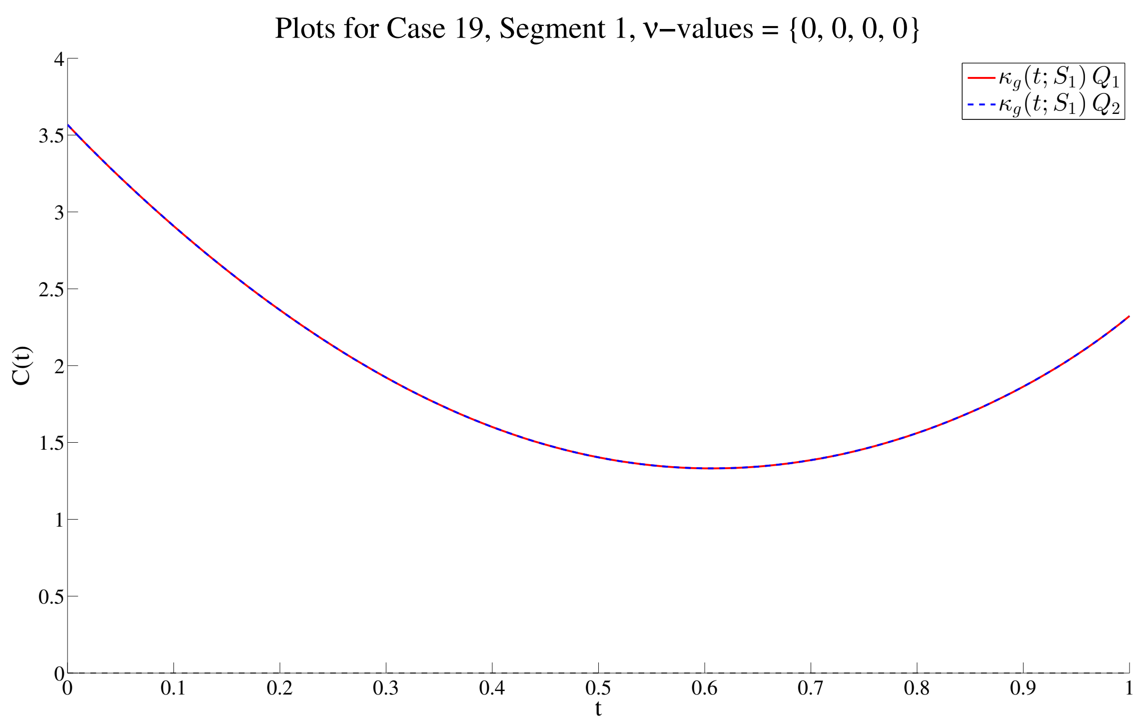
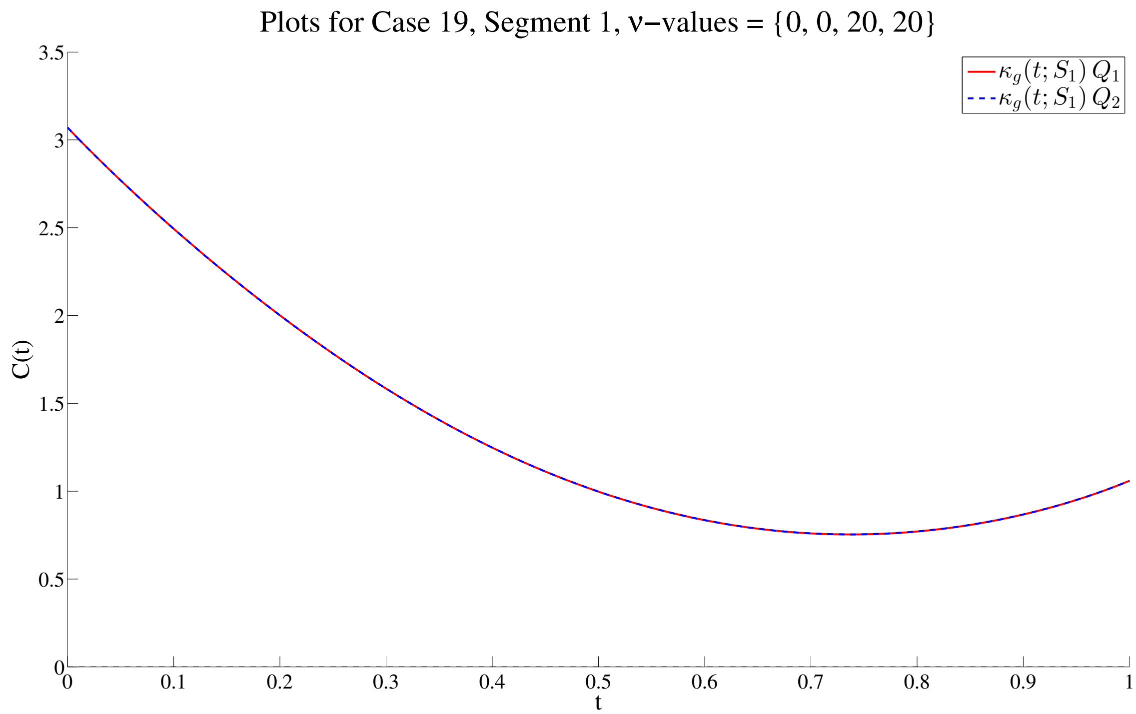


Figure 5.27: *Case 19 final setting on the sphere from different viewpoints. Tension values are* $\{0, 0, 20, 20, 20, 0\}$.





Plots for Case 19, Segment 1, ν−values = {0, 0, 20, 20}
$\kappa_g(t; S_1)\, Q_1$
$\kappa_g(t; S_1)\, Q_2$
C(t)
t
3.5
3
2.5
2
1.5
1
0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

5.20 Case 20

This setup is identical to Case 1, we have only switches the locations of nodes P_1 and P_2. This is to illustrate that co-circularity is not requested in this case where the ordering of the co-circular nodes is not respected, and the algorithm terminates with the initial zero tension values.

```
    X          Y          Z
------------------------------
 1.000000,  0.000000,  0.000000
 0.000000,  1.000000,  0.000000
 0.707100,  0.707100,  0.000000
 0.000000,  0.000000,  1.000000
 0.000000, -0.879700,  0.475500
 1.000000,  0.000000,  0.000000
```

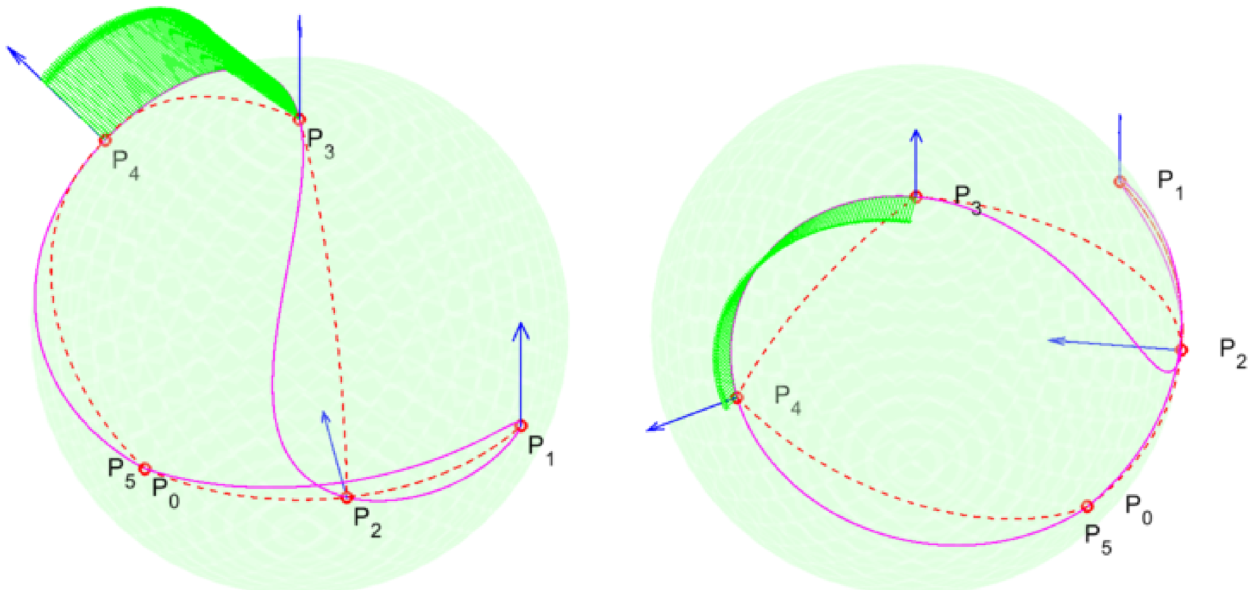
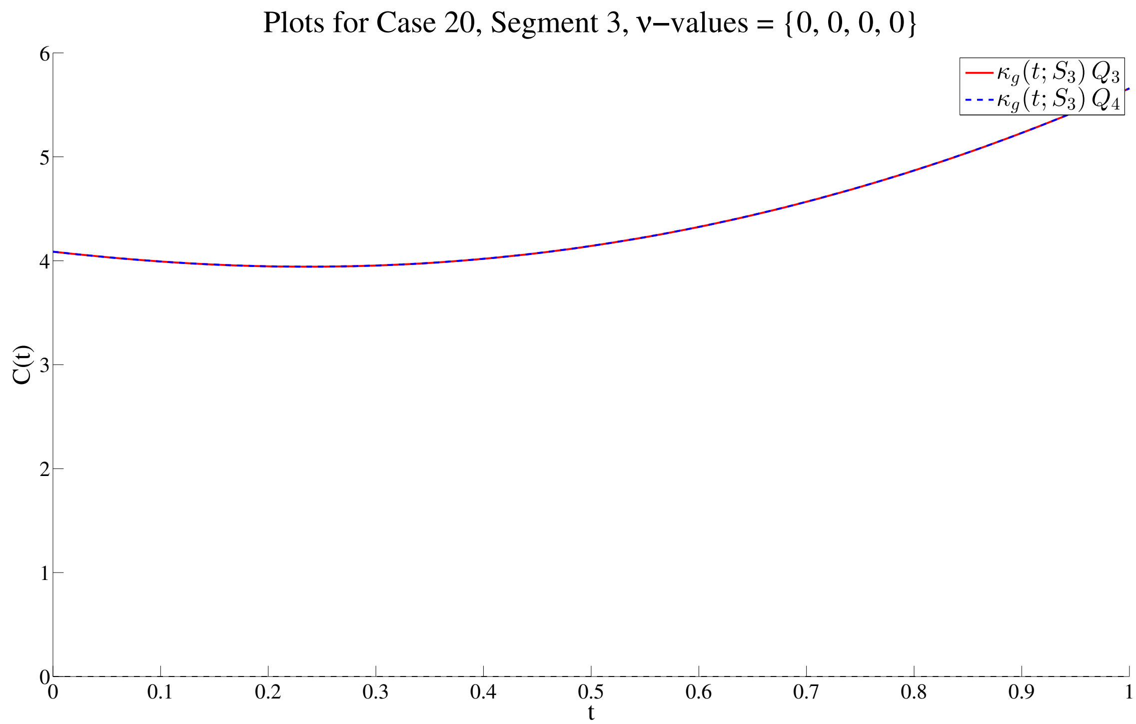


Figure 5.28: *Case 20 initial setting on the sphere from different viewpoints. This is also the final setting, meaning that the curve satisfies all shape-preserving requirements with all tension values equal to zero.*


Plots for Case 20, Segment 3, ν−values = {0, 0, 0, 0}
$\kappa_g(t; S_3)\, Q_3$
$\kappa_g(t; S_3)\, Q_4$
C(t)
t
0
1
2
3
4
5
6
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

Chapter 6

Appendix

6.1 Code

6.1.1 Example configuration file

File: config

```
input_file        input/data20.txt
input_type        cartesian
increase_type     plus
initial_nu        0.0
increase_factor   10.0
Stol              1e-10
Snmax             1000
epsilon           1e-3
normal_tol        1e-4
h                 1e-4
Bnmax             100
Btol              1e-10
```

6.1.2 Makefile

File: Makefile

```

CC=g++
ARGS=-llapack -lblas

spi:  obj/main.o obj/fd.o obj/nuspl.o obj/opt.o \
      obj/shape.o obj/spnode.o obj/util.o
  $(CC) -o spi  obj/main.o obj/fd.o obj/nuspl.o \
  obj/opt.o obj/shape.o obj/spnode.o obj/util.o $(ARGS)

obj/main.o: main.cpp
  $(CC) -c main.cpp -o obj/main.o

obj/fd.o: src/FiniteDifference.cpp src/FiniteDifference.hpp
  $(CC) -c src/FiniteDifference.cpp -o obj/fd.o

obj/nuspl.o: src/NuSpline.cpp src/NuSpline.hpp
  $(CC) -c src/NuSpline.cpp -o obj/nuspl.o

obj/opt.o: src/Options.cpp
  $(CC) -c src/Options.cpp -o obj/opt.o

obj/shape.o: src/ShapePreservation.cpp src/ShapePreservation.hpp
  $(CC) -c src/ShapePreservation.cpp -o obj/shape.o

obj/spnode.o: src/SplineNode.cpp src/SplineNode.hpp
  $(CC) -c src/SplineNode.cpp -o obj/spnode.o

obj/util.o: src/Utility.cpp src/Utility.hpp
  $(CC) -c src/Utility.cpp -o obj/util.o

clean:
  rm -rfv obj/*.o spi
```

6.1.3 Main

File: main.cpp

```
#include <iostream>
#include "src/Utility.hpp"
#include "src/SplineNode.hpp"
#include "src/NuSpline.hpp"
#include "src/ShapePreservation.hpp"
#include "src/Options.cpp"

using namespace std;

typedef ublas::vector<double>   Vec;
typedef std::vector<SplineNode> SVec;


// Central point of the method
int main(int argc, char** argv) {

  if (argc < 2) {
    cout << "No config file provided!" << endl;
    cout << "Usage: " << argv[0] << " config_file" << endl;
    return -1;
  }

  // Read options and print values on screen
  cout <<      "\t #### OPTIONS ####"           << endl;
  cout << "-----------------------------------" << endl;
  Options *opt = new Options();
  ReadOptions(argv[1], opt);

  cout << "Input file:      " << opt->input_file        << endl;
  cout << "Points type:     " << opt->input_type        << endl;
  cout << "Increase type:   " << opt->increase_type     << endl;
  cout << "Increase factor: " << opt->increase_factor   << endl;
  cout << "Initial tension: " << opt->initial_nu        << endl;
  cout << "Stol:            " << opt->Stol              << endl;
  cout << "Snmax:           " << opt->Snmax             << endl;
  cout << "epsilon:         " << opt->epsilon           << endl;
  cout << "normal_tol:      " << opt->normal_tol        << endl;
  cout << "h:               " << opt->h                 << endl;
  cout << "Bnmax:           " << opt->Bnmax             << endl;
  cout << "Btol:            " << opt->Btol              << endl;
  cout << "-----------------------------------" << endl << endl;

  // Read input points
  mp_type points;
  SVec nodes;
  if (opt->input_type.compare("cartesian") == 0)
    ReadCartesianPoints(opt->input_file, &points);
  else if (opt->input_type.compare("spherical") == 0)
    ReadSphericalPoints(opt->input_file, &points);
  else {
    printf("Please control your options file! Select a valid points format! \n");
    return -1;
  }

  // Create a vector of SplineNodes with the points read
  for (mp_type::iterator it = points.begin(); it != points.end(); it++) {
    SplineNode *sn = new SplineNode(*it, opt->initial_nu);
    nodes.push_back(*sn);
  }

```

```
  // Create a NuSpline object and initialize it
  NuSpline *spl = new NuSpline();
  spl->Init(&nodes, *opt);

  // Boolean masks
  bool idx_coll[nodes.size()];
  bool idx_ncnv[nodes.size()];
  bool idx_scnv[nodes.size()];
  bool idx[nodes.size()];

  bool flag = false;

  // Begin iterations
  do {
    bool convergence;

    cout << "Tension values:" << endl;
    for (int i = 0; i < nodes.size(); i++) {
      cout << nodes[i].GetNu() << " ";
    }
    cout << endl;

    // Compute the nu-spline with the current tension values
    convergence = spl->Execute(&nodes, opt->Snmax, opt->Stol);
    if (!convergence) {
      cout << "Spline could not be computed correctly with " \
      "Snmax = " << opt->Snmax << " and Stol = " << opt->Stol << endl;
      cout << "Please change options and try again! Exiting..." << endl << endl;
      return -1;
    }

    // Set all masks to False
    for (int i = 0; i < nodes.size(); i++) {
      idx_coll[i] = false;
      idx_ncnv[i] = false;
      idx_scnv[i] = false;
      idx[i] = false;
    }

    cout << "Criteria!" << endl;
    // Check shape-preservation criteria
    VerifyCocircularity   (Spline, nodes, idx_coll, opt);
    VerifyNodeConvexity   (Spline, nodes, idx_ncnv, opt);
    VerifySegmentConvexity(Spline, nodes, idx_scnv, opt);

    cout << "Co-circularity:    " << \
            (Verify(idx_coll, nodes.size()) ? "NOT OK" : "OK") << endl;
    cout << "Nodal convexity:   " << \
            (Verify(idx_ncnv, nodes.size()) ? "NOT OK" : "OK") << endl;
    cout << "Segment convexity: " << \
            (Verify(idx_scnv, nodes.size()) ? "NOT OK" : "OK") << endl;

    // Logical OR for the masks
    for (int i = 0; i < nodes.size(); i++) {
        idx[i] = ( idx_coll[i] || idx_ncnv[i] || idx_scnv[i] );
    }

    // Check whether there are tension values to increase
    flag = Verify(idx, nodes.size());
    if (flag) {
      // If yes, print out the boolean mask...
      cout << "Index mask:" << endl;
      for (int i = 0; i < nodes.size(); i++) {
        cout << idx[i] << " ";
      }
```

```
            cout << endl << endl;

            // ...and increment the tension values
            AugmentNuValues(&nodes, idx, opt);
        }

    } while (flag); // Iterate while tension values are not OK

    // Print out results
    cout << endl << "==========================================================" << endl;
    cout <<             "\t Final tension values and control points"              << endl;
    cout <<          "----------------------------------------------------------" << endl;
    for (int i = 0; i < nodes.size(); i++) {
        Vec v = nodes[i].GetD();
        double nu = nodes[i].GetNu();
        cout << "Node " << i << ": Tension value = " << nu << \
            ", \t Control point = [" << v(0) << ", " << v(1) << ", " << v(2) << "]" << endl;
    }

    return 0;
}
```

6.1.4 Options

File: src/Options.cpp

```
#ifndef __OPTIONS_H_INCLUDED
#define __OPTIONS_H_INCLUDED

#include <string>

class Options{

public:
  std::string input_file;       // Name of the input file
  std::string input_type;       // Input type (cartesian or spherical)
  std::string increase_type;    // Mode in which to increase tension
                                // values. Either 'times' or 'plus'
  double      increase_factor;  // Value operating on the tensions
                                // (either multiplication or addition)
  double      initial_nu;       // Initial tension value (for all nodes)
  double      Stol;       // Tolerance for the spline algorithm
  int         Snmax;      // Max iterations for the spline algorithm
  double      epsilon;    // Threshold after which nodes are considered co-circular
  double      normal_tol; // Threshold for convexity indicators
  double      h;          // Stem size for the finite difference approximations
  int         Bnmax;      // Max iterations for the bisection method
  double      Btol;       // Tolerance for the bisection method

  // Simple initialization
  Options() {
    input_file      = "";
    input_type      = "";
    increase_type   = "plus";
    increase_factor = 1.0;
    initial_nu      = 0.0;
    Stol            = 0.0;
    Snmax           = 0;
    epsilon         = 0.0;
    normal_tol      = 0.0;
    h               = 0.0;
    Bnmax           = 0;
    Btol            = 0.0;
  }
};

#endif
```

6.1.5 SplineNode

File: src/SplineNode.hpp

```
#ifndef __SPLINENODE_H_INCLUDED
#define __SPLINENODE_H_INCLUDED

#include <iostream>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/numeric/bindings/traits/ublas_vector2.hpp>

using namespace std;

typedef boost::numeric::ublas::vector<double> Vec;

class SplineNode {
private:
  Vec       P;    // Point on the sphere
  Vec       L;    // Left aux. point
  Vec       R;    // Right aux. point
  Vec       d;    // Control point (Nielson spline def.)
  double    Q;    // Convexity indicator
  double    nu;   // Tension value
  double    ti;   // Knot spacing value
  double    hi;   // Step parameter (h[i] = t[i] - t[i-1])

public:
  SplineNode();
  SplineNode(Vec, double);

  Vec     GetP();
  Vec     GetL();
  Vec     GetR();
  Vec     GetD();
  double  GetQ();
  double  GetNu();
  double  GetTi();
  double  GetHi();

  void SetP(Vec);
  void SetL(Vec);
  void SetR(Vec);
  void SetD(Vec);
  void SetQ(double);
  void SetNu(double);
  void SetHi(double);
  void SetTi(double);

  void Print();
};

#endif
```

File: src/SplineNode.cpp

```
#include "SplineNode.hpp"

using namespace std;

typedef boost::numeric::ublas::vector<double> Vec;

SplineNode::SplineNode(){
  this->nu = 0;
}

SplineNode::SplineNode(Vec P, double nu_value) {
  this->P  = P;
  this->d  = P;
  this->nu = nu_value;
}

Vec     SplineNode::GetP()                    { return this->P;   }
Vec     SplineNode::GetL()                    { return this->L;   }
Vec     SplineNode::GetR()                    { return this->R;   }
Vec     SplineNode::GetD()                    { return this->d;   }
double  SplineNode::GetQ()                    { return this->Q;   }
double  SplineNode::GetNu()                   { return this->nu;  }
double  SplineNode::GetTi()                   { return this->ti;  }
double  SplineNode::GetHi()                   { return this->hi;  }

void SplineNode::SetNu(double val)            { this->nu = val;   }
void SplineNode::SetP(Vec P)                  { this->P  = P;     }
void SplineNode::SetL(Vec L)                  { this->L  = L;     }
void SplineNode::SetR(Vec R)                  { this->R  = R;     }
void SplineNode::SetD(Vec d)                  { this->d  = d;     }
void SplineNode::SetHi(double val)            { this->hi = val;   }
void SplineNode::SetQ(double Q)               { this->Q  = Q;     }
void SplineNode::SetTi(double val)            { this->ti = val;   }

void SplineNode::Print() {
  std::cout << "P = " << this->P << ", d = " << this->d << std::endl;
}
```

6.1.6 NuSpline

File: src/NuSpline.hpp

```
#ifndef __NUSPLINE_H_INCLUDED
#define __NUSPLINE_H_INCLUDED

#include <iostream>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/numeric/bindings/traits/ublas_vector2.hpp>
#include "SplineNode.hpp"
#include "Options.cpp"
#include "Utility.hpp"

using namespace std;

class NuSpline {
    ublas::vector<double> F(std::vector<SplineNode> nodes, int i);
    double ComputeError(mp_type, mp_type  , int);
    double alpha (std::vector<SplineNode> , int);
    double delta (std::vector<SplineNode> , int);
    double gamma (std::vector<SplineNode> , int);
    double beta  (std::vector<SplineNode> , int);
    double lambda(std::vector<SplineNode> , int);
    double mu    (std::vector<SplineNode> , int);

public:
    void Init   (std::vector<SplineNode> *, Options);
    void Update (std::vector<SplineNode> *);
    bool Execute(std::vector<SplineNode> *, int, double);

};

#endif
```

File: src/NuSpline.cpp

```
#include "NuSpline.hpp"

typedef ublas::vector<double>      Vec;
typedef std::vector<SplineNode>    SVec;
typedef std::vector< Vec >         mp_type;

// Initialization method. Takes care of setting initial
// values for quantities needed in the computation of
// the nu-spline. All the info relative to a node is
// stored in the corresponding member of the vector.
void NuSpline::Init(SVec *nodes, Options opt) {

  int N = (*nodes).size();

  // Set spacing and knot values

  (*nodes)[0].SetTi(0.0);
  (*nodes)[0].SetHi(0.0);

  for (int i = 1; i < N - 1; i++) {
    (*nodes)[i].SetHi( GeoDist( (*nodes)[i-1].GetP(), (*nodes)[i].GetP() ) );
    (*nodes)[i].SetTi( (*nodes)[i-1].GetTi() + (*nodes)[i].GetHi() );
  }

  (*nodes)[N-1].SetHi( GeoDist( (*nodes)[N-1].GetP(), (*nodes)[N-2].GetP() ) );
  (*nodes)[N-1].SetTi( (*nodes)[N-2].GetTi() + (*nodes)[N-1].GetHi() );

  // Set Left and Right auxiliary points

  (*nodes)[0].SetR( G((*nodes)[0].GetD(), (*nodes)[1].GetD(), mu((*nodes), 0)) );
  for (int i = 1; i < N-1; i++) {
    (*nodes)[i].SetL ( G( (*nodes)[i-1].GetD(), (*nodes)[i].GetD(), lambda((*nodes), i) ) );
    (*nodes)[i].SetR ( G( (*nodes)[i].GetD(), (*nodes)[i+1].GetD(), mu((*nodes), i) ) );
  }
  (*nodes)[N-1].SetL ( G( (*nodes)[N-2].GetD(), (*nodes)[N-1].GetD(), lambda((*nodes), N-1) ) );


  // Set convexity indicators Qi

  for (int i = 1; i < N - 1; i++) {
    Vec Ll = dG((*nodes)[i-1].GetP(), (*nodes)[i].GetP(), 1.0);
    Vec Lr = dG((*nodes)[i].GetP(), (*nodes)[i+1].GetP(), 0.0);
    Vec cr = CrossProduct(Ll, Lr);
    double q = ublas::inner_prod((*nodes)[i].GetP(), cr) / ublas::norm_2(cr);

    if (fabs(fabs(q) - 1.0) < opt.normal_tol) {
      if (q < 0.0) {
        (*nodes)[i].SetQ( -1.0 );
      } else {
        (*nodes)[i].SetQ( 1.0 );
      }
    } else {
      (*nodes)[i].SetQ( 0.0 );
    }
  }

}


// This procedure updates the auxiliary values Li and Ri
// for each SplineNode. This has to be done at each iteration,
// otherwise the progress made by the algorithm is lost.
void NuSpline::Update(SVec *nodes) {
```

```
  int N = (*nodes).size();
  (*nodes)[0].SetR( G((*nodes)[0].GetD(), (*nodes)[1].GetD(), mu((*nodes), 0)) );
  for (int i = 1; i < N-1; i++) {
    (*nodes)[i].SetL ( G( (*nodes)[i-1].GetD(), (*nodes)[i].GetD(), lambda((*nodes), i) ) );
    (*nodes)[i].SetR ( G( (*nodes)[i].GetD(), (*nodes)[i+1].GetD(), mu((*nodes), i) ) );
  }
  (*nodes)[N-1].SetL ( G( (*nodes)[N-2].GetD(), (*nodes)[N-1].GetD(), lambda((*nodes), N-1) ) );
}


// Compute the quantity alpha_i
double NuSpline::alpha(SVec nodes, int i) {
  return GeoDist(nodes[i-1].GetD(), nodes[i].GetD());
}

// Compute the quantity delta_i
double NuSpline::delta(SVec nodes, int i) {
  assert(i > 0 && i < nodes.size()-1);

  double num = nodes[i].GetHi();
  double den = num + nodes[i+1].GetHi();

  double val = num/den;

  assert(!(val < 0.0 || val > 1.0));

  return val;
}


// Compute the quantity beta_i
double NuSpline::beta(SVec nodes, int i) {
  return GeoDist(nodes[i].GetL(), nodes[i].GetR());
}


// Compute the quantity gamma_i
double NuSpline::gamma(SVec nodes, int i) {
  double hi = nodes[i].GetHi();
  double hi1 = nodes[i+1].GetHi();
  double nu = nodes[i].GetNu();
  double val = ( 2 * (hi + hi1) ) / ( nu*hi*hi1 + 2*(hi + hi1) );

  assert(!(val < 0.0 || val > 1.0));

  return val;
}


// Compute the quantity lambda_i
double NuSpline::lambda(SVec nodes, int i) {
  double num = gamma(nodes, i-1)*nodes[i-1].GetHi() + nodes[i].GetHi();
  double den = num;

  // CAUTION: this is the point at which we take into account
  // the fact that the indices for h_i go up until n+1.
  // If the indices are within acceptable range, we add the
  // second part of the denominator, otherwise we just leave
  // it as is. This is consistent with the theory, as h_{n+1} = 0
  // but needs to be handled, as it leads to severe numerical
  // errors if not.
  if (i < (nodes.size() - 1)) {
    den = den + (gamma(nodes, i)*nodes[i+1].GetHi());
  }

```

```
  double val = num/den;

  assert(!(val < 0.0 || val > 1.0));

  return val;
}


// Compute the quantity mu_i
double NuSpline::mu(SVec nodes, int i) {
  double num = gamma(nodes, i)*nodes[i].GetHi();
  double den = num + nodes[i+1].GetHi();

  // CAUTION: this is the point at which we take into account
  // the fact that the indices for h_i go up until n+1.
  // If the indices are within acceptable range, we add the
  // second part of the denominator, otherwise we just leave
  // it as is. This is consistent with the theory, as h_{n+1} = 0
  // but needs to be handled, as it leads to severe numerical
  // errors if not.
  if (i < (nodes.size()-2)) {
    den = den + gamma(nodes, i+1)*nodes[i+2].GetHi();
  }

  double val = num/den;

  assert(!(val < 0.0 || val > 1.0));

  return val;
}


// Execute an iteration of the approximating scheme as described
// in Nielson. Essentially, calculate the successive value for
// the control point d_i from the previous values of the control
// points d_{i-1}, d_i, d_{i+1}
Vec NuSpline::F(SVec nodes, int i){
  double ai   =   alpha(nodes, i);
  double ai1  =   alpha(nodes, i+1);
  double di   =   delta(nodes, i);
  double bi   =   beta(nodes, i);
  double li   =   lambda(nodes, i);
  double mi   =   mu(nodes, i);

  double num1 = sin(bi);
  double num2 = ( sin( (1.0-di)*bi ) * sin( (1.0-li)*ai ) ) / sin(ai);
  double num3 = ( sin( di*bi ) * sin( mi*ai1 ) ) / sin(ai1);
  double den  = ( sin( (1.0-di)*bi ) * sin( li*ai ) )  / sin(ai) + \
                ( sin( di*bi ) * sin( (1.0-mi)*ai1 ) ) / sin(ai1);

  Vec qi      = nodes[i].GetP();
  Vec d_prev  = nodes[i-1].GetD();
  Vec d_next  = nodes[i+1].GetD();

  qi      =  (num1/den)*qi;
  d_prev  = -(num2/den)*d_prev;
  d_next  = -(num3/den)*d_next;

  Vec f(3);
  f = qi + d_prev + d_next;

  return f;
}
```

```
// Compute the error (difference) between successive approximations.
// This is a form of relative error measuring the change from one step
// to the next. If the difference is too small, we need not continue,
// as convergence has been manifested and successive steps do not offer
// significant improvements. Similar measure is used in fixed-point methods.
double NuSpline::ComputeError(mp_type nd, mp_type od, int N){

  double num, den, r;

  num = 0.0;
  den = 0.0;

    for (int i = 0; i < N; i++) {
      Vec cur = nd[i];
      Vec pre = od[i];
      Vec np = cur - pre;

      num = num + ublas::inner_prod(np, np);
      den = den + ublas::inner_prod(cur, cur);
    }

    r = sqrt(num/den);
  return r;
}


// This method executes the algorithm described in Nielson. Before
// calling this, one should call Init() so that the necessary values
// are present. When the procedure ends, all pertaining values are
// stored in the vector of SplineNodes, and the boolean value returned
// indicates whether convergence has been achieved or not.
bool NuSpline::Execute(SVec *nodes, int Nmax, double tol) {
  cout << "Executing now!" << endl;
  int iter    = 0;
  double err  = 10.0*tol;   // Just consider something big enough
  int N       = (*nodes).size();


  // Compy the current control points into a suitable container
  mp_type ctrd;
  for (int i = 0; i < N; i++) {
    ctrd.push_back((*nodes)[i].GetD());
  }

  // Keep another copy (needed to compute the error)
  mp_type oldd = ctrd;

  // Iterate while convergence is not reached and iterations
  // are within a reasonable range
  while (err > tol && iter < Nmax) {
    // Keep the old estimate
    oldd = ctrd;

    // For each point, compute a new estimate for the control point
    for (int i = 1; i < N-1; i++) {
      ctrd[i] = F(*nodes, i);
    }

    // Set the new estimate at each node
    for (int i = 1; i < N-1; i++) {
      (*nodes)[i].SetD(ctrd[i]);
    }

    // Update the nodes appropriately: since the control poitns
    // have changed, the auxiliary values Li and Ri need to
```

```
    // change as well.
    Update(nodes);

    // Increment counter and compute the error
    iter = iter + 1;
    err  = ComputeError(ctrd, oldd, N);
  }

  // Tell the calling function whether the result is OK
  return (err < tol);

}
```

6.1.7 ShapePreservation

File: src/ShapePreservation.hpp

```
#ifndef __SHAPEPRESERVATION_H_DEFINED
#define __SHAPEPRESERVATION_H_DEFINED

#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/numeric/bindings/traits/ublas_vector2.hpp>
#include "Options.cpp"
#include "FiniteDifference.hpp"
#include "Utility.hpp"

namespace ublas  = boost::numeric::ublas;

typedef ublas::vector<double>     Vec;
typedef std::vector<SplineNode>   SVec;

void   VerifyCocircularity    (Vec (*f)(SVec, int, double), SVec, bool *, Options *);
void   VerifyNodeConvexity   (Vec (*f)(SVec, int, double), SVec, bool *, Options *);
void   VerifySegmentConvexity(Vec (*f)(SVec, int, double), SVec, bool *, Options *);
double ObjFun                (Vec (*f)(SVec, int, double), SVec, int, int, Options *, double);
double DerivObjFun           (Vec (*f)(SVec, int, double), SVec, int, int, Options *, double);
bool   ExamineSegment        (Vec (*f)(SVec, int, double), SVec, int, Options *);

#endif
```

File: src/ShapePreservation.cpp

```
#include "ShapePreservation.hpp"

namespace ublas  = boost::numeric::ublas;
typedef ublas::vector<double> Vec;


// Verify that the co-circularity criterion is satisfied at the nodes
// for which it is required. For every node i for which the criterion
// fails, the elements {i-1, i, i+1} of the boolean vector idx are
// set true. This is to indicate that the nu-values of these nodes
// need to be incremented by the corresponding function (not here).
void VerifyCocircularity(Vec (*fun)(std::vector<SplineNode>, int, double),
            std::vector<SplineNode> nodes,
            bool *idx,
            Options *opt){

  for (int i = 1; i < nodes.size()-1; i++) {
    if (nodes[i].GetQ() == 0.0) {
      double d1 = GeoDist(nodes[i-1].GetP(), nodes[i].GetP());
      double d2 = GeoDist(nodes[i].GetP(), nodes[i+1].GetP());
      double d3 = GeoDist(nodes[i-1].GetP(), nodes[i+1].GetP());
      if ((d3 > d1) && (d3 > d2)) {
        double kk = kappa(fun, nodes, i, opt, 0);
        if (fabs(kk) > opt->epsilon) {
          idx[i-1] = true;
          idx[ i ] = true;
          idx[i+1] = true;
        }
      }
    }
  }
}



// Verify that the node convexity criterion is satisfied.
// For every node i for which the criterion fails, the elements
// {i-1, i, i+1} of the boolean vector idx are set true.
// This is to indicate that the nu-values of these nodes
// need to be incremented by the corresponding function (not here).
void VerifyNodeConvexity(Vec (*fun)(std::vector<SplineNode>, int, double),
             std::vector<SplineNode> nodes,
             bool *idx,
             Options *opt){
  for (int i = 0; i < nodes.size()-1; i++) {
    if (nodes[i].GetQ() != 0.0) {
      double kk = kappa(fun, nodes, i, opt, 0.0);
      if (kk*nodes[i].GetQ() < 0.0) {
        idx[i-1] = true;
        idx[ i ] = true;
        idx[i+1] = true;
      }
    }
  }
}



// Verify that the segment convexity criterion is satisfied.
// For every node i for which the criterion fails, the elements
// {i-1, i, i+1, i+2} of the boolean vector idx are set true.
// This is to indicate that the nu-values of these nodes
// need to be incremented by the corresponding function (not here).
```

```
void VerifySegmentConvexity(Vec (*fun)(std::vector<SplineNode>, int, double),
              std::vector<SplineNode> nodes,
              bool *idx,
              Options *opt){

  // Iterate through the internal nodes (note that node n-1 is examined in segment n-2)
  for (int i = 1; i < nodes.size() - 2; i++) {
    // Neither of the end-nodes of the segment i can be co-circular with their neighbors
    if (nodes[i].GetQ()*nodes[i+1].GetQ() > 0.0) {
      // For each segment that passes the controls, check the convexity of the spline
      bool condition = ExamineSegment(fun, nodes, i, opt);
      // In the case that the control fails, mark the indices
      if (!condition) {
        idx[i-1] = true;
        idx[ i ] = true;
        idx[i+1] = true;
        idx[i+2] = true;
      }
    }
  }
}




// Examine the convexity of segment i.
// If this function is invoked, all checks for the segment i have
// already been verified by the calling function.
bool ExamineSegment(Vec (*fun)(std::vector<SplineNode>, int, double),
          std::vector<SplineNode> nodes,
          int i,
          Options *opt){

  // Initial interval (for t) in which we check convexity
  double a = 0.0;
  double b = 1.0;

  // Sentinel values
  bool c1 = false;
  bool c2 = false;
  bool condition = false;

  // Here we are examining the segment convexity predicate involving
  // the binormal Q_i.

  // Objective function values at the ends of the interval [a, b].
  // The fourth argument of the below functions defines the use
  // of the i-th binormal.
  double fa = ObjFun(fun, nodes, i, i, opt, a);
  double fb = ObjFun(fun, nodes, i, i, opt, b);

  // If the values of the objective function at both ends is positive,
  // we can continue with the control at the internal of the interval.
  // Otherwise, there is no point: the criterion has already failed.
  if ((fa > 0.0) && (fb > 0.0)) {

    // Implement a simple bisection search for the root
    // of the derivative of the objective finction.
    // The root of the derivative provides the location
    // of an extremum for the objective function.
    int N = 0;
    double root = (a + b) / 2.0;
    while (N < opt->Bnmax) {
      double m = (a + b) / 2.0;
      double dfm = DerivObjFun(fun, nodes, i, i, opt, m);
```

```
      if ( dfm == 0.0 || ((b-a) / 2.0) < opt->Btol) {
        root = m;
        break;
      }

      double dfa = dfm = DerivObjFun(fun, nodes, i, i, opt, a);
      if (dfm * dfa > 0) {
        a = m;
      } else {
        b = m;
      }
      N++;
    }

    // Get the value of the objective function at its extremum.
    double fr = ObjFun(fun, nodes, i, i, opt, root);

    // If this value is positive, assume the function is
    // positive in the entire interval [a, b].
    if (fr > 0.0) {
      c1 = true;
    }
  }



  // Repeat the same procedure, but now for the binormal Q_{i+1}

  a = 0.0;
  b = 1.0;

  // The fourth argument in the below functions defines
  // the use of the (i+1)-th binormal.
  fa = ObjFun(fun, nodes, i, i+1, opt, a);
  fb = ObjFun(fun, nodes, i, i+1, opt, b);

  if ((fa > 0.0) && (fb > 0.0)) {
    int N = 0;
    double root = (a + b) / 2.0;
    while (N < opt->Bnmax) {
      double m = (a + b) / 2.0;
      double dfm = DerivObjFun(fun, nodes, i, i+1, opt, m);
      if ( dfm == 0.0 || ((b-a) / 2.0) < opt->Btol) {
        root = m;
        break;
      }

      double dfa = dfm = DerivObjFun(fun, nodes, i, i+1, opt, a);
      if (dfm * dfa > 0) {
        a = m;
      } else {
        b = m;
      }
      N++;
    }

    // Same logic as before
    double fr = ObjFun(fun, nodes, i, i+1, opt, root);
    if (fr > 0.0) {
      c2 = true;
    }
  }

  // If both sentinel values are true, the criterion is satisfied.
  condition = (c1 && c2);
```

```
  return condition;
}



// Objective function.
// By definition, product of the geodesic curvature with the
// convexity indicator at each node.
double ObjFun( Vec (*fun)(std::vector<SplineNode>, int, double),
      std::vector<SplineNode> nodes,
      int node_index,     // Essentially, segment index
      int ind_index,      // Indicator index
      Options *opt,
      double t) {

  double rr = kappa(fun, nodes, node_index, opt, t) * nodes[ind_index].GetQ();
  return rr;
}



// Objective function derivative.
// By definition, product of the derivative of the geodesic
// curvature with the convexity indicator at each node.
double DerivObjFun(Vec (*fun)(std::vector<SplineNode>, int, double),
        std::vector<SplineNode> nodes,
        int node_index,
        int ind_index,
        Options *opt,
        double t) {

  double rr = dkappa(fun, nodes, node_index, opt, t) * nodes[ind_index].GetQ();
  return rr;
}
```

6.1.8 FiniteDifference

File: src/FiniteDifference.hpp

```
#ifndef __FINITEDIFFERENCE_H_INCLUDED
#define __FINITEDIFFERENCE_H_INCLUDED

#include <iostream>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/numeric/bindings/traits/ublas_matrix.hpp>
#include <boost/numeric/bindings/lapack/gesv.hpp>
#include <boost/numeric/bindings/traits/ublas_vector2.hpp>
#include "SplineNode.hpp"

#define _USE_MATH_DEFINES
#include <math.h>

namespace ublas  = boost::numeric::ublas;
namespace lapack = boost::numeric::bindings::lapack;

typedef ublas::vector<int>       iVec;
typedef ublas::vector<double>     Vec;
typedef std::vector<SplineNode>   SVec;

class FiniteDifference {
  private:
    int     p;    // Accuracy order
    int     d;    // Derivative order
    int     iMax; // Maximum index
    int     iMin; // Minimum index
    string  type; // Type of scheme (fwd, bwd or ctr)
    iVec    idx;  // Indices (or multiples) of h
    Vec     C;    // Coefficients for the sum of auxiliary terms

    // Simple implementation of the factorial function.
    // Convention has been made to consider the factorial
    // of 0 and negative integers equal to 1. In reality,
    // no negative numbers are passed as input, but...
    // better safe than sorry.
    int Factorial(int);

  public:
    // The constructor calculates everything necessary
    // to compute the derivative of a given function
    FiniteDifference(int, int, string);

    // Computes the approximation of the derivative of the function
    // f, passed as argument here. The order of the derivative, as
    // well as the order of approximation, depend on the parameters
    // with which the instance of the class has been initialized.
    Vec Val(Vec (*f)(SVec, int, double), SVec, int, double, double);
};

#endif
```

File: src/FiniteDifference.cpp

```
#include "FiniteDifference.hpp"

namespace ublas  = boost::numeric::ublas;
namespace lapack = boost::numeric::bindings::lapack;

typedef ublas::vector<int>       iVec;
typedef ublas::vector<double>     Vec;
typedef std::vector<SplineNode>   SVec;


// Simple implementation of the factorial function.
// Convention has been made to consider the factorial
// of 0 and negative integers equal to 1. In reality,
// no negative numbers are passed as input, but...
// better safe than sorry.
int FiniteDifference::Factorial(int x) {
  return ( x < 2 ? 1 : ( x * Factorial(x-1) ) );
}

// The constructor calculates everything necessary
// to compute the derivative of a given function
FiniteDifference::FiniteDifference(int d, int p, string type) {

  // Keep parameters for later use
  this->p = p;
  this->d = d;
  this->type = type;

  // Set index limits based on scheme mode
  if (type.compare("fwd") == 0) {
    iMin = 0;
    iMax = d+p-1;
  } else if (type.compare("bwd") == 0) {
    iMax = 0;
    iMin = -(d+p-1);
  } else {
    iMax = floor((d+p-1)/2);
    iMin = -iMax;
  }

  int N = d + p;

  // Construct a matrix and a vector to express the problem
  //    AC = b
  // where C are the coefficients we seek.
  ublas::matrix<double, ublas::column_major> A(N,N);
  Vec b(N);

  // idx keeps the multiples of h used for each term
  // of the Taylor sum (i.e. -1h, 0h, 1h etc)
  idx = iVec(N);

  int ii = 0;
  int jj = 0;

  // Construct the matrix A
  for (int i = iMin; i <= iMax; i++) {
    idx(ii) = i;
    for (int j = 0; j <= N-1; j++) {
      A(jj++, ii) = pow((float)i, (float)j);
    }
    ii++;
    jj = 0;
```

```
  }

  // Fill the vector b with zeros...
  for (int i = 0; i < N; i++) {
    b(i) = 0.0;
  }
  // ..and only set the appropriate element to be unit.
  b(d) = 1.0;

  // Solve the linear system (the solution is stored in b)
  lapack::gesv(A, b);
  // Keep the solution on the desired vector
  C = b;
}


// Computes the approximation of the derivative of the function
// f, passed as argument here. The order of the derivative, as
// well as the order of approximation, depend on the parameters
// with which the instance of the class has been initialized.
Vec FiniteDifference::Val(Vec (*fun)(SVec, int, double),
    SVec nodes,
    int index,
    double t,
    double h) {

  int N = p + d;

  // Compute 'time steps' at which to compute the
  // auxiliary terms of the Taylor sum
  Vec tt(N);
  for (int i = 0; i < N; i++) {
    tt(i) = h*idx(i) + t;
  }

  Vec s(3);
  s(0) = 0.0; s(1) = 0.0; s(2) = 0.0;

  // Compute each term of the function at the appropriate
  // time and compute the weighted sum of the terms
  for (int i = 0; i < N; i++) {
    Vec v = (*fun)(nodes, index, tt(i));
    s = s + v*C(i);
  }

  // We have missed essential parts: the factorial
  // and some terms of h, so we need to update our
  // approximation. We use double precision in order
  // not to lose accuracy and avoid truncation errors
  double fd = (double)Factorial(d);
  double hd = (double)pow(h, d);

  s = (fd/hd)*s;

  return s;
}
```

6.1.9 Utility

File: src/Utility.hpp

```

#ifndef __UTILITY_H_INCLUDED
#define __UTILITY_H_INCLUDED

#include <iostream>
#include <string>
#include <fstream>
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/point_xy.hpp>
#include <boost/geometry/multi/geometries/multi_point.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/numeric/bindings/traits/ublas_vector2.hpp>
#include "Options.cpp"
#include "SplineNode.hpp"
#include "FiniteDifference.hpp"

#define _USE_MATH_DEFINES
#include <math.h>

namespace ublas = boost::numeric::ublas;
namespace bgeom = boost::geometry;

typedef ublas::vector<double>      Vec;
typedef std::vector<SplineNode>    SVec;
typedef std::vector< Vec >         mp_type;

// Typedef for spherical equatorial points.
typedef boost::geometry::model::point<
   double, 2, boost::geometry::cs::spherical_equatorial<boost::geometry::radian>
> spherical_point;

// Typedef for cartesian points.
typedef boost::geometry::model::point<
    double, 3, boost::geometry::cs::cartesian
> cartesian_point;

// Convesion of spherical points to cartesian coordinate system
template <typename SphericalPoint>
cartesian_point
Spherical2Cartesian(SphericalPoint const &);

// Conversion of cartesian points to spherical-equatorial system
template <typename CartesianPoint>
spherical_point
Cartesian2Spherical(CartesianPoint const &);

// Geodesic Distance (or distace on the sphere)
double GeoDist(Vec, Vec);

// Read options from external text file and return an object
void ReadOptions(char *, Options *);

// Read input points from text file, assuming spherical-equatorial coordinates
void ReadSphericalPoints(std::string, mp_type *);

// Read input points from text file, assuming cartesian coordinates
void ReadCartesianPoints(std::string, mp_type *);

// Geodesic on the sphere between points p and q at 'time' t
```

```
Vec G(Vec, Vec, double);

// Derivative of the geodesic between p and q on the sphere, at 'time' t
Vec dG(Vec, Vec, double);

// Simple cross-product. No ready solution was found.
Vec CrossProduct(Vec, Vec);

// Accept a boolean vector and return true if at least one element is True
bool Verify(const bool *, int);

// Augment the tension values that should be augmented
void AugmentNuValues(SVec *, const bool *, Options *);

// Compute a point on the Spline on segment i, for 'local'
// parameter value equal to t
Vec Spline(SVec, int, double);

// Normalizing factor for the finite difference approximations
// of the derivatives of the spline curve.
double fact(SVec, int, int);

// Geodesic curvature
double kappa(Vec (*fun)(SVec, int, double), SVec, int, Options *, double);

// Derivative of the geodesic curvature
double dkappa(Vec (*fun)(SVec, int, double), SVec, int, Options *, double);

#endif
```

File: src/Utility.cpp

```
#include "Utility.hpp"

namespace ublas = boost::numeric::ublas;
namespace bgeom = boost::geometry;

typedef ublas::vector<double>     Vec;
typedef std::vector<SplineNode>   SVec;
typedef std::vector< Vec >        mp_type;

// Typedef for spherical equatorial points.
typedef boost::geometry::model::point<
   double, 2, boost::geometry::cs::spherical_equatorial<boost::geometry::radian>
> spherical_point;

// Typedef for cartesian points.
typedef boost::geometry::model::point<
    double, 3, boost::geometry::cs::cartesian
> cartesian_point;


// Tension value increase function -- signature only!
double f(double);


// Convesion of spherical points to cartesian coordinate system
template <typename SphericalPoint>
cartesian_point
Spherical2Cartesian(SphericalPoint const& spherical_point)
{

    double lon = bgeom::get_as_radian<0>(spherical_point);
    double lat = bgeom::get_as_radian<1>(spherical_point);

    double x = cos(lat)*cos(lon);
    double y = cos(lat)*sin(lon);
    double z = sin(lat);

    return cartesian_point(x, y, z);
}


// Conversion of cartesian points to spherical-equatorial system
template <typename CartesianPoint>
spherical_point
Cartesian2Spherical(CartesianPoint const& cartesian_point)
{
    double x = bgeom::get<0>(cartesian_point);
    double y = bgeom::get<1>(cartesian_point);
    double z = bgeom::get<2>(cartesian_point);

    double lat = atan2(z, sqrt(x*x+y*y));
    double lon = atan2(y, x);

    return spherical_point(lon, lat);
}


// Geodesic Distance (or distace on the sphere)
double GeoDist(Vec x, Vec y) {
  double val = ublas::inner_prod(x, y)/(ublas::norm_2(x)*ublas::norm_2(y));
  // The argument of arccos must be in the interval [-1, 1]
  val = (val < -1.0 ? -1.0 : (val > 1.0 ? 1.0 : val));
  double theta = acos(val);
```

```
  return theta;
}


// Read options from external text file and return an object
void ReadOptions(char* filename, Options *opt) {

    std::map<std::string, std::string> *options = new std::map<std::string, std::string>();
    std::ifstream f;
    std::string key, value;

    f.open(filename);
    while (f >> key >> value) {
      (*options)[key] = value;
    }
    f.close();

    opt->input_file      = (*options)["input_file"];
    opt->input_type      = (*options)["input_type"];
    opt->increase_type   = (*options)["increase_type"];
    opt->increase_factor = atof((*options)["increase_factor"].c_str());
    opt->initial_nu      = atof((*options)["initial_nu"].c_str());
    opt->Stol            = atof((*options)["Stol"].c_str());
    opt->epsilon         = atof((*options)["epsilon"].c_str());
    opt->normal_tol      = atof((*options)["normal_tol"].c_str());
    opt->h               = atof((*options)["h"].c_str());
    opt->Btol            = atof((*options)["Btol"].c_str());
    opt->Bnmax           = atoi((*options)["Bnmax"].c_str());
    opt->Snmax           = atoi((*options)["Snmax"].c_str());

}


// Read input points from text file, assuming spherical-equatorial coordinates
void ReadSphericalPoints(std::string filename, mp_type *pts) {
    std::ifstream f;
    f.open(filename.c_str());
    double phi, theta;
    while (f >> phi >> theta) {
      cartesian_point cp = Spherical2Cartesian(spherical_point(phi, theta));
      Vec v(3);
      v(0) = cp.get<0>(); v(1) = cp.get<1>(); v(2) = cp.get<2>();
      pts->push_back(v);
    }
    f.close();

    return;
}


// Read input points from text file, assuming cartesian coordinates
void ReadCartesianPoints(std::string filename, mp_type *pts) {
    std::ifstream f;
    f.open(filename.c_str());
    double x, y, z;
    while (f >> x >> y >> z) {
      Vec v(3);
      v(0) = x; v(1) = y; v(2) = z;
      pts->push_back(v);
    }
    f.close();

    return;
}

```

```
// Geodesic on the sphere between points p and q at 'time' t
Vec G(Vec p, Vec q, double t) {
  assert(!(t < 0.0 || t > 1.0));

  double theta = GeoDist(p, q);
  if (theta < std::numeric_limits<double>::epsilon()) {
    return p;
  }

  Vec pt = (sin((1.0-t)*theta)/sin(theta))*p + (sin(t*theta)/sin(theta))*q;
  return pt;
}


// Derivative of the geodesic between p and q on the sphere, at 'time' t
Vec dG(Vec p, Vec q, double t) {

  assert(!(t < 0.0 || t > 1.0));

  double theta = GeoDist(p, q);
  Vec v = (-theta*cos((1.0-t)*theta)/sin(theta))*p + (theta*cos(t*theta)/sin(theta))*q;

  return v;
}


// Simple cross-product. No ready solution was found.
Vec CrossProduct(Vec x, Vec y) {

  Vec cp(3);

  cp(0) = x(1)*y(2) - x(2)*y(1);
  cp(1) = x(2)*y(0) - x(0)*y(2);
  cp(2) = x(0)*y(1) - x(1)*y(0);

  return cp;
}


// Accept a boolean vector and return true if at least one element is True
bool Verify(const bool *idx, int N) {
  bool b = false;
  for (int i = 0; i < N; i++) {
    b = (b || idx[i]);
  }

  return b;
}


// Increase the tension values that should be increased
void AugmentNuValues(SVec *nodes, const bool *idx, Options *opt) {
  for (int i = 0; i < (*nodes).size(); i++) {
    if (idx[i]) {
      if (opt->increase_type.compare("plus") == 0) {
        (*nodes)[i].SetNu( (double)(*nodes)[i].GetNu() + (double)opt->increase_factor );
      } else {
        (*nodes)[i].SetNu( (double)(*nodes)[i].GetNu() * (double)opt->increase_factor );
      }
    }
  }
}
```

```
// Compute a point on the Spline on segment i, for 'local'
// parameter value equal to t
Vec Spline(SVec nodes, int i, double t) {

  // Initial quantities
  Vec Pi  = nodes[i].GetP();
  Vec Ri  = nodes[i].GetR();
  Vec Li1 = nodes[i+1].GetL();
  Vec Pi1 = nodes[i+1].GetP();

  // First order de Casteljau
  Vec x = G(Pi, Ri, t);
  Vec y = G(Ri, Li1, t);
  Vec z = G(Li1, Pi1, t);

  // Second order de Casteljau
  Vec u = G(x, y, t);
  Vec w = G(y, z, t);

  // Third order de Casteljau
  Vec p = G(u, w, t);

  return p;
}



// Normalizing factor for the finite difference approximations
// of the derivatives of the spline curve.
double fact(SVec nodes, int i, int d) {

  // Consider the length of the first segment to be 'reference'.
  double dref = GeoDist(nodes[0].GetP(), nodes[ 1 ].GetP());
  // Compute the length of the current segment.
  double dcur = GeoDist(nodes[i].GetP(), nodes[i+1].GetP());
  // Compute the factor and return it.
  double v = dref / pow(dcur, (double)d);
  return v;
}



// Geodesic curvature
double kappa(Vec (*fun)(SVec, int, double), SVec nodes, int i, Options *opt, double t) {

  // Automatically decide which scheme type to use.
  // For simplicity, use the same type for both the first
  // and second derivative.
  string type = "ctr";
  if ((t + 5.0*opt->h) > 1.0) {
    type = "bwd";
  }
  if ((t - 5.0*opt->h) < 0.0) {
    type = "fwd";
  }

  FiniteDifference *fd1 = new FiniteDifference(1, 4, type);
  FiniteDifference *fd2 = new FiniteDifference(2, 4, type);

  // Normalize the derivatives by using the corresponding factors
  Vec dS  = fact(nodes, i, 1) * fd1->Val(fun, nodes, i, t, opt->h);
  Vec ddS = fact(nodes, i, 2) * fd2->Val(fun, nodes, i, t, opt->h);

  double v = ublas::inner_prod( fun(nodes, i, t), CrossProduct(dS, ddS) );

```

```
  return v;
}



// Derivative of the geodesic curvature
double dkappa(Vec (*fun)(SVec, int, double), SVec nodes, int i, Options *opt, double t) {

  // Automatically decide which scheme type to use.
  // For simplicity, use the same type for both the first
  // and third derivative.
  string type = "ctr";
  if ((t + 6.0*opt->h) > 1.0) {
    type = "bwd";
  }
  if ((t - 6.0*opt->h) < 0.0) {
    type = "fwd";
  }

  FiniteDifference *fd1 = new FiniteDifference(1, 4, type);
  FiniteDifference *fd3 = new FiniteDifference(3, 4, type);

  // Normalize the derivatives by using the corresponding factors
  Vec dS  = fact(nodes, i, 1) * fd1->Val(fun, nodes, i, t, opt->h);
  Vec ddS = fact(nodes, i, 3) * fd3->Val(fun, nodes, i, t, opt->h);

  double v = ublas::inner_prod( fun(nodes, i, t), CrossProduct(dS, ddS) );

  return v;
}
```

Bibliography

[1] Menelaos I. Karavelas and Panagiotis D. Kaklis. Spatial Shape-preserving Interpolation Using ν-Splines. *Numerical Algorithms*, 23(2-3):217–250, 2000.

[2] Gregory M. Nielson. ν-Quaternion Splines for the Smooth Interpolation of Orientations. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):224–229, March/April 2004.

[3] Panagiotis D. Kaklis. Personal communication.

[4] Panagiotis D. Kaklis and Menelaos I. Karavelas. Shape-preserving interpolation in $\mathbb{R}^3$. *IMA Journal of Numerical Analysis*, 17(3):373–419, June 1997.

[5] Panagiotis D. Kaklis and Nickolas S. Sapidis. Convexity-preserving interpolatory parametric splines of non-uniform polynomial degree. *Comput. Aided Geom. Des.*, 12(1):1–26, February 1995.

[6] Panagiotis D. Kaklis and Dimitrios G. Pandelis. Convexity-preserving polynomial splines of non-uniform degree. *IMA Journal of Numerical Analysis*, 10(2):223–234, 1990.

[7] Gerald E. Farin. *Curves and Surfaces for CAGD (5th edition)*. Academic Press, 2002.

[8] David F. Rogers. Chapter 1 - Curve and surface representation. In David F. Rogers, editor, *An Introduction to NURBS*, The Morgan Kaufmann Series in Computer Graphics, pages 1 – 15. Morgan Kaufmann, San Francisco, 2001.

[9] Wolfgang Boehm and Andreas Müller. On de Casteljau's algorithm. *Computer Aided Geometric Design*, 16(7):587 – 605, 1999.

[10] David Eberly. *Derivative Approximation by Finite Differences*. Geometric Tools, LLC, May 2001. http://www.geometrictools.com/Documentation/FiniteDifferences.pdf.