Evolución Temporal

Evolución de estados estacionarios de Pozo Infinito Unidimensional

Darío Mitnik

Funciones de Pozo Infinito

$\psi(x) = \sqrt{\frac{2}{L}} \sin(\frac{n\pi}{L}x)$

In [47]:

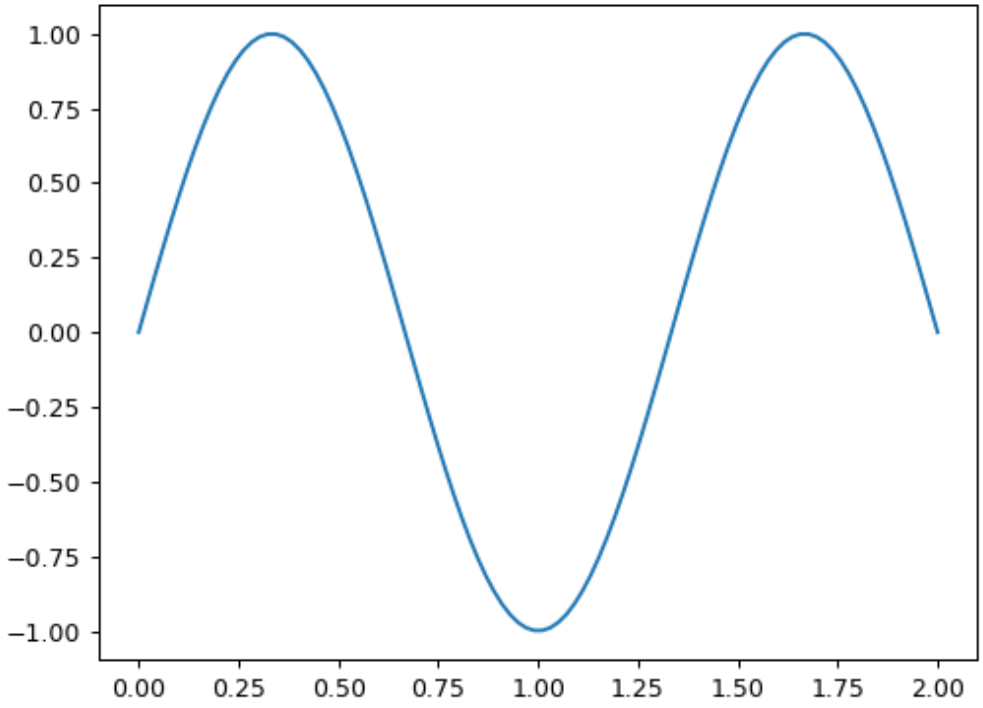
```
import numpy as np
import matplotlib.pyplot as plt
```

In [48]:

```
# Definición de funciones de onda

def psi(n,L,x):
    y = np.sqrt(2/L) * np.sin(n * np.pi * x / L)
    return y

# Energías
def ener(n,L):
    y = 1./2. * (n * np.pi / L)**2
    return y
```

In [49]:

```
L = 2
npts = 100
x = np.linspace(0,L,npts)
y = psi(3,L,x)

plt.plot(x,y)
plt.show()
```



In [51]:

```
# Chequeo Ortonormalidad

yy = psi(3,L,x)*psi(3,L,x)
sum = np.trapz(yy,x)
print('sum=',sum)
```

```
('sum=', 1.0)
```

Animación

In [52]:

```
# Copiamos código general de animación

%matplotlib notebook

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from IPython.display import HTML

def animar(f,x0=0,xf=1,dx=0.01,t0=0,tf=1,dt=0.01, ym=-2, yM= 2):
    """
    Funcion para realizar animaciones. Toma como entrada una funcion
        f(x,t)
    donde el primer argumento es la coordenada espacial y el segundo la tempora
l.

    Ademas pide como argumento de entrada
        x0 - la coordenada x inicial
        xf - la coordenada x final
        dx - el valor entre dos puntos sucesivos de x
        t0 - la coordenada t inicial
        tf - la coordenada t final
        dt - el valor entre dos puntos sucesivos de t
        ym - Valor minimo para y
        yM - Valor maximo para y


    Esta harcodeado el tiempo entre dos frames en 50 ms (20 frames por segundo).

    Se puede cambiar cambiando el valor de `interval` que aparece mas abajo.
    """

    # Definimos los intervalos para graficar
    nf = int( (xf-x0)/dx + 1)
    nt = int( (tf-t0)/dt + 1)
    x = np.linspace(x0,xf,nf)
    t = np.linspace(t0,tf,nt)


    # Ponemos los ejes para la figura
    fig, ax = plt.subplots()

    ax.set_xlim((x0, xf))
    ax.set_ylim((ym, yM))

    line, = ax.plot([], [], lw=2)

    # Definimos la funcion que crea un grafico vacio
    def init():
        line.set_data([], [])
        return (line,)

    # Funcion que crea cada frame
    def animate(i):
        y = f(x,i)
```

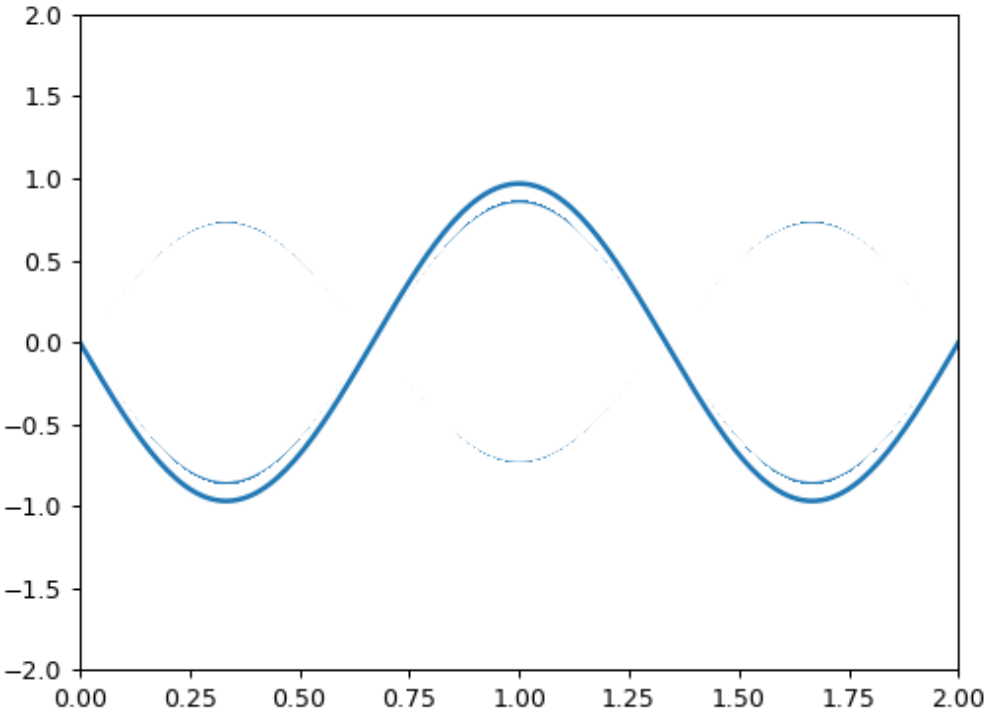
```
        line.set_data(x, y)
        return (line,)
    # Creamos el objeto animado
    anim = animation.FuncAnimation(fig, animate, init_func=init,
                                   frames=t, interval=20, blit=True)
    plt.show()
    return anim
```

In [108]:

```
# Definición de la función evolución con todas las variables

def Yev(x,t,n,L):

    # n: número cuántico principal
    # L: ancho del pozo
    # iim: Si querés plotear la parte imaginaria (default=real)

    y = psi(n,L,x)
    en = ener(n,L)
    phase = -1j * en * t
    yt = y*np.exp(phase)
    if (iim):
        yt = np.imag(yt)
    return yt

# Acá se convierte la función Yev en una f(x,t) -- sólo x y t como variables
L = 2
n = 3
f = lambda x,t: Yev(x,t,n,L)
```

In [109]:

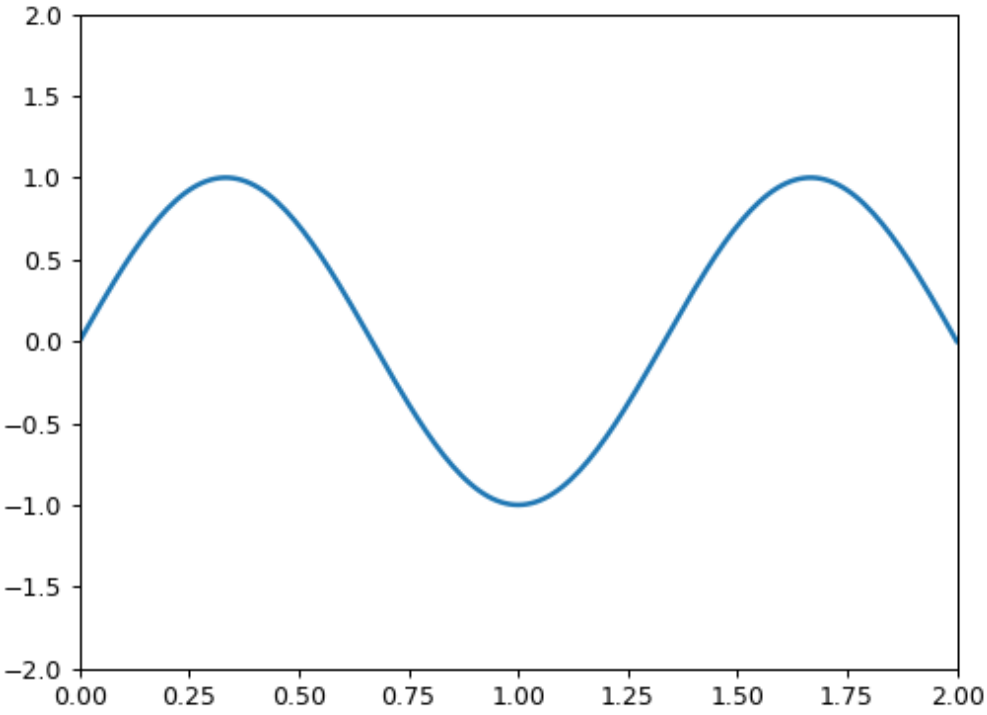
```
anim = animar(f,\
              x0=0,xf=L)
```

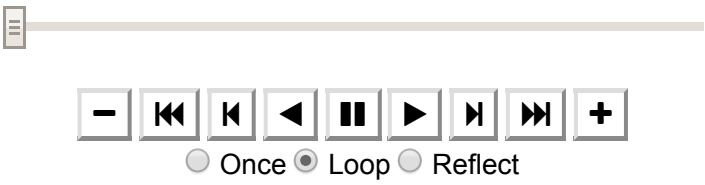


In [102]:

```
# Visualización Controlada

# Si ffmpeg está instalado se puede usar:
# HTML(anim.to_html5_video())

HTML(anim.to_jshtml())
```

Out[102]:





Once Loop Reflect

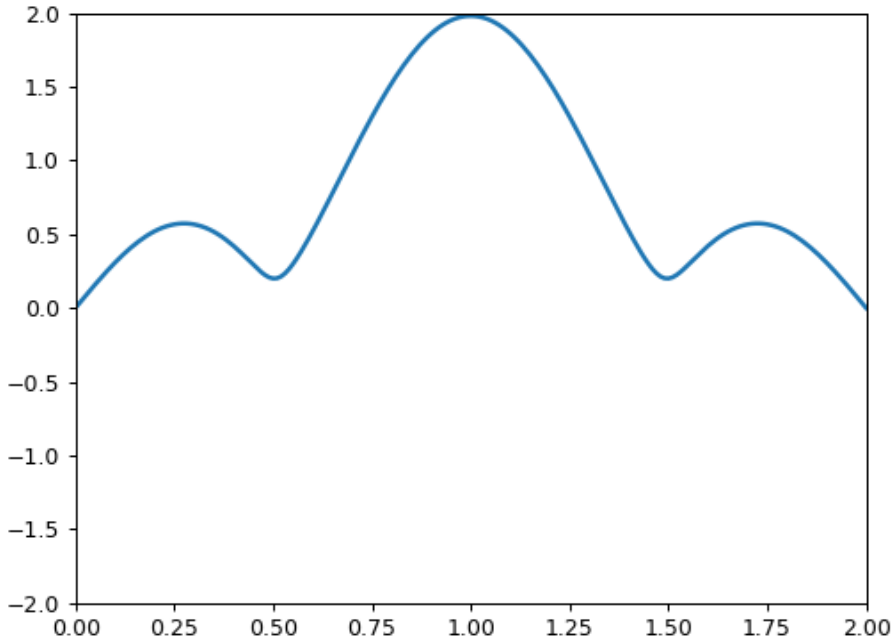
Combinación Lineal de Funciones

In [120]:

```
L = 2
n1 = 1
n2 = 3
f = lambda x,t: np.abs(Yev(x,t,n1,L) + Yev(x,t,n2,L))
```

In [118]:

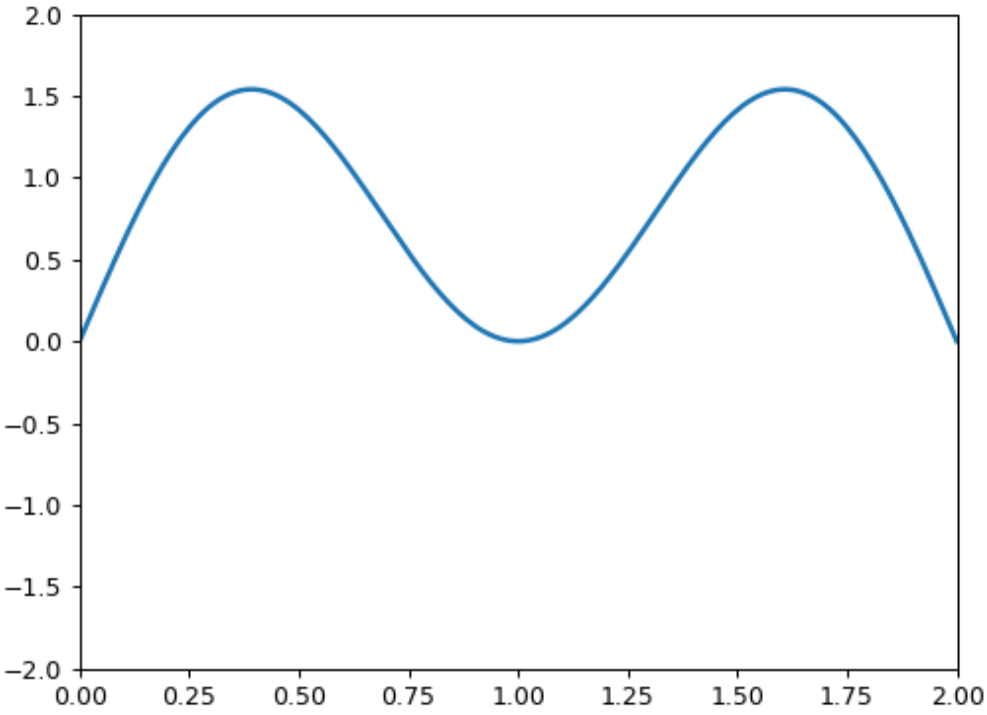
```
anim = animar(f,\
              x0=0,xf=L,tf=10)
```



In [121]:

```
# Visualización Controlada

# Si ffmpeg está instalado se puede usar:
# HTML(anim.to_html5_video())

HTML(anim.to_jshtml())
```

Out[121]:



Once Loop Reflect