```
ToCharacterCode["hola\nhola"]
```

```
{104, 111, 108, 97, 10, 104, 111, 108, 97}
```

```
FromCharacterCode[%]
```

```
hola
hola
```

■ LISTA DE CARACTERES DE UNA CADENA

Characters[string] devuelve las lista de caracteres de string
StringJoin[{n1,n2,...}] construye la cadena a partir de la lista de caracteres

```
Characters["hola"]
```

```
{h, o, l, a}
```

```
StringJoin[%]
```

```
hola
```

De este modo, pasando de la cadena a la lista de caracteres, podemos usar sobre "strings" las instrucciones que actúan sobre listas. Después de la manipulación deseada, volvemos a la cadena de caracteres.

APÉNDICE 2: GRÁFICOS 2D y 3D.

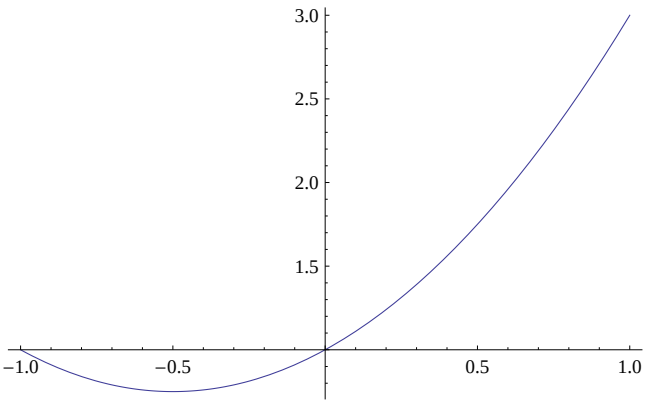
GRÁFICAS 2D

■ EL COMANDO PLOT

El comando para dibujar gráficas de funciones en una variable es **Plot**. La sintaxis es la siguiente:
Plot[f,{x,xmin,xmax}] dibuja la función **f** que depende de **x** para los valores de **x** comprendidos entre **xmin** y **xmax**.

■ Ejemplo

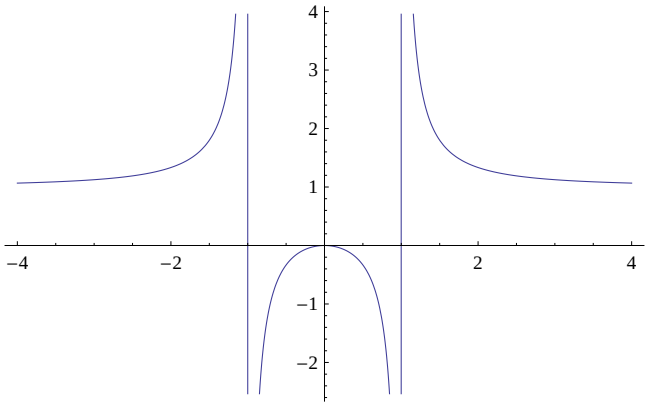
```
Plot[x^2 + x + 1, {x, -1, 1}]
```



Mathematica no sólo dibuja funciones acotadas, sino que también dibuja funciones con singularidades.

- **Ejemplo**

```
Plot[x^2 / (x^2 - 1), {x, -4, 4}]
```



Obsérvese la diferencia de escalas en ambos ejes en el ejemplo anterior. El programa elige en cada caso la escala más conveniente para realizar la gráfica, aunque esta escala se puede cambiar usando la opción **AspectRatio**.

■ COMO SE REALIZA UN GRÁFICO

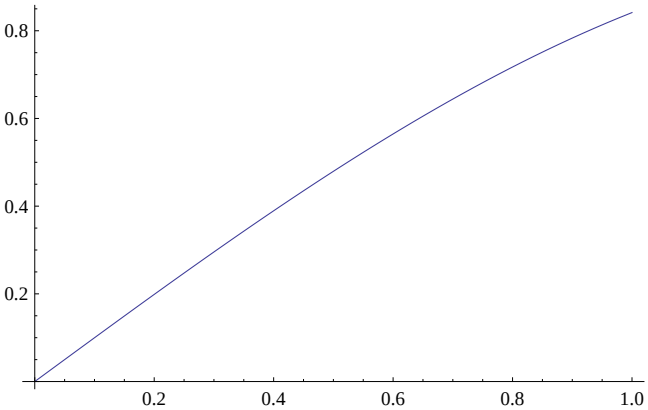
Para dibujar la gráfica de una función, el programa *Mathematica* la evalúa en un número de puntos igualmente separados en su dominio de definición; dicho número está determinado por el valor de la variable **PlotPoints**, y su valor por defecto es 50. Una vez dibujados los correspondientes puntos de la curva, considera tres puntos consecutivos y calcula el ángulo que forman los segmentos que los unen; si este ángulo es próximo a 180º, une los tres puntos con un segmento, mientras que en caso contrario repite el proceso con cada par de puntos, y así sucesivamente hasta que encuentra un ángulo próximo a 180º o llega al máximo número de subdivisiones permitido, que viene dado por el valor de la variable **MaxRecursion**. Normalmente los valores por defecto que usa el programa dan los resultados adecuados, pero puede ocurrir que no.

■ EL COMANDO SHOW

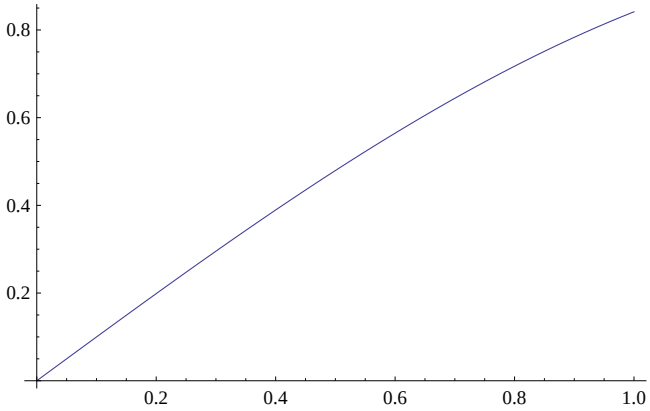
Con el comando **Show** podemos recuperar una gráfica que ya ha sido dibujada, sin que Mathema-tica tenga que dibujarla de nuevo.

- **Ejemplo**

```
AA = Plot[Sin[x], {x, 0, 1}]
```



```
Show[AA]
```



Incluso con el comando **Show** podemos modificar algunos de las opciones, las que afectan al aspecto de la función y no al algoritmo de muestreo.

■ OPCIONES EN UN GRÁFICO

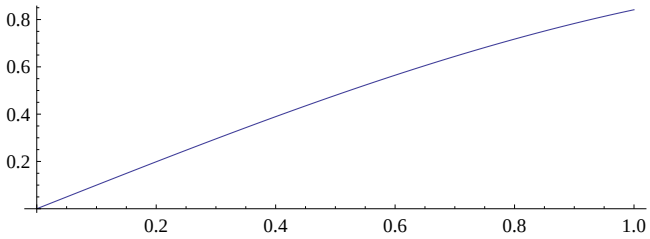
Una función como **Plot** tiene numerosas opciones que se pueden modificar. Para obtener un buen resultado, en ocasiones, es necesario probar con varias secuencias de los valores de los parametros. Todas las opciones tienen valores por defecto, valores que toman si no especificamos otro distinto. Para cambiar el valor por defecto de una opción escribimos a continuación de **{x,xmin,xmax}**, separado por comas, **Opción→valor**.

Las opciones para gráficos se pueden dividir en dos grupos: opciones que se pueden modificar con **Show** (sin necesidad de volver a dibujar la función) y opciones que no se pueden modificar con **Show** (opciones para las que hay que dibujar de nuevo la función usando **Plot**).

OPCIONES QUE SE PUEDEN MODIFICAR CON SHOW

AspectRatio. Indica la proporción entre la altura y la anchura. El valor por defecto es **1/Golden Ratio** para las gráficas en el plano y **Automatic** en dimensión 3. Los valores posibles son: cualquier numero real o **Automatic** (elige la escala que considera más adecuada).

```
BB = Show[AA, AspectRatio → 1/3]
```

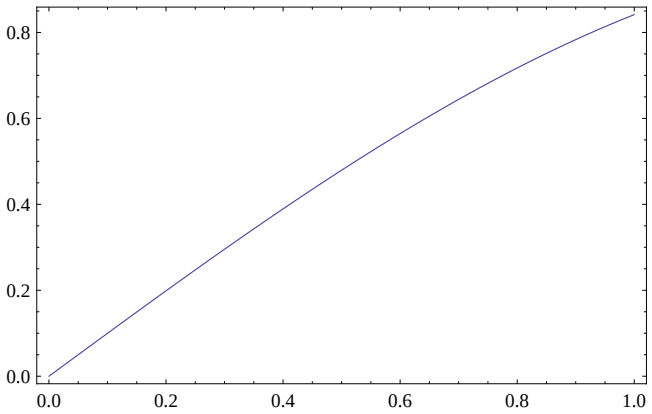


Axes. Indica si el dibujo incluye ejes o no. El valor por defecto es **True**, es decir, dibuja los ejes. Valores posibles: **True**, **False**.

AxesOrigin. Indica el punto donde colocamos el origen. Por defecto es **Automatic**, es decir, coloca el origen donde le parece más adecuado, siguiendo algoritmos internos. Valores posibles: cualquier punto del plano introducido en la forma **{x0,y0}**.

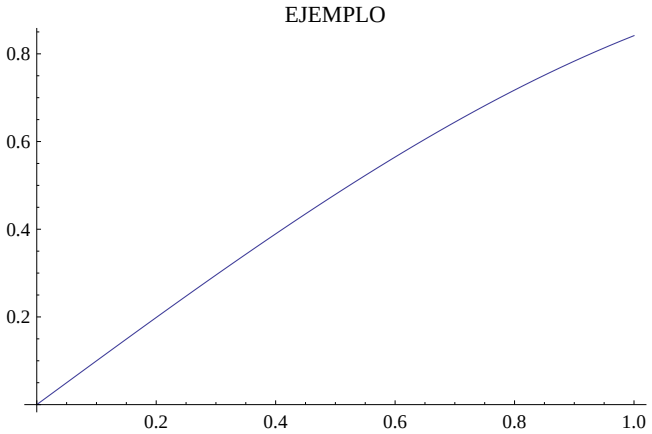
Frame. Indica si se dibuja una caja alrededor del dibujo o no. El valor por defecto es **False**. También es posible asignarle el valor **True**.

```
cc = Show[AA, Frame → True]
```



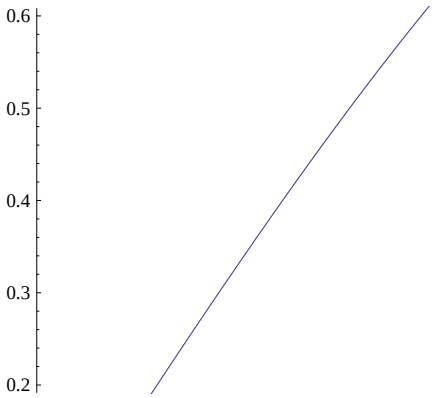
PlotLabel. Incluye una expresión como etiqueta para el dibujo. El valor por defecto es **None**. Para incluir una expresión la escribimos entre comillas.

```
DD = Show[AA, PlotLabel → "EJEMPLO"]
```



PlotRange. Indica el rango de coordenadas que se incluye en el dibujo. El valor por defecto es **Automatic** y nos muestra el rango que considera más oportuno, la parte más significativa del dibujo. Los posibles valores son: **All** (nos muestra todos los puntos), **{ymin,ymax}** (nos muestra aquellas partes de la función que se encuentran entre **ymin** e **ymax**).

```
EE = Show[AA, PlotRange → {.2, .6}, AxesOrigin → {0, 0}]
```

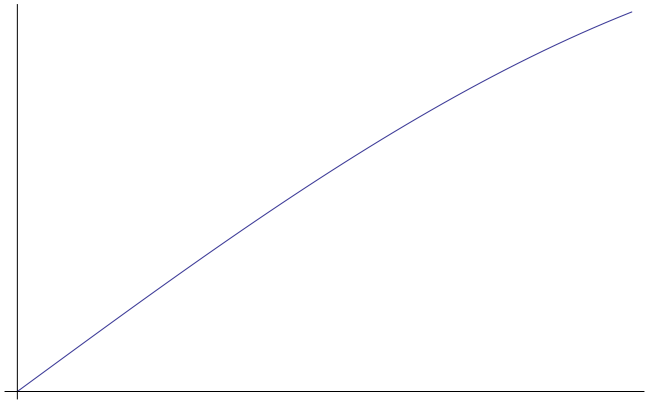


Ticks determina qué marcas se hacen en los ejes. El valor por defecto es **Automatic** y determina aquellas que, por algoritmos internos, el programa encuentra más adecuadas. Podemos especificar:

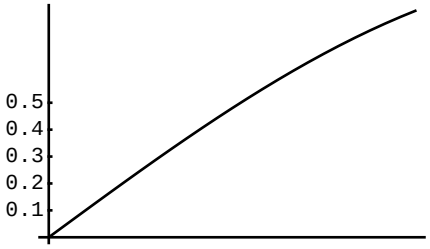
- None.Si no queremos marcas en los ejes

- {{x1, ..., xn}, {y1, ..., yn}}, donde **{x1, ..., xn}** son las marcas para el eje OX y **{y1, ..., yn}** son las marcas para el eje OY.

```
FF = Show[AA, Ticks → None]
```



```
FF = Show[AA, Ticks → {None, {.1, .2, .3, .4, .5}}]
```




```
- Graphics -
```

Hay otras opciones para especificar etiquetas en los ejes, en la caja, o para dibujar una cuadrícula en el gráfico y otras muchas que no vamos a estudiar de momento.

OPCIONES QUE NO PODEMOS MODIFICAR CON SHOW

PlotPoints. Indica el número mínimo de puntos donde muestrea la función. Por defecto es 50, aunque podemos indicar cualquier número.

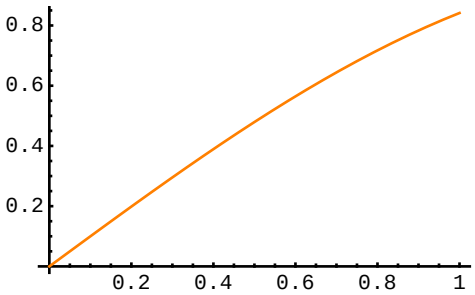
MaxRecursion. Indica el número máximo de subdivisones recursivas permitidas. El valor por defecto es 6, pero podemos indicar cualquier número.

Cuanto mayores sean **PlotPoints** y **MaxRecursion** más preciso es el gráfico de la función, aunque también **Plot** se evalua más lentamente; como siempre debemos buscar un equilibrio entre rapidez de ejecución y precisión.

PlotStyle. Mediante esta opción se puede cambiar el color, el grosor y el aspecto de la gráfica. El valor por defecto es **Automatic**. Las posibilidades para **PlotStyle** son:

- **RGBColor[, ,]**. Entre las comas se escriben tres números entre 0 y 1 que indican la mezcla de los colores rojo, verde y azul.

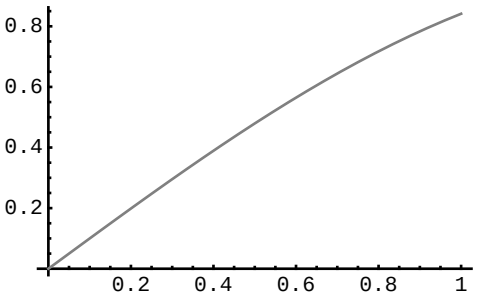
```
GG = Plot[Sin[x], {x, 0, 1}, PlotStyle → RGBColor[1, 0.5, 0]]
```




```
- Graphics -
```

- **GrayLevel**. Indicamos un número entre 0 y 1 que mide el tono de gris.

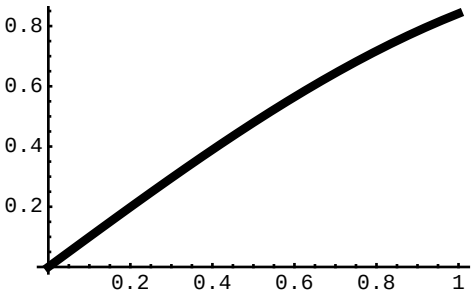
```
HH = Plot[Sin[x], {x, 0, 1}, PlotStyle → GrayLevel[.5]]
```




```
- Graphics -
```

- **Thickness**. Representa el grosor de la línea utilizada. Mide la razón entre el ancho de la línea y el de toda la gráfica. Puede tomar valores entre 0 y 1.

```
II = Plot[Sin[x], {x, 0, 1}, PlotStyle → Thickness[.02]]
```




```
- Graphics -
```

NOTA

Colocando el puntero del ratón y haciendo "clik" sobre un gráfico ya dibujado aparece alrededor del dibujo una cuadrícula. Si mantenemos pulsada la tecla Ctrl y movemos el puntero sobre la cuadrícula anterior, en la parte inferior izquierda del libro de notas aparecen las coordenadas de los puntos correspondientes.
Esto nos puede servir para determinar máximos, mínimos y asíntotas de la función que queremos dibujar

■ EJERCICIO 1

Representar las gráficas de las siguientes funciones de modo que se observen sus asíntotas, sus cortes con los ejes, los máximos y mínimos (esto se consigue aumentando la longitud del intervalo o cambiando las opciones de **Plot** o **Show**). Probar con diferentes secuencias hasta que se obtenga la gráfica más adecuada. Para cada secuencia observar como cambia el gráfico.
Moviéndote por el gráfico ya realizado, como se indica en la nota anterior, averigua, aproximadamente, los máximos, mínimos, asíntotas y cortes con los ejes de cada función.

(a) f[x] = $\frac{x^3+x}{x^5-1}$

(b) g[x] = x+Log[x^2+1]

(c) h[x] = $\frac{1}{2}$ Sin[2x]+Cos[x]

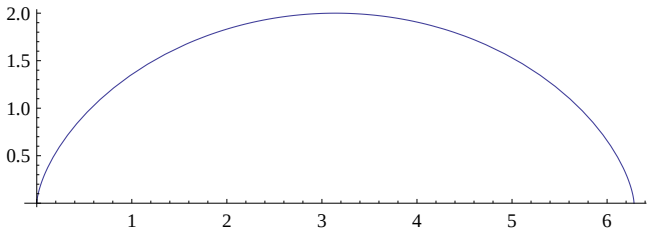
■ CURVAS DADAS EN FORMA PARAMÉTRICA

Para realizar gráficos de curvas dadas en forma paramétrica lo hacemos con:
ParametricPlot[{fx,fy},{t,tmin,tmax}].
Es posible modificar el aspecto de una gráfica utilizando las opciones que hemos visto en el apartado anterior.

Ejemplo

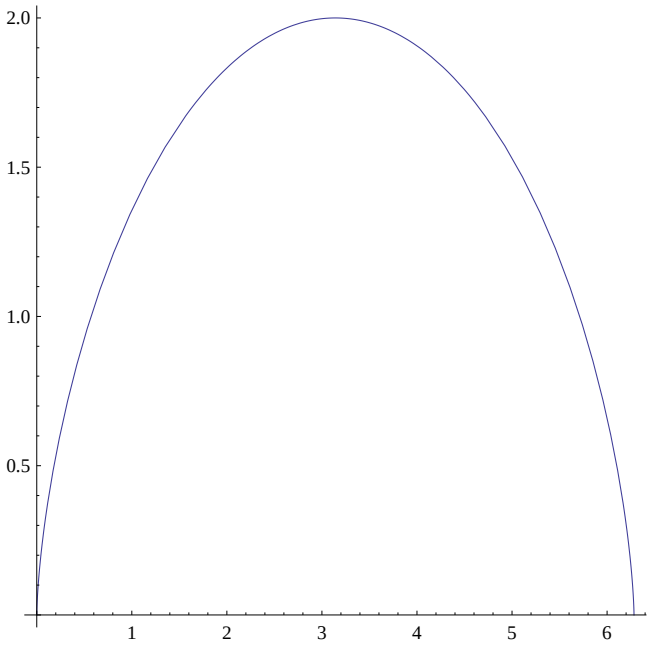
Dibujamos la gráfica de la cicloide (recordamos que la cicloide se define como la trayectoria descrita por un punto fijo P de una circunferencia cuando ésta se mueve por una recta fija sin desplazarse)

```
ParametricPlot[{t - Sin[t], 1 - Cos[t]}, {t, 0, 2 π}]
```



Si queremos la curva para otra escala, modificamos **AspecRatio** a 1, usando **Show**.

```
Show[%, AspectRatio → 1]
```



■ EJERCICIO 2

Obtener la gráfica de la astroide de ecuaciones paramétricas: x=a Cos[t]^3, y=a Sin[t]^3, para t entre 0 y 2π. Dar a a diferentes valores y observar como cambia la curva.

■ CURVAS QUE VIENEN DADAS EN FORMA IMPLÍCITA

En versiones anteriores a la 6.0.

Para dibujar gráficas que vienen definidas implícitamente por una ecuación debemos cargar el paquete **"Graphics`ImplicitPlot`"** (en versiones anteriores a 6.0.).
La instrucción para dibujar curvas de este tipo es:
ImplicitPlot[ecuación,{x,xmin,xmax}]
Una ecuación es una expresión del tipo **expr1==expr2**. Notar que reservamos para las ecuaciones == (éste es el igual lógico). El signo = lo utilizamos para hacer asignaciones a variables.
Es posible modificar el aspecto del dibujo con las opciones vistas en apartados anteriores.

Ejemplo

Vamos a dibujar la circunferencia de centro el origen y radio 1.

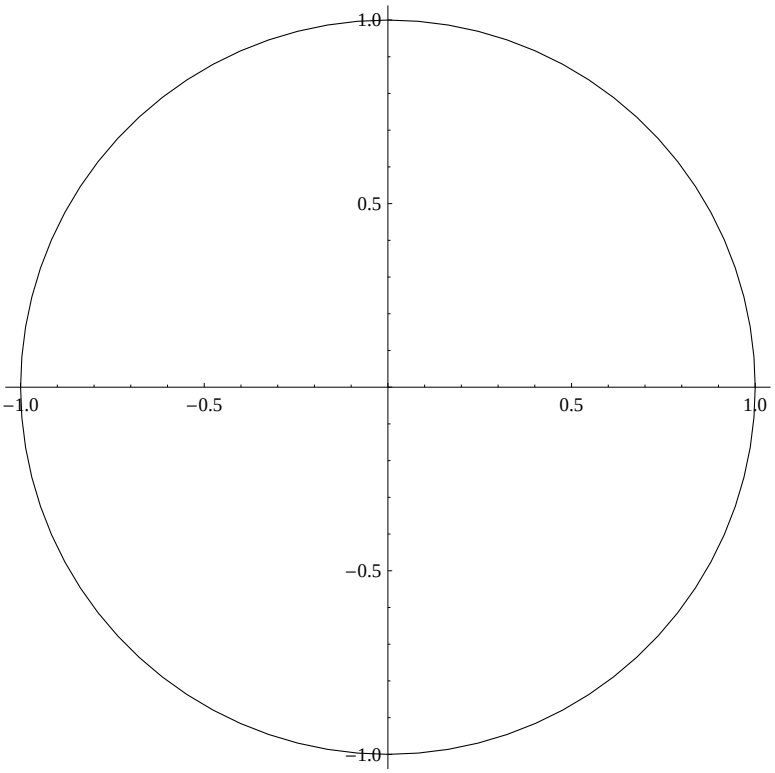
```
Needs["Graphics`ImplicitPlot`"];
VV = ImplicitPlot[x^2 + y^2 == 1, {x, -1, 1}]
```

General::obspkg:

Graphics`ImplicitPlot` is now obsolete. The legacy version being loaded may conflict with current Mathematica functionality. See the Compatibility Guide for updating information.

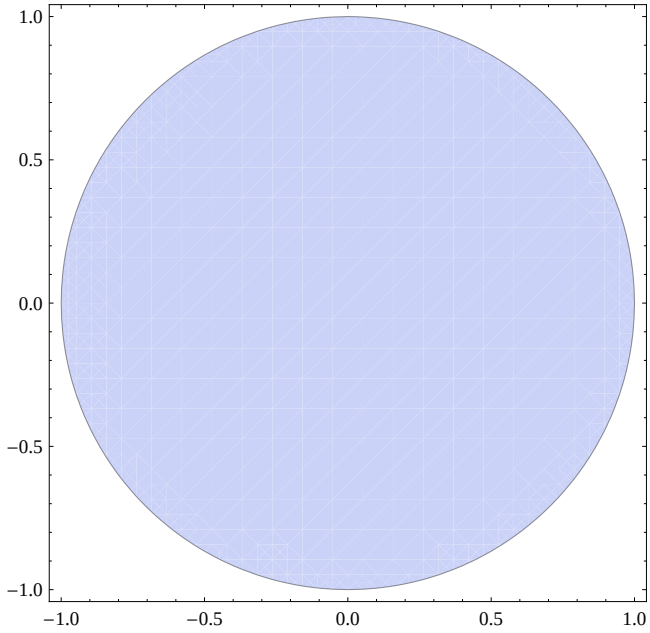
ImplicitPlot::shdw:

Symbol ImplicitPlot appears in multiple contexts {Graphics`ImplicitPlot`, Global`}; definitions in context Graphics`ImplicitPlot` may shadow or be shadowed by other definitions.

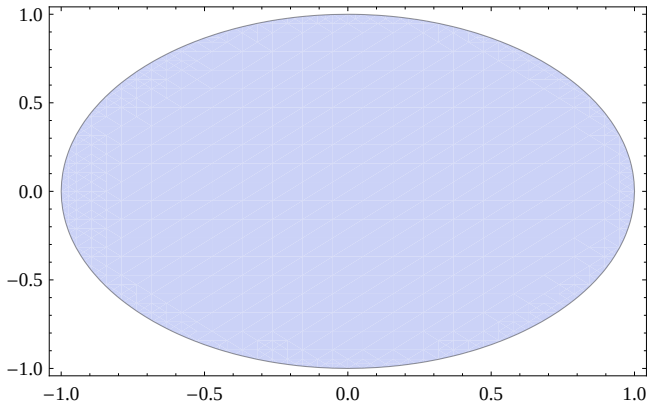


En la versión 6.0. utilizamos la instrucción RegionPlot (ver la ayuda para dibujar

```
VR = RegionPlot[x^2 + y^2 <= 1, {x, -1, 1}, {y, -1, 1}]
```




```
Show[VR, AspectRatio → 1/GoldenRatio]
```



■ EJERCICIO 3

Obtener las gráficas de las siguientes curvas dadas por sus ecuaciones implícitas:

(a) Estrofoide $y^2 = \frac{x\,(x-a)^2}{2\,a-x}$ (se entiende que la representamos eligiendo un valor para a)

(b) Cisoide $y^2 = \frac{x^3}{2\,a-x}$ (igual que en el apartado anterior probar con diferentes valores para a.

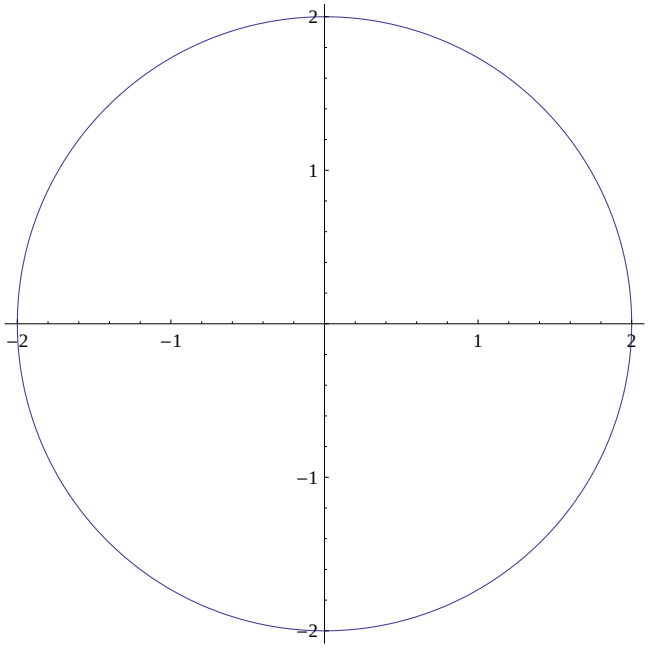
Hay que tener cuidado al elegir los limites para **x**, ya que la curva no está definida en todo **R**, sino sólo en aquellas **x** en que es posible despejar la **y** como función de **x**.

■ CURVAS QUE VIENEN DADAS EN COORDENADAS POLARES

Para dibujar una curva de la que conocemos su ecuación en coordenadas polares (r = f (t), siendo t el ángulo y r el radio polar) debemos cargar el paquete "Graphics`Graphics`" (A partir de la versión 6.0.no es necesario cargarlo). La función que dibuja gráficos en coordenadas polares es la siguiente : PolarPlot[f, {t, tmin, tmax}] dondefes la expresión del radio polar en función del ángulo, tel ángulo ytmin, tmax, los límites para el ángulo.

Ejemplo:
Dibujamos una circunferencia de centro el origen y radio 2, cuya ecuación en cordenadas polares esr == 2.

```
PolarPlot[2, {t, 0, 2 π}]
```



EJERCICIO 5

Obtener las gráficas de las siguientes funciones dadas en forma polar :

(a) Lemniscata de Bernoulli : $r = a\sqrt{2\,\mathrm{Cos}[2\,t]}$

(b) Cardiodie : $r = a\,(1 - \mathrm{Cos}[t])$ Probar con diferentes valores para a y observar como varía la gráfica de la curva.

GRÁFICAS 3D

■ EL COMANDO PLOT3D

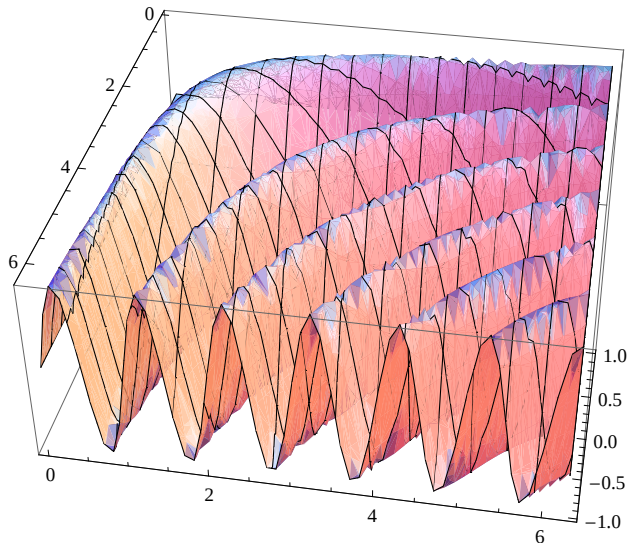
Para dibujar gráficas de funciones de dos variables se utiliza
Plot3D[f,{x,xmin,xmax},{y,ymin,ymax}] dibuja la función **f** que depende de **x** e **y** para los valores de **x** comprendidos entre **xmin** y **xmax** y los valores de **y** entre **ymin** e **ymax**.
Las opciones de Plot3D son las mismas que las de Plot.

■ Ejemplo

PlotPoints

```
Plot3D[Sin[x y], {x, 0, 2 π}, {y, 0, 2 π}]
```

PlotPoints



Colocando el puntero sobre el gráfico podemos girarlo para cambiar el punto de vista.

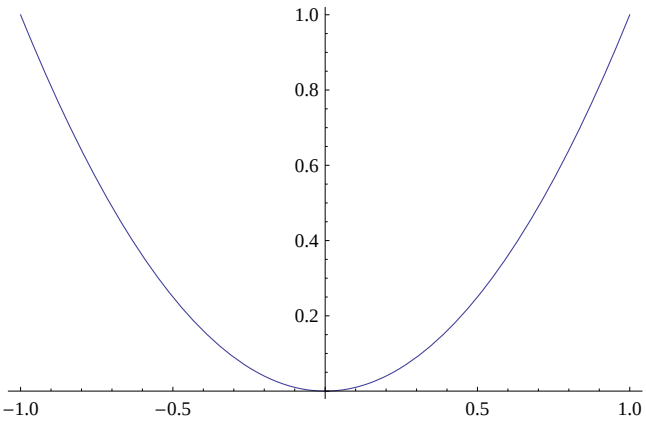
■ SUPERFICIES DE REVOLUCIÓN

En versiones anteriores a la 6.0.:
Una superficie de revolución es la superficie que genera una curva al girar alrededor de un eje. Para dibujarla debemos cargar primero el paquete **"Graphics`SurfaceOfRevolution`"**.
La función incluida en este paquete que nos dibuja la superficie resultado de girar **y=f[x]** alrededor del eje OY es la siguiente: **SurfaceOfRevolution[f,{x,xmin,xmax}]**.
También podemos girar curvas dadas en ecuaciones paramétricas con:
SurfaceOfRevolution[{fx,fy},{t,tmin,tmax}].

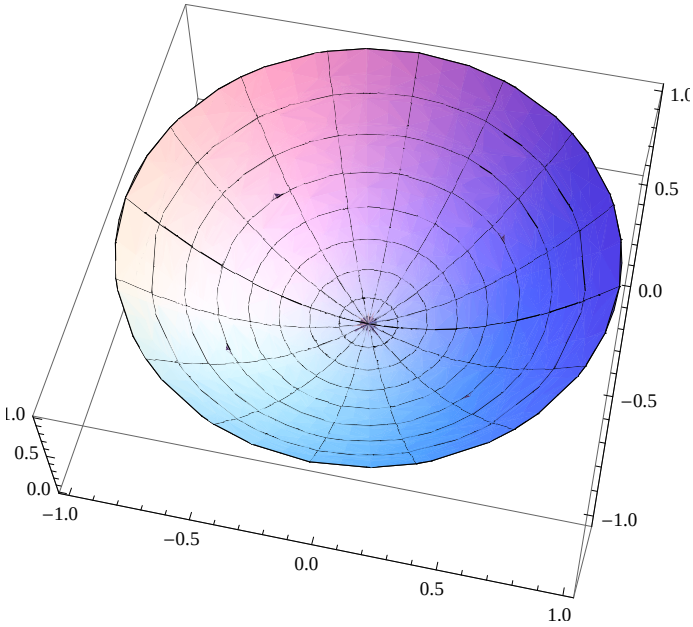
■ Ejemplo

Obtenemos la superficie resultado de girar la gráfica de la parabola $y=x^2$ alrededor del eje OY variando x entre 0 y 1.

```
Plot[x^2, {x, -1, 1}]
```



```
RevolutionPlot3D[x^2, {x, -1, 1}, ViewPoint → {0.397`, -1.807`, 2.833`}]
```



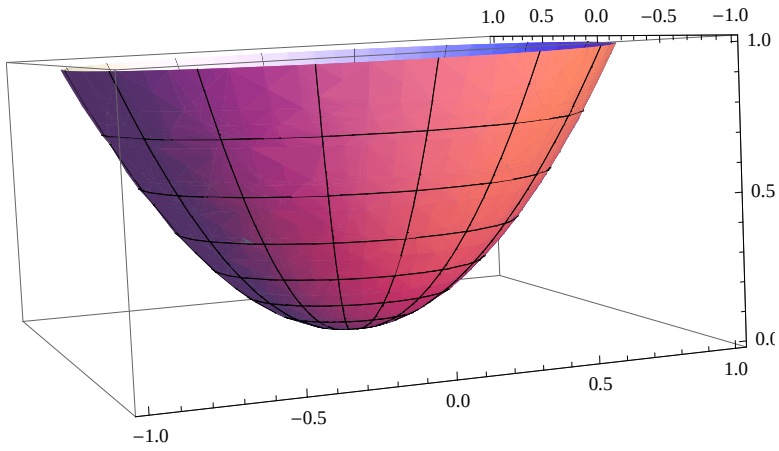

```
Needs["Graphics`SurfaceOfRevolution`"];
RevolutionPlot3D[x^2, {x, -1, 1}]
```

General::obspkg:

Graphics`SurfaceOfRevolution` is now obsolete. The legacy version being loaded may conflict with current Mathematica functionality. See the Compatibility Guide for updating information.

SurfaceOfRevolution::shdw:

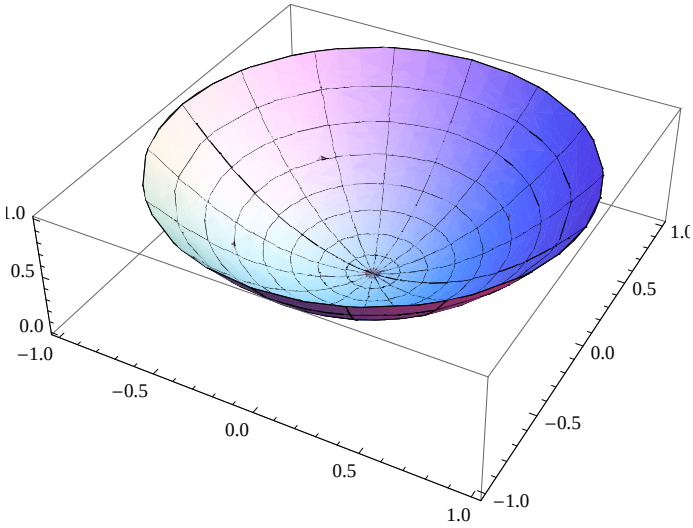
Symbol SurfaceOfRevolution appears in multiple contexts {Graphics`SurfaceOfRevolution`, Global`}; definitions in context Graphics`SurfaceOfRevolution` may shadow or be shadowed by other definitions.



En la versión 6.0.: La instrucción para dibujar la superficie que se obtiene al g

```
RevolutionPlot3D[f[x],{x,xmin,xma}]
```

```
RevolutionPlot3D[x^2, {x, -1, 1}]
```



Podemos cambiar el punto de vista de un gráfico en dimension 3.
En versiones anteriores a la 6.0.:
con la opción **ViewPoint**. Contamos con la ayuda de **"3DViewPoint Selector"** en el menú Input que nos muestra la posición de los ejes para los distintos puntos de vista.
En la versión 6.0.: Basta hacer click con el boton izquierdo del ratón y manteniendolo pulsado moverlo hasta "colocarlo" en la posición deseada.

■ EJERCICIO 4

Dibujar la superficie de revolución obtenida al girar la gráfica de la función Sin[x] alrededor del eje OY para los valores de x comprendidos entre -1 y 1.
Cambiar el punto de vista para el gráfico obtenido hasta colocar los ejes en la posición habitual.

LISTAS DE DATOS

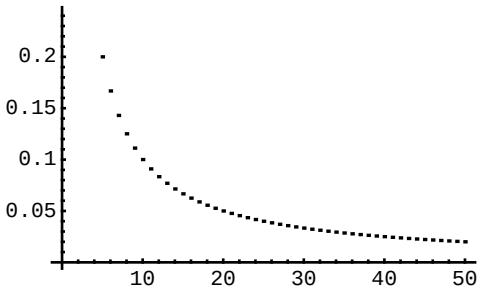
Para dibujar listas de datos contamos con los siguientes comandos:
ListPlot[{y1,y2,...yn}] dibuja los puntos `y1`, `y2`, ..., `yn` en 1, 2, ..., n.
ListPlot[{{x1,y1},{x2,y2},...{xn,yn}}] dibuja los puntos del plano {x1,y1}, {x2,y2},..., {xn,yn}.
ListPlot[{{x1,y1},{x2,y2},...{xn,yn}},PlotJoined->True] dibuja los puntos del plano {x1,y1}, {x2,y2}, ..., {xn,yn} uniéndolos con líneas.

En análisis usaremos estas funciones para dibujar términos de sucesiones de números reales. El dibujo nos permitirá en algunos casos concluir el valor del límite, además de la rapidez en la convergencia.
Para generar listas de datos contamos con la función **Table**.

■ Ejemplo

Sabemos que la sucesión $\frac{n^3+1}{n^4+5}$ tiene por límite 0. Lo comprobamos dibujando los 50 primeros términos de la sucesión.

```
ListPlot[Table[(n^3 + 1)/(n^4 + 5), {n, 50}]]
```



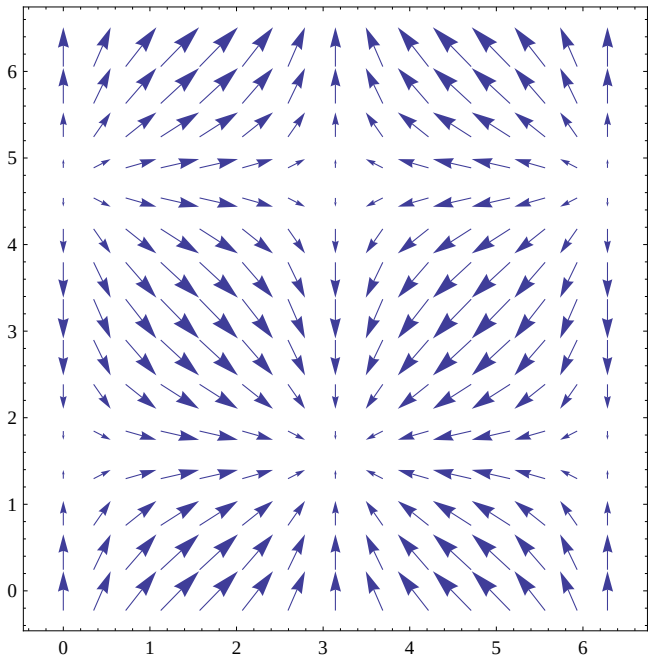

```
- Graphics -
```

DIBUJANDO CAMPOS DE DIRECCIONES

VectorPlot[{f_x, f_y}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}]
genera un dibujo del campo vectorial de componentes {f_x, f_y} como función de x,y.

Ahora dibujamos el gráfico del campo en dos dimensiones de componentes {sin (x), cos (x)}

```
VectorPlot[{Sin[x], Cos[y]}, {x, 0, 2 Pi}, {y, 0, 2 Pi}]
```



Otras funciones para dibujar campos son:

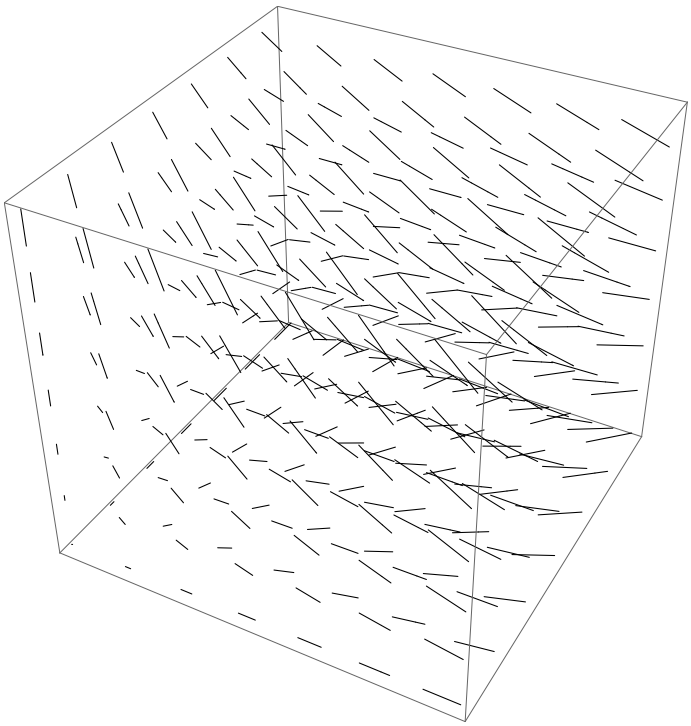
VectorPlot3D[{f_x, f_y, f_z}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}, {z, z_{min}, z_{max}}]
genera un dibujo 3D del campo vectorial de componentes {f_x, f_y, f_z} como función de x,y,z.

GradientFieldPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}]
genera un dibujo del campo gradiente de la función escalar f.

GradientFieldPlot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}, {z, z_{min}, z_{max}}]
genera un dibujo 3D del vector gradiente de la función f.

HamiltonianFieldPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}]
genera un dibujo del campo hamiltoniano asociado a la función f.

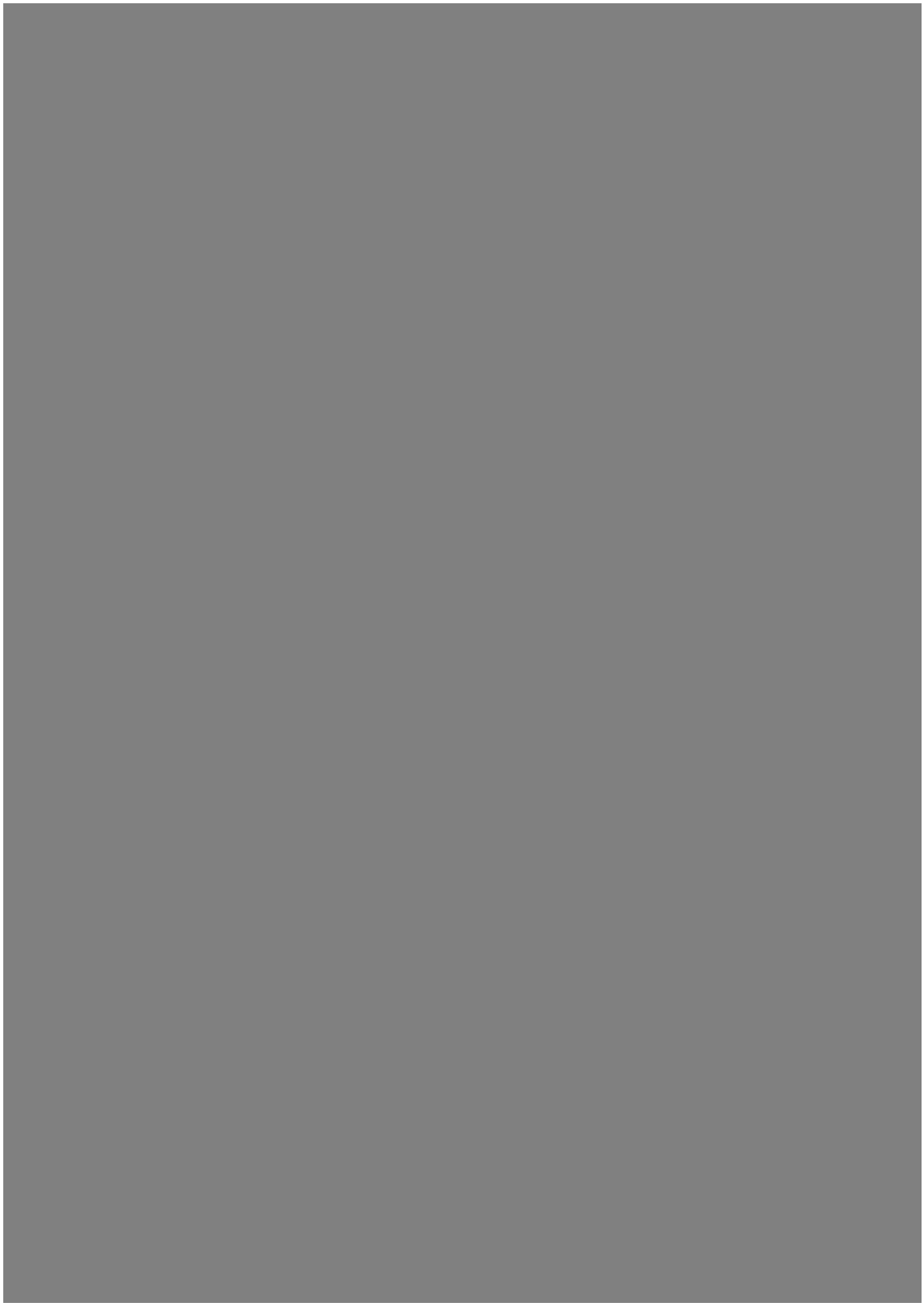
```
GradientFieldPlot3D[x^2 + y^2 - z^2, {x, 0, 1}, {y, 0, 1}, {z, 0, 1}]
```



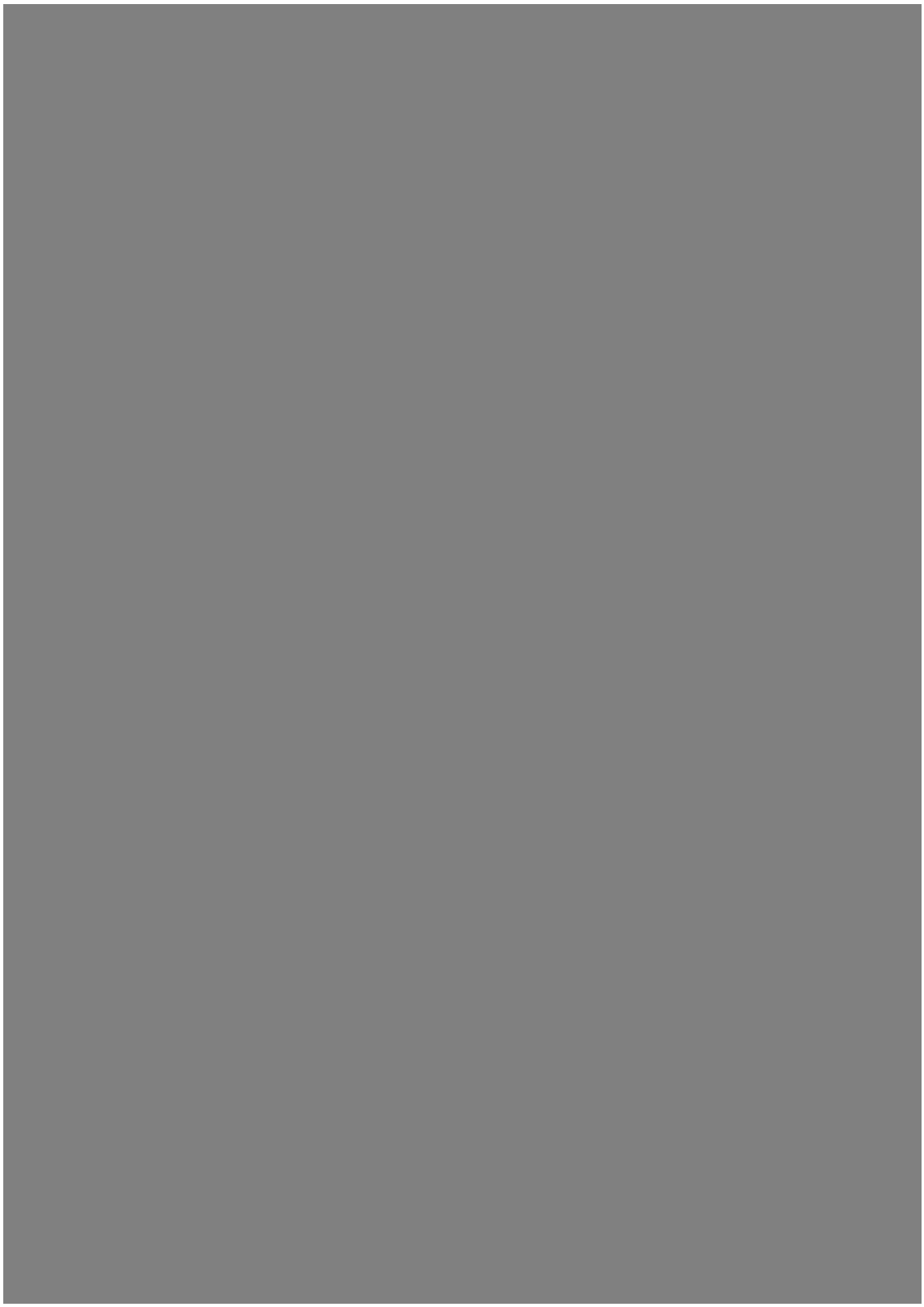
ANIMACIÓN Y DINAMIZACIÓN DE GRÁFICOS

La instrucción Manipulate permite crear animaciones interactivas con solo unas pocas lineas de código. La sintaxis de Manipulate es esencialmente la de Table

```
Manipulate[Plot[Sin[n x], {x, 0, 2 Pi}], {n, 1, 20}]
```



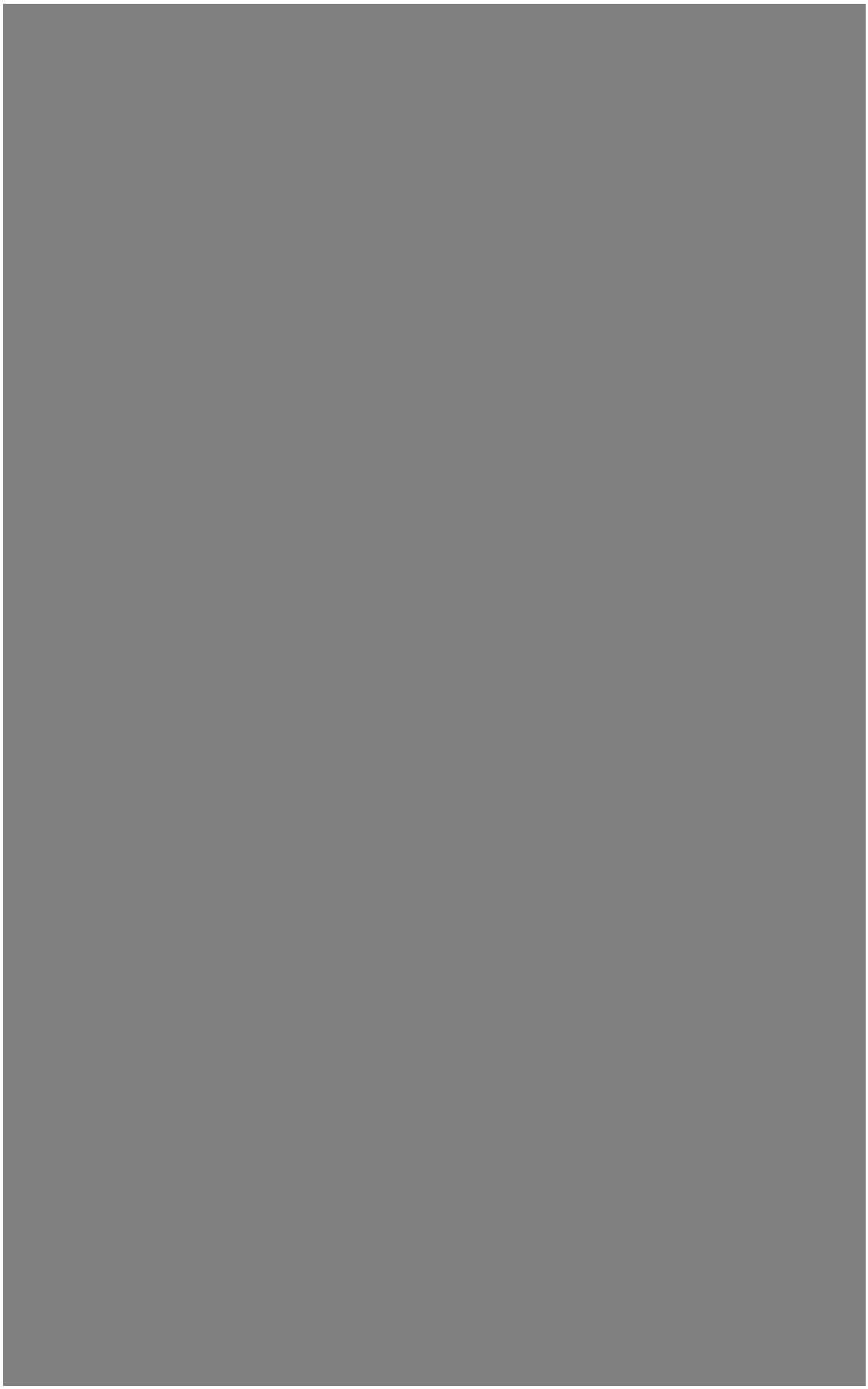
Desplazando el cursor obtenemos la gráfica de Sin[nx] para distintos valores de n. En la barra aparece un icono +; haciendo click sobre el se abre un pequeño panel de controles adicionales



El panel permite ver el valor número de la variable respecto de la que estamos realizando la animación, asi como el uso de los controles de animación

Como Table , Manipulate permite especificaciones para más de una variable.

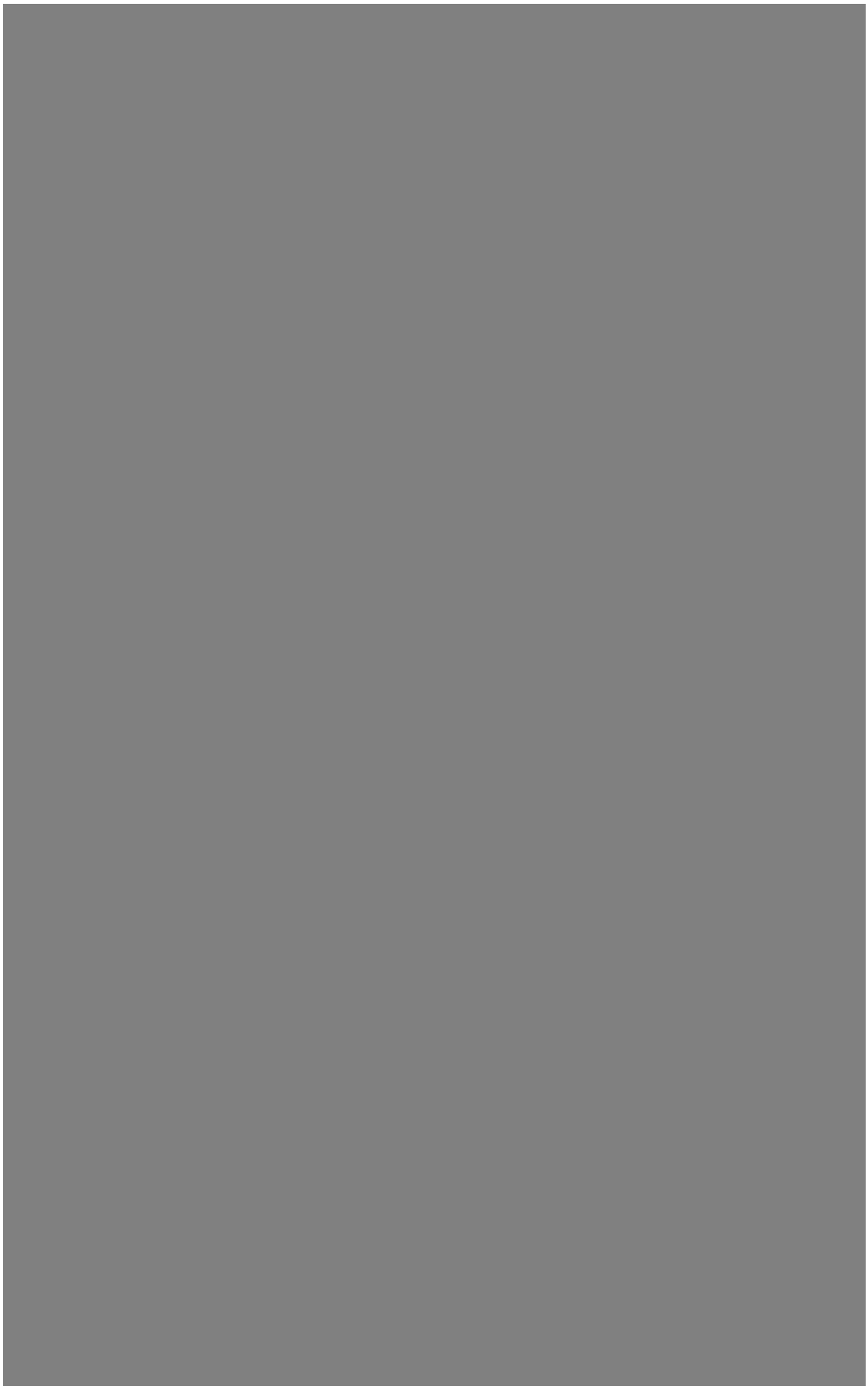
```
Manipulate[Plot[Sin[n1 x] + Sin[n2 x], {x, 0, 2 Pi}, PlotRange → 2],
 {n1, 1, 20}, {n2, 1, 20}]
```



Por defecto Manipulate utiliza el nombre de las variables para poner etiquetas a cada control. Pero si se quiere etiquetar los controles con otras etiquetas más descriptivas, se puede hacer usando especificaciones para las variables de la forma

{{var, init, label}, min, max}.

```
Manipulate[Plot[Sin[n1 x] + Sin[n2 x], {x, 0, 2 Pi}, PlotRange → 2],
 {{n1, 1, "label1"}, 1, 20}, {{n2, 1, "label2"}, 1, 20}]
```



En el tutorial, tutorial/AdvancedManipulateFunctionality, encontrarás posibilidades más avanzadas de uso para Manipulate.

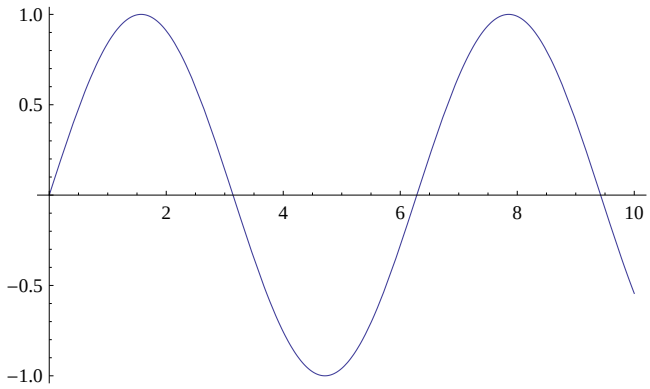
EXPORTANDO GRÁFICOS REALIZADOS CON MATHEMATICA A OTROS FORMATOS

Mathematica permite exportar gráficos y animaciones a otros formatos, conservando la dinamización siempre que sea posible.La instrucción mas simple para exportar gráficos (en general, cualquier tipo de dato) es

Export["file.ext", exp] exporta expr a file, convirtiendolo al formato correspondiente a la extensión del archivo ext.
Export[file, exp,"format"] exporta exp en el formato especificado

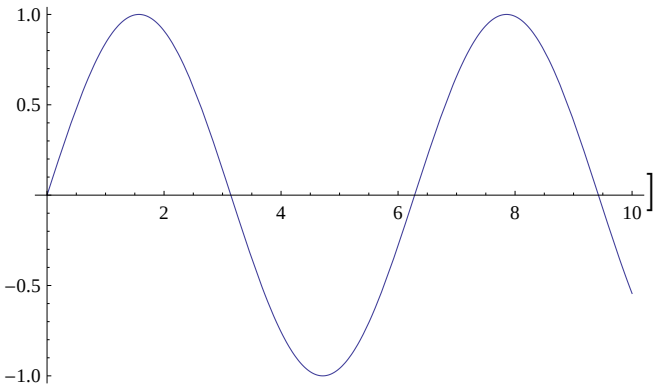
Ejemplos :

```
A = Plot[Sin[x], {x, 0, 10}]
```



Para exportar este gráfico a un archivo gif podemos hacerlo de las siguientes maneras

```
Export["test.gif",
```




```
]
```

```
Export["test.gif", A]
```

```
Export["test.gif", Plot[Sin[x], {x, 0, 10}]]
```

Podemos exportar a los siguientes formatos gráficos

```
 "EPS"          Encapsulated PostScript (.eps)
 "PDF"  Adobe Acrobat portable document format (.pdf)
 "SVG"         Scalable Vector Graphics (.svg)
"PICT"                 Macintosh PICT
 "WMF"          Windows metafile format (.wmf)
"TIFF"               TIFF (.tif, .tiff)
 "GIF"           GIF and animated GIF (.gif)
"JPEG"               JPEG (.jpg, .jpeg)
 "PNG"                PNG format (.png)
 "BMP"          Microsoft bitmap format (.bmp)
 "PCX"                PCX format (.pcx)
 "XBM"          X window system bitmap (.xbm)
 "PBM"          portable bitmap format (.pbm)
 "PPM"          portable pixmap format (.ppm)
 "PGM"         portable graymap format (.pgm)
 "PNM"          portable anymap format (.pnm)
"DICOM"   DICOM medical imaging format (.dcm, .dic)
 "AVI"       Audio Video Interleave format (.avi)
```