

# Motion Control in Computer-Aided Modeling

## Reality Check 5

Jenaveve White

### Reality Check

Jenaveve White

### Introduction

Computer-aided modeling and manufacturing play a crucial role in modern engineering and design, requiring precise spatial control along predefined motion paths. This project explores how Adaptive Quadrature can be applied to solve a fundamental problem within this field: the equipartition of an arbitrary path into subpaths of equal length. Maintaining constant speed along a path is essential for various applications, such as numerical machining, where equal increments ensure consistent material interaction, or motion planning in computer animation, where more complex velocity profiles may be necessary. In robotics and virtual reality, parametrized curves and surfaces are often needed to ensure smooth navigation. The project includes developing Python programs to compute arc lengths, determine parameters for path segmentation using numerical methods like the Bisection Method and Newton's Method, create visualizations for path equipartition, and demonstrate these concepts through Python animations. This practical approach will reinforce understanding of adaptive numerical techniques and their applications in engineering and computational modeling.

The parametric curve that will be modeled is

$$P(t) = \begin{cases} x(t) = 0.5 + 0.3t + 3.9t^2 - 4.7t^3 \\ y(t) = 1.5 + 0.3t + 0.9t^2 - 2.7t^3 \end{cases}$$

### Activity 1

Write a Python function that computes the arc length of the path  $P$  from  $t = 0$  to  $t = s$  for a given  $0 \leq s \leq 1$ .

## Procedure

In this code, a procedure was developed to compute the arc length of a parametric path defined by polynomial functions  $x(t)$  and  $y(t)$ . First, the functions  $dx\_dt(t)$  and  $dy\_dt(t)$  were implemented to calculate the derivatives of  $x$  and  $y$  with respect to  $t$ , representing the rate of change along the path. The magnitude of the velocity vector at each point  $t$ , computed using,

$$\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}$$

was defined in the `integrand()` function. To find the total arc length from a start point of 0 to a given endpoint `e`, the `quad()` function from `scipy.integrate` was used to numerically integrate the integrand over the specified interval. The resulting arc length is displayed below. This represents the distance traveled along the path from  $t=0$  to  $t=1$

The arc length from 0 to  $t=1$  is: 2.495246747514834

## Activity 2

Write a program that, for any input  $0 \leq s \leq 1$ , finds the parameter  $t(s)$  that is  $s$  of the way along the path. Use the Bisection Method to locate the point  $t(s)$  to three correct decimal places. What function is being set to zero? What bracketing interval should be used to start the Bisection Method?

## Procedure

The task involves dividing a parametric path  $(x(t), y(t))$ , defined by polynomial functions  $x(t)$  and  $y(t)$ , into subpaths of equal length. The code begins by calculating the arc length of the path using numerical integration, with the function `arc_length(e)` integrating the magnitude of the velocity vector.

$$\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}$$

To find the specific parameter  $t^*$  such that the path length from  $t=0$  to  $t^*$  equals  $s$  times the total path length, the Bisection Method is applied through the `bisect()` function. The function iteratively locates the root of an objective function that represents the difference between the arc length up to  $t^*$  and the target length. The function being set to zero is the arc length from 0 to the  $t^*$  value -  $s$  \* the total length. For the bracketing interval to start the Bisection Method `[0,1]` was used to ensure that  $f(b) * f(a) < 0$ .

## Activity 3

Equipartition the path of Figure 5.6 into  $n$  subpaths of equal length, for  $n=4$  and  $n=20$ . Plot analogues of Figure 5.6, showing the equipartitions.

### Procedure

The `equipar(n)` function uses this bisection method to find  $s$  values for different segments, dividing the path into  $n$  equal subpaths. Finally, the `plot_equipartitions(n, func)` function visualizes the path and its partition points using Plotly, showing the uniform segmentation. This approach facilitates applications like CNC machining, where consistent tool movement is crucial, and animation, where equal path segments ensure natural motion.

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

## Activity 4

Replace the Bisection Method in Step 2 with Newton's Method, and repeat Steps 2 and 3. What is the derivative needed? What is a good choice for the initial guess? Is computation time decreased by this replacement?

### Procedure

To solve the problem of finding the parameter  $t(s)$  such that the arc length from  $t = 0$  to  $t(s) = s$  \* the total path length, Newton's Method was used as an alternative to the Bisection Method. Newton's Method iteratively refines an initial guess  $x_0$  using the update formula  $x_{n+1}$  equals  $x_n$  minus  $f(x_n)$  divided by  $f'(x_n)$ . For this problem, the function  $f(t)$  equals  $\text{arc\_length}(t) - s$  \*  $\text{arc\_length}(1)$  was set to zero, where the derivative  $f'(t)$  is the magnitude of the derivative vector, computed as:

$$\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}$$

A good initial guess ensures convergence. A good choice for this initial guess is 0.5. Newton's Method reduces computation time compared to the Bisection Method because it converges more quickly when starting near the root, but it requires explicit computation of the derivative

$f'(t)$ . The code provided uses Newton's Method to find partition points for an arbitrary path, which are then visualized using Python's plotly library to show path segmentation into equal-length subpaths.

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

## Activity 5

Use Python animation commands to demonstrate traveling along the path, first at the original parameter  $0 \leq t \leq 1$  speed and then at the (constant) speed given by  $t^*(s)$  for  $0 \leq s \leq 1$ .

### Procedure

To animate traveling along a parametric path in Python at different speeds, we first prepare data arrays for path coordinates. This involves computing coordinates using functions like  $x(t)$  and  $y(t)$  and generating points based on uniform parameter values for original speed, as well as adaptive partitions  $s_i$  for constant-speed traversal. Using Matplotlib, the figure was created with subplots to display animations side by side, initialized with Line2D objects for movement representation. An `update()` function is defined to update plot data at each frame, and `animation.FuncAnimation()` is configured to animate the traversal by looping through frames of the data array. The animation showcases both traversal modes, enabling comparison and visualization of the path movement, helping to illustrate concepts in adaptive numerical methods and path segmentation.

## Activity 6

Experiment with equipartitioning a path of your choice. Choose a path defined by parametric equations (see Parametric Equation in Wikipedia for ideas), partition it into equal arc length segments, and animate as in Step 5.

## Procedure

To explore equipartitioning, a path is defined using parametric equations, specifically a circle parameterized by  $x(t) = \cos(t)$  and  $y(t) = \sin(t)$ . The process begins by computing the path's arc length through numerical integration, implemented in the `arc_length2()` function. To partition the path into equal arc length segments, the `equipar2()` function applies the Bisection Method to find parameter values  $t_i$  such that the arc length from  $t=0$  to  $t_i$  matches the target segment length. The `plot_equipartitions2()` function visualizes the segmented path by plotting the path and marking the partition points. For the animation, a data array of coordinates along the path is created, and a plot is initialized with Matplotlib. An `update()` function is defined to animate the traversal frame-by-frame using `animation.FuncAnimation()`. This demonstrates the visual transition along the path with constant-speed traversal, highlighting the effectiveness of numerical methods in engineering visualization.

Unable to display output for mime type(s): text/html