Graphing Calculators

The inexpensive, handheld graphing calculator has replaced (for the most part) the slide-rule and handbook[1] as primary assistant in graphing functions. But there are times when one needs the ability to probe ever deeper into specific aspects of a particular function. This needs to be done on a programmable graphics gadget. Consider the following "poem" which plots the sine fucntion in an unusual, and in an unusually interesting way.

```
10 REM SINEWHEEL
11 CLS : REM CLEAR SCREEN
14 DATA .01745 : REM 1 DEGREE
15 DATA 6, 28, 28, 32
16 READ DG, DX, R, XO, YO
50 FOR X=0 TO 360 STEP DX
60 XR = XO + R*COS(X*DG)
61 YR = YO - R*SIN(X*DG)
62 XX = XO + R + R*X*DG
69 REM DRAW LINE WITH PEN DOWN
70 LINE (XO,YO)-(XR,YR),1
71 LINE (XR,YR)-(XX,YR),1
72 LINE (XX,YR)-(XX,YO),1
80 NEXT
```

Since the original BASIC was not graphics oriented, its graphical syntax varies among different implementations. If you are trying this out on an Apple II you should replace lines 70-72 and 11 with this.

```
75 HPLOT XO,YO TO XR,YR TO XX,YR TO XX,YO
11 HGR : HCOLOR = 15 : REM WHITE
```

Even without a computer, you can figure out what the picture will look like simply by drawing (with a pencil) what the program is telling the computer to do. Lines 50 through 80 form a package called a loop. In essence, this loop generates the 60 points `XR,YR` on a circle centered at `XO,YO` and radius `R` in 6 degree steps. Does the circle go clockwise or counterclockwise ?[2] Now things become tricky. Notice how line 62 computes the x-coordinate of a point that unrolls the circle on a line. Surely you have seen such an animation illustrating the meaning of the sine function.

In the picture drawn by this animation[3] reveals a sine-graph only as an illusion. Most of the time one wants to draw a polygonal line through the points spaced

[1] My constant companion in the fifties at homework and study sessions was the "Burington: Handbook of Mathematical Tables and Formulas". Sometimes, when the professor wanted us to improve our scores we were allowed to take the Burington to exams.

[2] In a right handed coordinate system, it is clockwise. Many early computers saved resources by making the y-axis go down the screen instead of up the screen.

[3] On slow computer the drawing proceeds slowly enough to understand what is going on. On a fast computer one would "slow it down" by putting a pause into the loop. If you don't know the correct syntax for the official pause function in the particular computer language at hand, use a long enough "empty" loop `FOR P=0 TO 10000: NEXT` , for example.

closely enough to give another illusion, that of a smooth curve. To trace the circle, connect successive values of `XR,YR` in the loop. In Applesoft BASIC the `HPLOT TO XR,YR` command connect the current point to the previous point drawn, a syntactic structure suitable for loops. The one point drawing command `HPLOT XR, YR` completes the plotting suite in Applesoft. The same idea is expressed in basiCGLUT by replacing the `glBegin(GL_POINTS)` by `glBegin(GL_LINE_STRIP);` bracketing the stream of point plotting commands.[4]

Exercise 13. Modify the Sinwheel to trace a "continuous" sine curve. Add color to the lines drawn. Generalize the concept to other functions. Make it interactive.

There are two ways to go from here. We could continue the previous exercise until you can build a general function grapher. To this end we include an pocket program on automatic scaling. But a more interesting way of applying the trick in Sinewheel to visualizing Chaos is described in the next section.

A persistent problem with simple graphics systems is one of *scaling.* The screen coordinates, in pixel-units, are ill adapted for computing. One uses world coordinates for computation, and grown-up function graphers *auto-scale.* That is, they automatically scale the function values to fit inside a prescribed rectangle on your screen, called a *viewport.* The Functificator, below, is such an autoscaler in BASIC. The following version served us well in the Apple Lab, see if you can implement it efficiently in a modern language.

```
10 REM FUNCTIFICATOR
11 READ P, Q, A, B
12 DATA 9, 1, 0, 1
20 READ XM,  YM
21 DATA 300,200
30 DIM Y(XM)
39 DX = (B-A)/XM
40 FOR X = A TO B STEP DX
50 Y = X^P*(1-X)^Q
60 IF Y < MIN  THEN MIN = Y
61 IF Y > MAX  THEN MAX = Y
70 I=I+1 : Y(I)=Y
80 NEXT X
90 DY = (MAX-MIN)/YM
100 CLS
110 FOR I=0 TO XM
120 PSET(I,(Y(I)-MIN)/DY)
130 NEXT I
```

[4] As with human languages, OpenGL syntax has several ways of modifying a stem. The `glBegin()`accepts a word as its argument telling it what to begin. But the vertex drawing stem `glVertex` has many variants. The one closest to `HPLOT` is `glVertex2f()` , the "2f" specifies that the arguments of the function will be the 2 float point coordinates.

It graphs the function[5] (line 50)

$$y = x^p(1-x)^q,$$

whose parameters, the two powers p, q and the domain $[0, 1]$ of its independent variable, x, are set on lines 11 and 12. `XM` and `YM` hold the dimensions, in pixels, of the viewport. On line 30 BASIC allocates an array `XM` of values of Y, one for each pixel. There are exactly `XM` steps of size `DX` (line 39) on the interval `[A,B]`. We can write a loop as in line 40, using world coordinates, or, we could write it as a counted loop in pixel coordinates, as we do in line 110.

In theory, we need only one loop to draw the graph of the function. But, since we do not know *a priori* the range of the values, we use two loops. The first, 40–80, finds the minimum and maximum of the values at the same time, and we use these to scale the resulting graph into the available vertical space in the viewport. In BASIC, the first time the name of a variable is encountered it has value 0. Convince yourself that this loop does its job for any function on the interval, just as long as initially `MIN=MAX`.

Once we know the range of the y-values, we can scale them into the available range, line 90. Note that substitution yields

$$\frac{\mathtt{Y(I)} - \mathtt{MIN}}{\mathtt{MAX} - \mathtt{MIN}}\mathtt{YM}$$

for vertical screen coordinate in line 120.

Exercise 14. A linear mapping of a range $a \le x \le b$ into the range $A \le X \le B$ can be easily remembered from this formula

$$X = A\frac{b-x}{b-a} + \frac{x-a}{b-a}B.$$

A clever way of reading this expression is that when $x = a$ then $X = A$, and when x finally gets to b then the formula reduces to B. In between, the two fractions always add up to 1. Thus you can think of this as a weighted average of A and B. Incidentally, there is nothing that requires A, B to be numbers. They can be vectors, matrices, anything mathematical that adds and scales. Now, here's the exercise. Write a functificator that automatically scales a parametrized curve,

$$(x, y) = (f(t), g(t)),\ t_{min} \le t \le t_{max}$$

into an arbitrary *viewport*. To keep things interesting, try *Lissajoux* figures, where the functions are sinusoidal, $f(t) = \alpha\cos(\lambda t - \phi)$, with differeing amplitudes, α, frequencies, λ, and phase angles, ϕ.

[5] These functions are of central importance in the construction of *splines*, which are curves whose shape are controlled by a few points on or near them. They serve as *weights* or *basis functions* for Bezier splines.