▷Project 6. Fundamental Theorem of Plane Curves

This Matlab project is concerned in part with the visualization and animation of curves. Before getting into the details of the project, I would like to make a few general remarks on the subject of mathematical visualization that you should keep in mind while working on this project—or for that matter when you have any programming task that involves visualization and animation of mathematical objects.

1) How should you choose an error tolerance?

First, an important principle concerning the handling of errors in any computer graphics context. Books on numerical analysis tell you how to estimate errors and how to keep them below a certain tolerance, but they cannot tell you what that tolerance should be—that must depend on how the numbers are going to be used. Beginners often assume they should aim for the highest accuracy their programming system can provide—for example fourteen decimal places for Matlab. But that will often be far more than is required for the task at hand, and as you have already seen, certain algorithms may require a very long time to attain that accuracy. The degree of one's patience hardly seems to be the best way to go about choosing an error tolerance.

In fact, there is often is a more rational way to choose appropriate error bounds. For example, in financial calculations it makes no sense to compute values with an error less than half the smallest denomination of the monetary unit involved. And when making physical calculations, it is useless to calculate to an accuracy much greater than can be measured with the most precise measuring instruments available. Similarly, in carpentry there is little point to calculating the length of a board to a tolerance less than the width of the blade that will make the cut.

This same principle governs in mathematical visualization. My approach is to choose a tolerance that is "about half a pixel", since any higher accuracy won't be visible anyway. It is usually fairly easy to estimate the size of a pixel. There are roughly 100 pixels per inch, so for example if you are are graphing in a six inch square window, and the axes go from minus one to one, then six hundred pixels equals two length units, so half a pixel accuracy means a tolerance of $\frac{1}{600}$ or roughly 0.002.

1) How should you represent a curve?

Mathematically a curve in $\mathbf{R}^n$ is given by a map of an interval $[a, b]$ into $\mathbf{R}^n$. We can only represent the curve on a computer screen when $n = 2$ or $n = 3$. Let's consider the case of plane curves ($n = 2$) first. If $\alpha(t) = (x(t), y(t))$ then for any N we can divide the interval $[a, b]$ into N equal subintervals of length $h = \frac{b-a}{N}$, namely $[t_k, t_{k+1}]$, where $t_k = a + kh$ and $k = 0, \ldots, N(-1$. We associate to α and N an approximating "N-gon" α_N (i.e., a polygon with N sides) with vertices $v_k := (x(t_k), y(t_k))$. It is some α_N with N suitably large) that actually gets drawn on the computer screen when we want to display α. This reduces the actual drawing problem to that of drawing a straight line segment, and the latter is of course built into every computer system at a very low level.

In Matlab the code for plotting the curve α, or rather the polygon α_{30} would be:

```
N = 30
h = (b-a)/N;
t = a:h:b ;
plot(x(t),y(t)), axis equal;
```

To plot a curve $\alpha(t) = (x(t), y(t), z(t))$ in $\mathbf{R}^3$ is really no more difficult. In Matlab the only change is that the last line gets replaced by:

```
plot3(x(t),y(t),z(t)), axis equal;
```

only now one has to be more careful about interpreting just what it is that one sees on the screen in this case. The answer is that one again is seeing a certain polygon in the plane, but now it is the projection of the polygon in $\mathbf{R}^3$ with vertices at $v_k := (x(t_k), y(t_k), z(t_k))$. (The projection can be chosen to be either an orthographic projection in some direction or else a perspective projection from some point.)

1) How do you create animations?

Viisualization can be a powerful tool for gaining insight into the nature of complex mathematical objects, and frequently those insights can be further enhanced by careful use of animation. Remember that time is essentially another dimension, so animations allow us to pack a lot more information onto a computer screen in a format that the human brain can easily assimilate. The number of ways that animation can be used are far to numerous to catalog here, but in addition to obvious ones, such as rotating a three dimensional object, one should also mention "morphing". Mathematical objects frequently depend on several parameters (e.g., think of the family of ellipses: $x = a\cos(\theta)$, $y = b\sin(\theta)$). Morphing refers to moving along a curve in the space of parameters and creating frames of an animation as you go.

All animation techniques use the same basic technique—namely showing a succession of "frames" on the screen in rapid succession. If the number of frames per second is fast enough, and the change between frames is small enough, then the phenomenon of "persistence of vision" creates the illusion that one is seeing a continuous process evolve. Computer games have become very popular in recent years, and they depend so heavily on high quality animation that the video hardware in personal computers has improved very rapidly. Still, there are many different methods (and tricks) involved in creating good animations, and rather than try to cover them here we will have some special lectures on various animation techniques, with particular emphasis on how to implement these techniques in Matlab.

Matlab Project # 6.

Your assignment for the sixth project is to implement the Fundamental Theorem of Plane Curves using Matlab. That is, given a curvature function $k : [0, L] \to \mathbf{R}$, construct and plot a plane curve $x : [0, L] \to \mathbf{R}^2$ that has k as its curvature function. To make the solution unique, take the initial point of x to be the origin and its initial tangent direction to be the direction of the positive x-axis. You should also put in an option to plot the evolute of the curve as well as the curve itself. Finally see if you can build an animation that plots the osculating circle at a point that moves along the curve x. For uniformity, name your M-File PlaneCurveFT, and let it start out:

`function x = PlaneCurveFT(k,L,option)`

If option is not given (i.e., nargin = 2) or if option = 0, then just plot the curve x. If option = 1, then plot x and, after a pause, plot its evolute in red. Finally, if option = 2, then plot x and its evolute, and then animate the osculating circle (in blue) along the curve, also drawing the radius from the center of curvature.

[To find the curve x, you first integrate k to get $\overrightarrow{t} = x'$, and then integrate $\overrightarrow{t}$. The curvature, k, will be given as a Matlab function, so you can use the version of Simpson's Rule previously discussed for the first integration. But $\overrightarrow{t}$ will not be in the form of a Matlab function that you can substitute into that version of Simpson's Rule, so you will need to develop a slightly modified version of Simpson's. where the input is a matrix that gives the values of the integrand at the nodes rather than the integrand as a function.]