Plotting the spirograph equations with gnuplot

Víctor Luaña[*]
Universidad de Oviedo, Departamento de Química Física y Analítica, E-33006-Oviedo, Spain.
(Dated: October 15, 2006)

GNUPLOT[1] internal programming capabilities are used to plot the continuous and segmented versions of the spirograph equations. The segmented version, in particular, stretches the program model and requires the emmulation of internal loops and conditional sentences. As a final exercise we develop an extensible minilanguage, mixing GAWK and GNUPLOT programming, that lets the user combine any number of generalized spirographic patterns in a design.

Article published on Linux Gazette, November 2006 (#132)

I. INTRODUCTION

Imagine the movement of a small circle that rolls, without slipping, on the inside of a rigid circle. Imagine now that the small circle has an arm, rigidly atached, with a plotting pen fixed at some point. That is a recipe for drawing the hypotrochoid,[2] a member of a large family of curves including epitrochoids (the moving circle rolls on the outside of the fixed one), cycloids (the pen is on the edge of the rolling circle), and roulettes (several forms rolling on many different types of curves) in general.

The concept of wheels rolling on wheels can, in fact, be generalized to any number of embedded elements. Complex lathe engines, known as *Guilloché* machines, have been used since the XVII or XVIII century for engraving with beautiful designs watches, jewels, and other fine craftsmanships. Many sources attribute to Abraham-Louis Breguet the first use in 1786 of Gilloché engravings on a watch,[3] but the technique was already at use on jewelry. Ancient machines still at work can be seen at RGM Watch Company webpages.[4] Intrincated Guilloché patterns are usually incorporated on bank notes and official documents to prevent forgery. The name "Spirograph" comes, actually, from the trade denomination of a toy invented in 1962 by Denys Fisher, a british electronic engineer, and licensed to several toy companies over the years.

Our purpose, however, is not to explore the history or even the mathematical aspects of the Spirograph decorations, but rather our interest is centered on the techniques needed to use GNUPLOT as the drawing engine of the cycloid-related curves.

Section II presents a simple derivation for the hypotrochoid equations and discusses a generalization to any number of rolling wheels due to F. Farris.[5] Section III describes the several techniques required to draw the cycloid-related curves with GNUPLOT. From the use of complex arithmetic to the simulation of an implicit do loop and the recursive definition of user functions, GNUPLOT offers a large capability for the creation of algorithmic designs. The techniques discussed in section III are embedded within a simple GAWK filter that reads a formal description of a cycloid pattern and uses GNUPLOT to produce the final plot. The design of this filter is the subject of section IV.

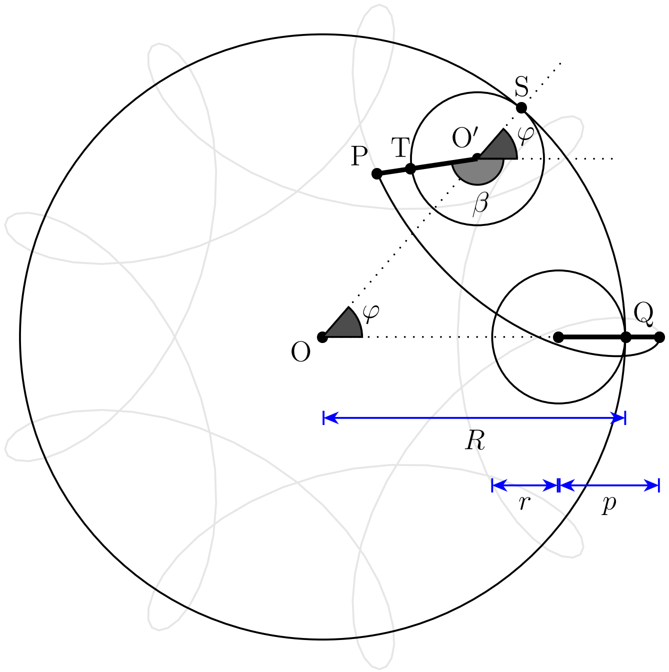


FIG. 1: Geometry for the hypotrochoid equations. The grayed figure corresponds to $R = 9$, $r = 2$, and $p = 3$.

II. THE HYPOTROCHOID AND SOME RELATED CURVES

Figure 1 shows the formation of a hypotrochoid and will help us in determining the parametric equations for the curve. Three lenghts determine the shape of the curve: R, the radius of the fixed circle; r, the radius of the moving circle; and p, the distance from the pen to the moving circle center. The center of the fixed circle, point O, will serve as the origin of coordinates. Points O′ and P designate the current position of the rolling circle center and of the pen, respectively.

The current position for O′ is easily described in circular coordinates: fixed length $\overline{\mathrm{OO'}} = (R - r)$ and variable angle φ. This is easily translated into cartesian coordinates:

$$x_{\mathrm{OO'}} = (R - r)\cos\varphi, \qquad y_{\mathrm{OO'}} = (R - r)\sin\varphi. \quad (1)$$

Similarly, the position of the pen relative to O′ is also simple to describe in circular coordinates: fixed length $\overline{\mathrm{O'P}} = p$ and variable angle $2\pi - \beta \equiv -\beta$. In cartesian coordinates:

$$x_{\mathrm{O'P}} = p\cos\beta, \qquad y_{\mathrm{O'P}} = -p\sin\beta. \tag{2}$$

The angles φ and β are not independent, however. The circles roll without slipping. Hence, the arc $\overset{\frown}{\mathrm{QS}} = R\varphi$ on the fixed circle must be identical to the arc $\overset{\frown}{\mathrm{TS}} = r(\varphi+\beta)$ on the rolling circle. The relationship $\beta = (R-r)\varphi/r$ follows immediately. This equation is easy to interpret in terms of a gearing mechanism. The fixed and rolling wheels must have teeth of equal size to be able to engage together. Therefore, the number of teeth must be proportional to the wheel perimeter and, equivalently, to the wheel radius.

Using all together, the current position of the pen relative to the fixed center O is given by $\vec{r} = \vec{r}_{\mathrm{OP}} = \vec{r}_{\mathrm{OO'}} + \vec{r}_{\mathrm{O'P}}$, or, equivalently:

$$x(\varphi) = (R-r)\cos\varphi + p\cos\left(\frac{R-r}{r}\varphi\right), \tag{3}$$

$$y(\varphi) = (R-r)\sin\varphi - p\sin\left(\frac{R-r}{r}\varphi\right). \tag{4}$$

The equations admit r and p being either positive or negative. A negative r would represent a moving wheel rolling on the outside, rather than the inside, of the fixed cirumference, i.e. it will be a epitrochoid curve. Choosing $p = r$ with r positive or negative, will produce hypo or epicycloid curves, respectively.

It is easy to observe that multiplying the three parameters R, r, and p by a common factor produces a global scaling of the curve dimensions but do not changes its shape. On the other hand, the figure traced by this parametric equations closes only if R/r is a rational number. Let us assume that n and m are the smallest integers such that $|R/r| = n/m$, and let $g = \gcd(n,m)$ be the greatest common divisor of n and m. The curve will then close after a total rotation of m/g times 2π ($\varphi \in [0, 2m\pi]$) and it will show n/g lobes or spikes.

The equations can be generalized for three or more wheels rolling one inside each other, and Frank Farris did so in a celebrated article on *Mathematics Magazine*.[5] At this level it is better to give up a direct simulation of the physical engine gears and examine directly the equations. On the other hand, a very compact and powerful notation is obtained by using complex variables, with the convention that the real and imaginary parts represent the x and y cartesian coordinates, i.e. $z = x + iy$ where i is the imaginary number. The general Farris equations are:

$$z(t) = \sum_{k=1}^{n} a_k e^{i2\pi(n_k t + \theta_k)}, \quad t \in [0,1], \tag{5}$$

where n is the number of engaged wheels: wheel k have its center fixed on a point of the circumference of wheel $k-1$. On each wheel, a_k is related to the radius, n_k to the rotation speed, and θ_k is an initial phase angle. Farris demonstrated that the $z(t)$ curve has g-fold rotational symmetry if all the pairwise differences $|n_k - n_j|$ have g as their greatest common divisor.

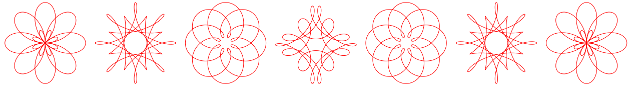
III. RENDERING THE CURVES IN GNUPLOT

The two wheels parametric equations, eq. 3 and 4, can be readily translated into the following GNUPLOT drawing code:

```
 set terminal png size 600,600 \
    x000000 xffffff x404040 xff0000
 set output "fig-spiro02.png"
 set size ratio -1
 set nokey
 set noxtics
 set noytics
 set noborder
 set parametric
#
 x(t)=(R-r)*cos(t) + p*cos((R-r)*t/r)
 y(t)=(R-r)*sin(t) - p*sin((R-r)*t/r)
#
 R=100.0; r=2.0; p=80.0
 set samples 2001
#
 plot [t=0:2*pi] x(t),y(t)
```

The code saves the image as a PNG file, useful for insertion on a web page, but any GNUPLOT terminal could be used. An EPS/PDF vector file with white background is better for a printed version of the document, whereas an unscaled PNG raster file with black background may look better and render faster in a web browser. The use of the PNG terminal is a little tricky, as two versions that differ in the recognized syntax appear to exist. If GNUPLOT chokes on the `png size 600,600` part try using `png picsize 600 600` instead. Notice, on the other hand, that we have removed the default axes, labels and tics. Identical scaling of both axes has also been enforced to avoid distortion of the image. The result can be seen in Fig. 2.

A little exploration will reveal that: (a) $p = 0$ produces a circle; (b) an ellipse results if $R = 2r$ and $p \neq r$, its axes being $(r + p)$ and $|r - p|$; (c) the hypocycloids are obtained by choosing $p = r$; (c) $R = 2r = 2p$ gives rise to a line of length $2a$; (d) negative values for p and/or r results on some extraordinary specimens.



The beauty and diversity of the trochoid curves calls for a journey of exploration and discovery. This is much facilitated if the gnuplot code is embedded in a text or graphical user interface (TUI *vs.* GUI). A simple `csh` script can serve as a rudimentary but effective wrapper:

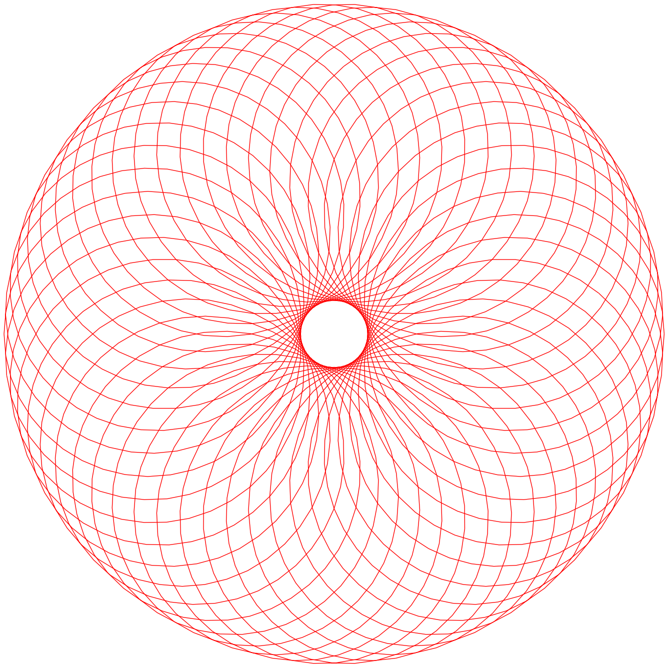


FIG. 2: Hypotrochoid curve for: $R = 100$, $r = 2$ and $p = 80$.

```
#! /bin/csh
set code = $0:t
if ($#argv < 3) goto help
set n1 = $1; set n2 = $2; set n3 = $3
set a1 = 1.0; set a2 = 1.0; set a3 = 1.0
set s1 = 0.0; set s2 = 0.0; set s3 = 0.0
if ($#argv >= 4) set a1 = $4
if ($#argv >= 5) set a2 = $5
if ($#argv >= 6) set a3 = $6
if ($#argv >= 7) set s1 = $7
if ($#argv >= 8) set s2 = $8
if ($#argv >= 9) set s3 = $9

cat << EOF | gnuplot
 set size ratio -1
 set nokey
 set noxtics
 set noytics
 set noborder
 set parametric
#
 n1p = {0,1}*2*pi*${n1}
 n2p = {0,1}*2*pi*${n2}
 n3p = {0,1}*2*pi*${n3}
 s1p = {0,1}*2*pi*${s1}
 s2p = {0,1}*2*pi*${s2}
 s3p = {0,1}*2*pi*${s3}
 z(t) = ${a1}*exp(n1p*t+s1p) \
      + ${a2}*exp(n2p*t+s2p) \
      + ${a3}*exp(n3p*t+s3p)
#
 set terminal png size 600,600 x000000 \
     xffffff x404040 xff0000 xffa500 x66cdaa \
     xcdb5cd xadd8e6 x0000ff xdda0dd x9500d3
 set output "fig-spiro03.png"
#
 set samples 2001
 plot [t=0:1] real(z(t)),imag(z(t))
EOF

xv fig-spiro03.png
exit(0)

help:
cat << EOF
USE:     $code n1 n2 n3 [a1 a2 a3 [s1 s2 s3]]
PURPOSE: Plot Farris wheels on wheels on wheels
         curve for (n1,n2,n3,a1,a2,a3,s1,s2,s3).
         Default value for a1, a2, a3: 1.0.
         Default value for s1, s2, s3: 0.0.
EXAMPLE: $code 1 7 -17 1 0.5 1.0/3 0 0 0.24
EOF
```

In this example we have used Farris equations for three wheels. Complex numbers (notice the `{0,1}` constant, equivalent to i in GNUPLOT sintax) are used to evaluate the $z(t)$ function, but its real and imaginary parts must be explicitely extracted and passed to the `plot` order. Therefore, $z(t)$, is actually called twice for each point. Perhaps future GNUPLOT versions will recognize a single complex expression as a complete argument of the parametric `plot`. In any case, the complex arithmetic provides a very compact notation.

The script, on the other hand, can be called with anything from 3 to 9 parameters. The three obligatory parameters are n_1, n_2 and n_3, that adjust the relative speed of the three wheels. The next group of three are a_1, a_2 and a_3, related to the relative size of the wheels, and a default value of 1,0 is assumed for any parameter not given in the input. The last group corresponds to the initial phase angles, θ_1, θ_2 and θ_3, with a default value of 0,0. The script parameters are used only within GNUPLOT assignments. This means that the user can enter expressions rather than single integer or real values. Some care must be taken, however, when entering fractions: use 1,0/3 and not 1/3, that would be interpreted by GNUPLOT as an integer division and would produce an unexpected 0.

Figure 3 represents some of the most characteristic patterns exhibited by three rolling wheels of identical size. These patterns occur when the wheel speeds, $[n_1, n_2, n_3]$, their differences, $\Delta n_{ij} = n_j - n_i$, and the greatest common divisor of the differences, $g = \gcd(|\Delta n_{ij}|)$, satisfy appropriate conditions. Large values for g produce highly symmetric and generally nice motives. Some of the most pleasant designs, however, show only a moderate symmetry and a more subtle interplay between regularity and uniqueness.

Some trends, found by the observation of a large number of cases, can help in designing a particular motif. Assuming that the three wheels are of equal size, differences like $(-g, 2g, 3g)$ can produce g-points stars, whereas g-petals daisies tend to happen for $(\pm 2g, \pm g, \pm g)$ differences. Palm leaves and nephroids occur when two of the Δn_{ij} differences coincide, in absolute value, with two of the wheel speeds. Crosses and Maasai shields are *rara avis* that require a large number of circumstances: the

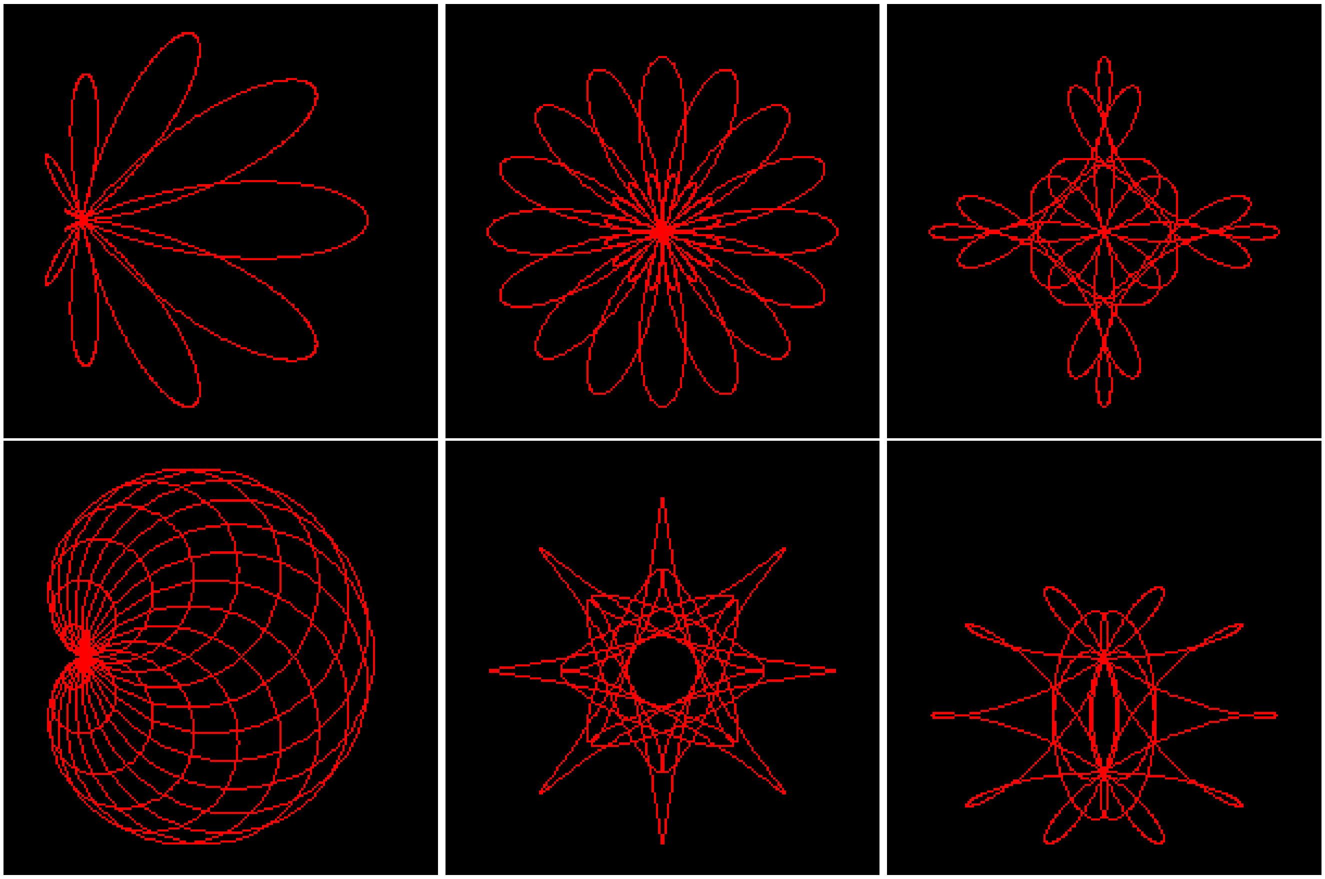
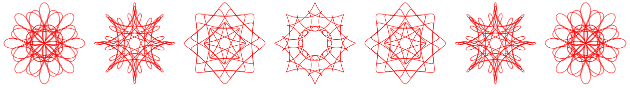


FIG. 3: Typical patterns shown by three rolling wheels of equal size. From left to right and top to bottom: (a) $[7,-5,2]$ palm leaf, (b) $[19,-13,3]$ daisy flower; (c) $[13,-7,-3]$ cross; (d) $[19,17,-2]$ nephroid; (e) $[13,-11,-3]$ 8-point star; (f) $[11,-7,-3]$ Maasai shield.

sum of all wheel speeds must be odd (positive or negative), g must be a power of 2, the sum of two of the differences must be equal to the third.

Changing the wheel sizes will also produce significant variations on the drawings. Adding small phase angles to one or more wheels can be used to introduce some irregularity into an otherwise too symmetric and uninteresting motif.



The two previous example codes have sampled the spirographic equations with a large number of points, large enough to show the true nature of the curves: continuous and derivable, as they are sums of exponential functions. The web is plenty of simplistic java applets that render incorrectly the equations by using a small number of points per roll. The method, albeit a wrong representation of the true curves, can produce quite pleasent images. On a declarative language this type of plot would be produced using a simple loop:

```
nturns = abs(rsmall) / gcd(Rbig,abs(rsmall))
M = nturns * resolution
inumber = {0,1}
for (k=0; k<=M; k++) {
   ang1 = inumber * k * 2*pi/M
   ang2 = ang1 * (rsmall-Rbig)/rsmall
   z[k] = (Rbig-rsmall)*exp(ang1) + p*exp(ang2)
   if (k>0) { PLOT LINE from z[k-1] to z[k] }
   }
```

where `resolution` holds the number of sample points used for each roll around the main (fixed) wheel and `nturns` is the number of times this wheel must be rolled around. The above pseudocode assumes complex arithmetic and the availability of a `gcd()` function.

Loops and conditional sentences are not part of the GNUPLOT language, but there are ways around this limitation. First, an implicit loop is automatically done on each plot order. We just have to be careful fixing an appropriate `samples` value and a correct range for the independent variable (the parametric variable in our case). The ternary operator (`a?b:c`, evaluate and return `b` if `a` is true, and `c` otherwise) can be used as a restricted conditional form. GNUPLOT user-defined functions can be recursive, on the other hand, and this can also be used as a restricted form of loop.

The next code uses all of the above ideas. Notice, in particular, the recursive definition of the `gcd()` function,

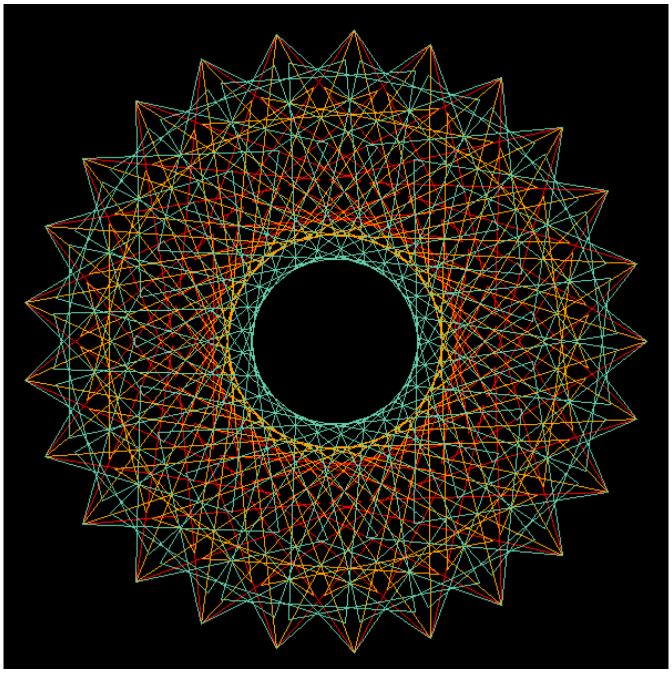


FIG. 4: Curve stitching patterns from the hypotrochoid curve with: $R = 100$, $r = 2$ and $p = 70$. The three patterns correspond to a resolution of 75, 125 and 175 sample points, respectively.

that implements Euclid's algorithm for the greatest common divisor. The calculation of the number of turns and sample points is simplified by assuming that R and r are integers.

```
set size ratio -1
set nokey
set noxtics
set noytics
set noborder
set parametric
#
x(t) = (R-r)*cos(t) + p*cos((R-r)*t/r)
y(t) = (R-r)*sin(t) - p*sin((R-r)*t/r)
#
# Greatest common divisor:
gcd(x,y) = (x%y==0 ? y : gcd(y,x%y))
#
R = 100; r = -49; p = -66; res = 10
#
rr = abs(r)
nturns = rr / gcd(R,rr)
samp = 1 + res * nturns
set samples samp
#
plot [t=0:nturns*2*pi] x(t),y(t)
```

The last code works well for drawing a single curve with a given resolution, but the most interesting patterns are obtained by mixing several renderings of one or more curves with well chosen resolutions. To do this within a single GNUPLOT run we have to take explicit control of the angles used for each equation. For instance:

```
set terminal png picsize 600 600 x000000 \
    xffffff x404040 xff0000 xffa500 x66cdaa \
    xcdb5cd xadd8e6 x0000ff xdda0dd x9500d3
set output "fig-spiro05.png"
set size ratio -1
set nokey
set noxtics
set noytics
set noborder
set parametric
#
# General parametric equations:
x(t,R,r,p) = (R-r)*cos(t) + p*cos((R-r)*t/r)
y(t,R,r,p) = (R-r)*sin(t) - p*sin((R-r)*t/r)
#
# Values for the dummy parameter:
t(i,n) = i*2*pi/n
#
# Greatest common divisor:
gcd(x,y) = (x%y==0 ? y : gcd(y,x%y))
#
# The different curves:
R1 = 100; r1 = 2; p1 = 70; res1 =  75
R2 = 100; r2 = 2; p2 = 70; res2 = 125
R3 = 100; r3 = 2; p3 = 70; res3 = 175
#
nseg1 = res1 * abs(r1) / gcd(R1,abs(r1))
nseg2 = res2 * abs(r2) / gcd(R2,abs(r2))
nseg3 = res3 * abs(r3) / gcd(R3,abs(r3))
n12 = (nseg1 * nseg2) / gcd(nseg1,nseg2)
nsamp = (n12 * nseg3) / gcd(n12,nseg3)
nsamp1 = nsamp + 1
set samples nsamp1
print "nseg{1,2,3} ---> ", nseg1, nseg2, nseg3
print "nsamp ---------> ", nsamp
#
plot [i=0:nsamp] \
   x(t(i,res1),R1,r1,p1),y(t(i,res1),R1,r1,p1) \
 , x(t(i,res2),R2,r2,p2),y(t(i,res2),R2,r2,p2) \
 , x(t(i,res3),R3,r3,p3),y(t(i,res3),R3,r3,p3)
```

The result of this code is represented in the figure 4. The intricate embroidery of the three curve representations, only recognizable by their different color, shows an appealing and delicate beauty that deserves further exploration. However, using the same `plot` order for the three is far from being effective and poses many problems for its generalization to an arbitrary number and class of representations. In particular, the number of sample points has to be a minimum common multiple of the best number of sample points for each independent figure.

In the case of fig. 4, the three curves would need 75, 125 and 175 sample points, respectively, but plotting the three simultaneously requires 2625 samples, instead. So, the first component is repeated 35 times, 21 times the second, and 15 times the third. This repetition will add substantially to the plotting time but, if the final result is written to a raster format like PNG, there will be no increase on the size of the final file. If we use a vector format like EPS, however, the file size will also increase substantially.

We can avoid the unnecessary repetition by turning to a two pass method. In the first pass, each curve is created independently in GNUPLOT, and its points are saved in a file using a `set terminal table` output mode. The second pass combines the points from all the previous files into a single design, that is saved on whatever raster or vector format seems appropriate. Instead of providing an example of this technique we will use the idea for our final and most ambitious project.

IV. A MINILANGUAGE FOR COMBINING AN ARBITRARY NUMBER OF SPIROGRAPH-LIKE PATTERNS

All the techniques developed in the previous section can be made more accesible if we design a simple way of describing a plot and we create the tool for translating the description to the appropriate GNUPLOT orders. The perfect translation tool would hide the details of the GNUPLOT sintax from the user while maintaining an appropriate flexibility.

We have written an experimental translator in awk for rapid prototyping and easy change. We are going to describe the language currently recognized. The following notation will be used. Fixed names are written in **boldface**. Variable data appears in *italica*, enclosed within square brackets, [], if the data is optional. The type of the data is indicated by a suffix of the variable name: *var.s* (a string sequence); *var.i* (an integer); *var.r* (a real value); *var.c* (a complex value in GNUPLOT notation, i.e., {`real_part,imaginary_part`}); *var.re* and *var.ce* (a real or complex expression, like `pi*{0,1}/12`). Blank characters are forbidden within the integer, real and complex data and expressions.

The instructions currently implemented in our translation script are:

PROJECT *projectname.s*

The project name serves as root for the names of all the files, temporary or final, that the drawing process requires.
Default *projectname*: `tmp`.

TERMINAL *parameters.s*

Select a GNUPLOT terminal type. Typical elections would be PNG or EPS, but any type accepted by GNUPLOT will serve.
Default: png size 600,600 x000000 xffffff x404040 ...

CURVE *samples.i*

Start a new curve. A plot is formed by one or more curves. Each curve is made by one or more terms. Each curve is computed independently and written to a temporary file. All curves are plotted together at the end to form the final image. The default number of sample points (2001) can be changed independently for each curve. Some special term types (like **TROCHOID**) may take control of this and ignore the *samples* value. The general form of a curve is:

$$z(t) = \sum_{k=1}^{N} z_k(t)$$

where t is the independent parameter and $z_k(t) = x_k(t) + i y_k(t)$ is a complex term contribution to the curve.

ADDTERM LINE *z0.ce z1.ce*

Add a linear term contribution to the current curve:

$$z_k(t) = \texttt{z0} + (\texttt{z1} - \texttt{z0})t$$

The line passes through `z0` and `z1`, two arbitrary points in the complex plane. Remember than complex values are entered using the GNUPLOT notation: `{x,y}` means $x + iy$. Blank character must be avoid within each number.

ADDTERM SPIRAL1 *a.re* [*n.i*]

Add a generalized Archimedes spiral:

$$z_k(t) = a(2\pi t)^n e^{i2\pi t}$$

Default: $n = 1$.

ADDTERM SPIRAL2 *a.re b.re*

Add a equiangular spiral:

$$z_k(t) = a e^{t/\tan b} e^{i2\pi t}.$$

Maximize the number of spiral rolls by choosing b close to $\pm\pi$.

ADDTERM TROCHOID *R.re r.re p.re* [*samples.i*]

Add a trochoid term:

$$z_k(t) = (R - r)e^{i2\pi t} + p e^{-i2\pi(R-r)t/r}.$$

If *samples* is given, the number of sampling points and the number of curve rollings is internally computed.

ADDTERM WHEEL *a.re n.re s.re*

Add a Farris rolling wheel term:

$$z_k(t) = a e^{i2\pi(nt+s)}$$

where a is the wheel radius, n the rolling frequency, i.e. the number of rolls when t goes from 0 to 1, and s determines the initial phase angle. An arbitrary number of Farris wheels can be added in a curve.

STYLE *style.s*

This order takes control of the GNUPLOT style used on the current curve and its copies (see the **LOOP** order below). The *style.s* must follow the GNUPLOT sintax. For example: `style lt 1` (line type 1 for all copies of the curve), `style with points pt -1` (use points instead of lines). The default style is to use lines for the curves, increasing the line number style for each successive curve, including copies.

LOOP *var.s var_ini.r var_end.r* [*var_inc.r*]
Repeat the current curve according to the next implicit loop:

```
var = var_ini
while (var does not reach var_end) do
   # compute and draw a new installment
   # of the current curve
   var = var_ini + (var_inc)
end
```

The *var.s* string can be used as a variable in the definition of the curve (see the examples below). The *var_inc.r* value can be negative. There is currently a limit of 1000 times for the number of copies produced by this loop.

TRANGE *t0.r t1.r*
Change the default range for the t parametric variable. Notice that the presence of a trochoid term may take control for t and ignore this input.
Default: $t \in [0, 1]$.

GFACTORS *freal.re fimag.re*
Multiplicative factors for the real and imaginary parts of the drawn functions. In other words, the image finally drawn is `plot freal*real(z(t)), fimag*imag(z(t))`.
Default: `freal = fimag = 1.0`.

LABEL *x.r y.r text.s*
Add a GNUPLOT label at the indicated position. The plot center has coordinates (0,0). The label is centered on the `(x,y)` point. The plot can contain any number of labels, but the program makes no attempt to avoid collisions between them.

Some examples of the above rules in action may help to understand its possibilities. The first example, represented in Fig. 5(a), shows the use of a simple trochoid curve with a small sampling:

```
project example01
curve 10
   addterm trochoid 100 -49 76
```

The second example shows the use of a loop order to create copies of a curve, applying a little phase rotation and size change to each new copy. See the resulting image in Fig. 5(b):

```
project example02
terminal png size 600,600 x000000
curve
   style lt 1
   loop kkk 0 20 1
   addterm wheel 4*0.98**kkk  -3   kkk/200.
   addterm wheel 5*1.02**kkk   2   kkk/200.
```

The third image, Fig. 5(c), corresponds to a multiple copy version of three Farris wheels:

```
project example03
terminal png size 600,600 x000000
curve
   style lt 1
   loop kkk 0 20 1
   addterm wheel 4  -5   -kkk/60.
   addterm wheel 3   2    kkk/60.
   addterm wheel 2   9    0.
```

Our last example shows a decoration around a text. The decoration was designed by starting with a large Farris wheel and adding two other much smaller wheels, always maintaining an 8-fold symmetry. The use of **gfactors** provides an easy way to stretch an otherwise round motif.

```
project example04
terminal postscript eps enhanced color "Helvetica" 48
gfactors 1.4 0.7
curve
   style lt 1
   loop kkk 0 10 1
   addterm wheel 200*0.96**kkk   1   0.
   addterm wheel  10*0.96**kkk   9   0.
   addterm wheel   9*0.96**kkk  23   0.
label 0 0 G_nuplot rules!
```

V. FINAL REMARKS

We should not end this report without mentioning, at least, some of the excellent java applets that can be found on Internet.[6–10] A well conceived GUI can be of great help on the interactive exploration of a designed subset of the Spirograph vast parametric space. It is not the only approach, however. A custom minilanguage can give access to an arbitrarily large parametric space and hide the dirty details of code generation. GNUPLOT engine has been used for years to produce professional quality plots. Some of its drawbacks, like the lack of true loop mechanisms, can be saved with some ingenuity or embedding the engine within a more general programming tool.

Acknowledgements

The author thanks the Spanish Ministerio de Educación y Ciencia for financial support under project BQU2003-06553.

---

[*] E-mail: `victor@carbono.quimica.uniovi.es`; Visit: `http://web.uniovi.es/qcg/`

[1] T. Williams, L. Hecking, and H.-B. Broeker, *Gnuplot,*

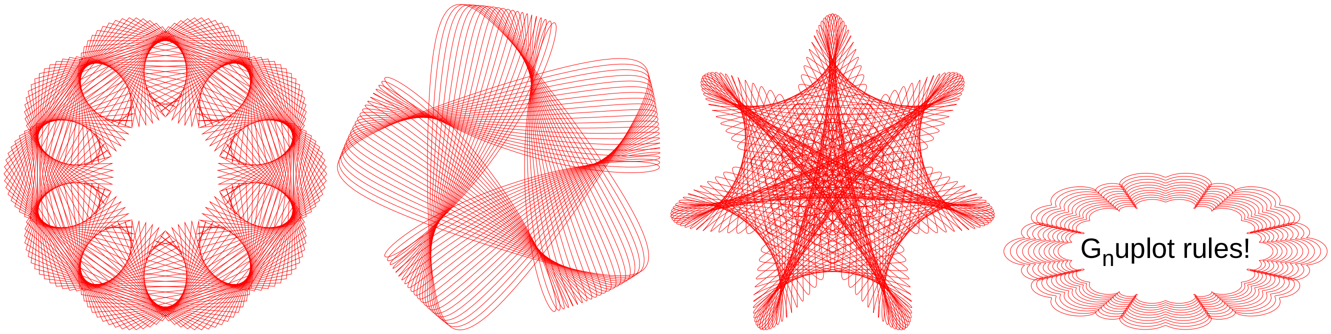


FIG. 5: Example images created with the help of the SpiroLang minilanguage. See the text for a description of the patterns and the actual orders used to create them.

version 4 (2004), based on the original version released by Thomas Williams and Colin Kelley in 1986., URL `http://www.gnuplot.info/`.

[2] E. W. Weisstein, *Hypotrochoid*, from MathWorld–A Wolfram Web Resource (2006), last accessed 2006-09-06., URL `http://mathworld.wolfram.com/Hypotrochoid.html`.

[3] C. Pérez, *L'ancien regime. Part two: The heritage of the classical wristwatch* (2000), last accessed 2006-09-06., URL `http://people.timezone.com/msandler/Articles/CarlosClassical/Classical.html`.

[4] RGM watches company, *Engine-turning "guilloche"* (2006), last accessed 2006-09-06., URL `http://www.rgmwatches.com/engine.html`.

[5] F. Farris, Mathematics Magazine **69**, 185 (1996).

[6] A. Páramo Fonseca, *La gran belleza de las trocoides* (2004), last accessed 2006-09-30., URL `http://temasmatematicos.uniandes.edu.co/Trocoides/paginas/introduccion.htm`.

[7] D. Little, *Spirograph* (2001), last accessed 2006-09-30., URL `http://www.math.psu.edu/dlittle/java/parametricequations/spirograph/index.html`.

[8] D. Little, *Spirograph v1.0* (1997), last accessed 2006-09-30., URL `http://www.math.psu.edu/dlittle/java/parametricequations/spirograph/SpiroGraph1.0/index.html`.

[9] A. Garg, *Spirograph*, last accessed 2006-09-30., URL `http://www.wordsmith.org/anu/java/spirograph.html`.

[10] N. Ziring, *Spiro applet version 1.0* (2000), last accessed 2006-09-30., URL `http://cgibin.erols.com/ziring/cgi-bin/spiro.pl/spiro.html`.