Representaciones Gráficas con *Mathematica* I

Uno de los aspectos más llamativos de *Mathematica* es su capacidad para crear fácilmente gráficos de extraordinaria complejidad. Manejando unos pocos comandos podrás hacer sorprendentes representaciones gráficas y animaciones en dos y en tres dimensiones. Ya has visto algunos ejemplos de esto en las prácticas anteriores y ahora iniciamos un estudio más sistemático de las funciones gráficas de *Mathematica*.

Dibujando gráficas en el plano

`Plot[función,{x,xmin,xmax},opciones]` es el comando que se utiliza para representar la gráfica de una función de una variable. Poniendo una **lista** de funciones como primer argumento, puedes representar juntas las gráficas de todas ellas; en este caso las `opciones` también hay que darlas como una **lista**. También puedes usar `Plot` sin especificar ninguna opción.

```
In[1]:=
 Plot[x^2,{x,0,2}]


In[2]:=
 Plot[{x^2,4Cos[x]},{x,0,2Pi}]
```

Pero es justamente su gran cantidad de opciones lo que proporciona versatilidad y potencia al comando `Plot`. Para verlas escribe `Options[Plot]`. ¿Las has visto ya? Pues no te asustes porque, aunque son muchas, en la práctica es suficiente con conocer unas pocas. Fíjate en que las opciones son **reglas** de la forma "`opción->valor`", y todas ellas tienen prefijados unos valores por defecto. Si no especificas una determinada opción, *Mathematica* supone que quieres utilizar el valor por defecto de la misma. Por ejemplo, *Mathematica* presupone que no quieres enmarcar la gráfica -`Frame->False`- ni poner etiquetas a los ejes -`AxesLabel->None`.

Los posibles valores de una opción debes buscarlos en la ayuda del programa. Cuando el valor de una opción es `Automatic` eso significa que se elige por un algoritmo interno. Por ejemplo, la opción `PlotRange->Automatic` indica que al hacer una representación gráfica, *Mathematica* representará las partes que considera "más interesantes" de la misma. Si la función crece muy rápidamente *Mathematica* puede cortar la gráfica y no representar las partes donde la función toma valores muy grandes. Si te fijas, eso es lo que ha hecho en la segunda gráfica con la función `x^2`. Con la importante opción `PlotRange->{ymin,ymax}` puedes controlar exactamente el rango de valores que se representarán en la gráfica, `PlotRange->All` es la forma de decirle a *Mathematica* que muestre el intervalo completo del eje de ordenadas donde toma valores la función. Experimenta un poco con la segunda de las gráficas anteriores.

Ahora vas a ver cómo ajustando los valores de `xmin` y `xmax` y con la opción `PlotRange` puedes mostrar detalles de una gráfica, algo parecido a "hacer zoom" sobre ella.

```
In[3]:=
 Plot[x Sin[1/x],{x,-1,1}]


In[4]:=
 Plot[x Sin[1/x],{x,-.1,.1},PlotRange->{-.07,.07}]
```

Otra opción importante es `AspectRatio`. Aquí tienes la gráfica de una circunferencia:

```
In[5]:=
 Plot[{-Sqrt[1-x^2],Sqrt[1-x^2]},{x,-1,1}]
```

¿Seguro que es una circunferencia? Pues cualquiera diría que es una elipse. Esto se debe a que la razón "altura/anchura" (`AspectRatio`) que *Mathematica* usa por defecto es la inversa de la razón áurea $\frac{2}{1+\sqrt{5}}$ lo que es agradable estéticamente pero no representa las proporciones reales de las figuras que dibuja. Para que lo haga basta escribir `AspectRatio->Automatic`. Te propongo que uses también las opciones `AxesLabel` y `PlotLabel`, fíjate en que son *cadenas* de texto de la forma `"texto"` y no debes olvidar las comillas. También pondremos marcas en los ejes con la opción `Ticks` y un color de fondo con `Background`.

```
In[6]:=
   Plot[{-Sqrt[1-x^2],Sqrt[1-x^2]},{x,-1,1},
   AspectRatio->Automatic, AxesLabel->{"x",""},
   PlotLabel->çircunferencia",
   Ticks->{{-4/5,-1/3,1/2}, {-3/4,1/4,2/3,{0.95,z"}}},
   Background->RGBColor[1,3/4,1/2]]
```

`PlotStyle` es otra opción importante que, entre otras cosas, permite controlar el color y el grosor de las líneas de un gráfico. El grosor se especifica con `Thickness[r]` donde el grosor de la línea, `r`, viene expresado como una fracción de la anchura total del gráfico. Por ejemplo, para que la línea tenga un grosor del 2 % de la anchura del gráfico, escribimos:

```
In[7]:=
 Plot[Cos[x],{x,0,2Pi},PlotStyle->Thickness[0.02]]
```

`PlotStyle` permite dibujar curvas a trazos con la opción `Dashing[{`r_1`,`r_2`,..}]` que indica que las líneas del gráfico han de ser dibujadas a trozos con segmentos de longitudes sucesivas $r_1, r_2, ..$ repetidas cíclicamente, donde los números $r_1, r_2, ..$ se interpretan como una fracción de la anchura total del gráfico. Procura entender los siguientes ejemplos.

```
In[8]:=
 g1=Plot[Sin[x]+Sin[2x],{x,0,2Pi},
 PlotStyle->Dashing[{0.04}]];


In[9]:=
 g2=Plot[Sin[x],{x,0,2Pi},
 PlotStyle->Dashing[{0.03,0.02,0.001,0.02}]];


In[10]:=
 g3=Plot[Sin[x]+Sin[4x],{x,0,2Pi},
 PlotStyle->Dashing[{.001,.02}]];
```

Esto es útil para representar varias gráficas juntas. Compruébalo con `Show[g1,g2,g3]`. Naturalmente, puedes combinar las opciones de color, grosor de línea y trazos.

Para comprender la utilidad de otras opciones de `Plot` vamos a pedirle a *Mathematica* que haga algunas gráficas complicadas. Usaremos el comando `Nest[f,expr,n]` que proporciona la composición de `f` consigo mismo `n` veces aplicado a `expr`. Usaremos también la llamada "función logística" $x \mapsto rx(1-x)$ con valores del parámetro r que especificaremos en una lista "`rvalor`". Observa la forma de definir lo que en la jerga de *Mathematica* se conoce como una "función pura".

```
In[11]:=
 h[r_]:=Function[x, r x (1 - x)]


In[12]:=
 rvalor={2,3.2360,3.4985,3.5546,3.5667,3.5692};


In[13]:=
 f[n_,x_]:=Nest[h[rvalor[[n]]],x,2^(n-1)]
```

Comprueba qué complicada puede ser la función obtenida iterando unas cuantas veces una función sencilla.

```
In[14]:=
 f[4,x]


In[15]:=
 Expand[ %]
```

Increíble, ¿verdad? La función `f[6,x]` debe ser complicadísima, prueba, sin embargo, a calcular `f[6,0.33]`. ¿Qué te parece?

Veamos las gráficas de algunas de estas funciones. Para dibujar una gráfica, *Mathematica* elige, de forma predeterminada, 25 puntos en los que evalúa la función y, en el caso de que la variación de la función entre puntos próximos sea muy grande, aumenta el número de puntos. Para saber el tiempo que tarda *Mathematica* en ejecutar un comando se usa `Timing[Comando]` cuya salida es una *lista* cuyo primer elemento es el tiempo en segundos que ha necesitado *Mathematica* para ejecutar dicho comando y el segundo elemento es el resultado de realizar el comando. Como la función `f[6,x]` es muy complicada y su gráfico más irregular, *Mathematica* tarda mucho más tiempo en representarla que a `f[1,x]`.

```
In[16]:=
 Timing[Plot[f[1,x],{x,0,1}]]


In[17]:=
 p25=Timing[Plot[f[6,x],{x,0,1}]]
```

Para mejorar la calidad de la gráfica de `f[6,x]` podemos aumentar el número de puntos que *Mathematica* usa para dibujarla. Disponemos para ello de la opción `PlotPoints`, cuyo valor por defecto es de 25 y que vamos a aumentar a 100. Observa que el tiempo de realización del gráfico es aproximadamente cuatro veces mayor.

```
In[18]:=
 p100=Timing[Plot[f[6,x],{x,0,1},PlotPoints->100]]
```

Este ejemplo muestra que la ejecución de gráficos puede ser lenta y más si son tridimensionales. Para ahorrar tiempo es buena idea asignar nombre a los gráficos, lo que permite mostrarlos y modificarlos después con el comando `Show`. Como ejemplo, vamos a hacer "zoom" en los dos gráficos anteriores.

```
In[19]:=
 Show[p25[[2]], PlotRange->{.35, .56}]


In[20]:=
 Show[p100[[2]], PlotRange->{.35, .56}]
```

Como ya los gráficos han sido producidos, el proceso es mucho más rápido. Compara también las calidades de los dos gráficos y observa el notable parecido entre estas gráficas y las anteriores. Ello es porque el gráfico de `f[6,x]` es un elemento de una sucesión de gráficos que converge a un fractal. Para comprobarlo, podemos hacer una tabla con los gráficos de las funciones `f[n,x]`, para `n=1,2,..,6`, y una animación para visualizar la convergencia de las `f[n,x]` a una función que tiene un gráfico fractal. Para que una animación tenga calidad es necesario que todos los gráficos tengan el mismo tamaño, lo que se logra fijando el mismo intervalo en el eje de abscisas, `{x,0,1}`, y haciendo que se representen en el mismo intervalo del eje de ordenadas `PlotRange->{0,1}`. Podemos también dar un color a cada gráfico en función del índice `n`. Esto se hace con la opción `PlotStyle->Hue[(n-1)/6]`. `Hue[arg]` es

una función de período 1, `Hue[0]=Hue[1]`, que asigna a cada valor de su argumento un color desde `Hue[0]` que es rojo, siguiendo los colores del arco iris, anaranjado, amarillo, verde, azul, añil y violeta, y regresando al rojo (`Hue[1]`). Cuando el iterador `n` de la tabla recorre los valores de 1 a 6, `Hue[(n-1)/6]` recorre una secuencia de colores que permite identificar cada uno de los gráficos.

```
In[21]:=
   animacion=Table[Plot[f[n,x],{x,0,1},PlotRange->{0,1},
   PlotStyle->Hue[(n-1)/6]],{n,6}]
```

También puedes mostrtar todos los gráficos juntos usando el comando `Show` basta que escribas `Show[animacion]`.

Dibujando conjuntos de puntos

`ListPlot[{y1,y2,..}]` representa en el plano los puntos de coordenadas $(1,y_1),(2,y_2),\ldots$; y

`ListPlot[{{x1,y1},{x2,y2},..}]`

representa en el plano los puntos $(x_1,y_1),(x_2,y_2),\ldots$. Como una forma usual de obtener listas es mediante `Table`, es frecuente el uso de `ListPlot[Table[]]`.

Las opciones de `ListPlot` son casi las mismas que las de `Plot`. Puedes comprobarlo escribiendo `Options[ListPlot]`.

Un resultado de Stirling afirma que una buena aproximación de $\log n!$ es $n\log n - n$. Podemos comprobarlo dibujando los valores de estas expresiones desde $n = 1$ hasta $n = 50$.

```
In[22]:=
 logfact=ListPlot[Table[Log[n!], {n, 50}],
 PlotStyle->RGBColor[1,0,1]]


In[23]:=
 aprox=ListPlot[ Table[n Log[n]-n,{n,50}], PlotStyle->RGBColor[0,0,1]]


In[24]:=
 Show[logfact,aprox]
```

¿Te parece muy buena la aproximación? Con `Show[logfact,aprox,PlotRange->{120,135}]` la apreciarás mejor.

Curvas parametrizadas planas

Con `ParametricPlot[{fx, fy},{t,tmin,tmax}]`, donde `fx` y `fy` son funciones de `t` en el intervalo `[tmin,tmax]`, puedes representar curvas dadas por sus ecuaciones paramétricas. Para ver las opciones que tiene este comando escribe `Options[ParametricPlot]`. Veamos algunos ejemplos.

```
In[25]:=
 ParametricPlot[{Cos[t],Sin[t]},{t,0,2Pi},
 AspectRatio->Automatic]


In[26]:=
 ParametricPlot[{3 Cos[t],Sin[t]},{t,0,Pi},
 PlotStyle->RGBColor[1,1,0], AspectRatio->Automatic]


In[27]:=
 ParametricPlot[{Cos[3t],Sin[5t+3]},{t,0,2Pi}]


In[28]:=
 ParametricPlot[{Sin[t],Sin[√2 t]},{t,0,10Pi}]
```

La última curva no parece que se vaya a cerrar. Prueba a aumentar el intervalo sustituyendo `10Pi` por `150Pi`. ¿Qué crees que está pasando?

Primitivas y directivas gráficas

Seguro que te has preguntado muchas veces qué quiere decir ese "`Out[n]=-Graphics-`" que *Mathematica* escribe siempre al terminar un gráfico. Pues bien, ahí guarda *Mathematica* toda la información sobre el gráfico en cuestión. Para verla basta que pongas `InputForm[Out[n]]`.

```
In[29]:=
 Plot[x^2,{x,0,1}]

Out[29]=
 -Graphics-


In[30]:=
 InputForm[ %]
```

¿Sorprendido? Fíjate que este resultado muestra claramente que un gráfico es un comando de la forma `Graphics[primitivas,opciones]`. En este caso todas las opciones son las prefijadas pues no hemos especificado ninguna y hay una única primitiva, `Line`, cuya sintaxis es

`Line[{{x1,y1},{x2,y2}...}]`

que dibuja una línea poligonal que une los puntos $(x_1,y_1),(x_2,y_2)\ldots$; es decir, el gráfico generado por `Plot` es una poligonal formada por segmentos que unen puntos próximos de la gráfica de la función. Estos puntos son tan próximos que, a simple vista, no se advierte que la curva está formada por la yuxtaposición de los segmentos que unen cada punto con el siguiente. Puedes actuar directamente sobre el gráfico modificando la salida anterior en una nueva celda y mostrando el resultado con `Show`.

Además de `Line` otras primitivas gráficas son `Point`, `Rectangle`, `Circle`, `Disk`, `Polygon` y `Text`. Como su nombre indica las primitivas gráficas son los elementos básicos (primitivos) que utiliza *Mathematica* para crear gráficos complejos.

El siguiente ejemplo muestra a la primitiva `Point` en acción.

```
In[31]:=
 ListPlot[Table[i^2,{i,3}],
 PlotStyle->{PointSize[.02],RGBColor[0,1,0]}]


In[32]:=
 InputForm[ %]
```

Observa que los puntos de un gráfico realizado por `ListPlot` son producidos por la primitiva `Point[{x, y}]`. Fíjate también en que las especificaciones de color y tamaño de los puntos, que indicamos con `PlotStyle` figuran en una lista que contiene como *sublista* a la lista de los puntos del gráfico *a la cual preceden*. Las especificaciones como `PointSize` o `RGBColor` que preceden a una primitiva o a una lista de primitivas a las que modifican, se conocen como *directivas gráficas* o, simplemente, *directivas*. Otras directivas que ya conoces son `Thickness` y `Dashing`. En general, las especificaciones dadas con `PlotStyle` en comandos como `Plot`, o `ListPlot`, o `ParametricPlot`, que tienen como salida un objeto gráfico son *directivas gráficas que se aplican globalmente*. Lo que hacen precisamente estos comandos es facilitar la construcción de objetos gráficos, es decir de comandos del tipo `Graphics[primitivas,opciones]` y mostrarlos mediante un comando `Show`.

Puedes construir directamente objetos gráficos en *Mathematica* dando una lista (de listas) de primitivas gráficas. Es posible incluir en cada lista directivas que especifican la forma en que se representarán los siguientes elementos gráficos de esa lista o de sus sublistas, pero dichas directivas no tiene ningún efecto fuera de la lista que las contiene.

Escribe `Options[Graphics]`, para ver las opciones del comando `Graphics`. Procura entender los ejemplos que siguen.

```
In[33]:=
     vertices={Hue[0],PointSize[.03],
     Table[Point[{Cos[2 k Pi/5],Sin[2 k Pi/5]}],{k,0,5}]};
     lados={Hue[.5],Thickness[.02],Line[Table[
     {Cos[2 k Pi/5],Sin[2 k Pi/5]},{k,0,5}]]};

In[34]:=
     penta=Show[Graphics[{lados,vertices}],AspectRatio->1];

In[35]:=
     Show[Graphics[{{Hue[0],Rectangle[{0,0},{1,1}]},
     {Hue[.5],Thickness[.02],Circle[{1,1},1/3]}}],
     AspectRatio->1];
```

La primitiva `Polygon[{x1,y1},{x2,y2},..]` representa un polígono relleno cuyos vértices son los puntos $(x_1,y_1),(x_2,y_2),\ldots$ y cuyos lados son los segmentos que unen cada punto con el siguiente y el último con el primero. Por ejemplo, `Polygon[{0,0},{1,0},{0,1}]` es un triángulo. Un caso especial de

`Polygon` es `Rectangle` en la que se dan los vértices inferior izquierdo y superior derecho. La primitiva `Disk` dibuja círculos rellenos. La primitiva `Circle[{x,y},{rx,ry}]`, produce una elipse de semiejes `rx` y `ry`. Para obtener arcos de circunferencia o sectores de círculos o de elipses basta incluir al final del correspondiente comando una lista de la forma `{t1,t2}` donde `0⩽t1<t2⩽2Pi`. Por ejemplo, `Disk[{1,1},{2,3},{Pi/2,3Pi/2}]` es un sector de elipse comprendido en el semiplano de la izquierda.

Cicloides y trocoides

Consideremos una rueda de radio 1 que gira sin deslizar sobre una recta horizontal. Supongamos que la velocidad angular de la rueda sea constante igual a 1, de modo que la velocidad linear del centro de la rueda será también 1. Supongamos además que el origen del sistema de coordenadas es el punto de contacto de la rueda com la recta en el instante $t = 0$. Sujeta en el centro de la rueda por uno de sus extremos hay una varilla rígida de longitud r. Podemos suponer que en el instante $t = 0$ la varilla está vertical con su extremo libre apuntando hacia abajo. Queremos estudiar la ecuación paramétrica de la curva descrita por el extremo libre de la varilla cuando la rueda gira sin deslizar sobre la recta. Si $r = 1$, es decir si la longitud de la varilla es igual al radio de la rueda, con lo que su extremo libre coincide con el punto de contacto de la rueda con la recta horizontal, esta curva se llama cicloide. Si $r \neq 1$, la curva descrita se llama una trocoide. Son, evidentemente, curvas periódicas con período 2π. Basta describir sus ecuaciones para $0 \leqslant t \leqslant 2\pi$.

Como la velocidad angular es constante igual a 1, el ángulo que forma la varilla en el instante $t \in [0, 2\pi]$ con su posición inicial es igual a t. Por tanto, las coordenadas de su extremo libre referidas al centro de la rueda son $(-r\cos t, -r\,\mathrm{sen}\, t)$; y como la posición del centro de la rueda en el instante t es $(t, 1)$, resulta que las coordenadas del extremo de la varilla en el instante t referidas al origen son $(-r\,\mathrm{sen}\, t, -r\cos t) + (t, 1) = (t - r\,\mathrm{sen}\, t, 1 - r\cos t)$. Hemos obtenido así las ecuaciones paramétricas buscadas con lo que ya podemos representar la curva. Pero nuestro propósito es más ambicioso: vamos a realizar una animación de cómo se genera dicha curva al girar la rueda. Procura entender lo que sigue.

```
In[36]:=
     rueda[t_]:={GrayLevel[.8],Disk[{t,1},1]};
     recta={Thickness[.01],Line[{{-2,0},{14,0}}]};
     varilla[r_,t_]:={Line[{{t,1},{t-r Sin[t],1-r Cos[t]}}],
     {PointSize[.015],Hue[0],Point[{t-r Sin[t],1-r Cos[t]}]}};

In[37]:=
     trocoide[r_,t_]:={t-r Sin[t],1-r Cos[t]};
     cicloide[t_]:=trocoide[1,t]

In[38]:=
     cic[t_]:=ParametricPlot[Evaluate[cicloide[u]],{u,0,t},
     PlotStyle->Hue[0], DisplayFunction->Identity];

In[39]:=
     Do[Show[{Graphics[{recta,rueda[t],varilla[1,t]}],cic[t]},
     AspectRatio->Automatic,DisplayFunction->$DisplayFunction,
     PlotRange->{{-2,14},{0,2.1}}],{t,.01,4Pi+.01,2Pi/10}];
```

```
In[40]:=
     troc[r_,t_]:=ParametricPlot[Evaluate[trocoide[r,u]],{u,0,t},
     PlotStyle->Hue[0], DisplayFunction->Identity];

In[41]:=
     graftroc[r_]:=Do[Show[{Graphics[{recta,rueda[t],
     varilla[r,t]}],troc[r,t]},AspectRatio->Automatic,
     DisplayFunction->$DisplayFunction,
     PlotRange->{{-2,14},{Min[-.1,.9-r],Max[2.1,1.1+r]}}],
     {t,.01,4Pi+.01,2Pi/10}];

In[42]:=
     graftroc[1.8]

In[43]:=
     graftroc[.7]
```