Física del Attosegundo

Ejercicios Computacionales

Darío Mitnik

In [9]:
```
# Import libraries

import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

```

1) Forward Derivative

Discretizamos una función $f_i = f(x_i)$ y la escribimos como un vector $\vec{f}$:

$$\vec{f} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix}$$

La derivada "hacia adelante" se define como:

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2\,\Delta x}$$

Definiendo la matriz D:

$$\mathbf{D} = \frac{1}{2\,\Delta x} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Podemos escribir la operación de derivación como el producto de la matriz D por el vector f

$$\vec{f}' = \mathbf{D}\,\vec{f}$$

In [10]:
```python
# Función que deriva una función y(x) hacia adelante

def forwarderiv(y,x):

    # grid (se asume uniforme)
    h = x[1]-x[0]
    n = len(x)-1

    # Matriz D
    ones = np.ones(n) # vector (1,1,....1)
    D = np.diag(ones,1) + np.diag(-ones,-1)

    # Multiplicación D.y
    Deriv = np.dot(D,y) / (2 * h)

    return Deriv
```

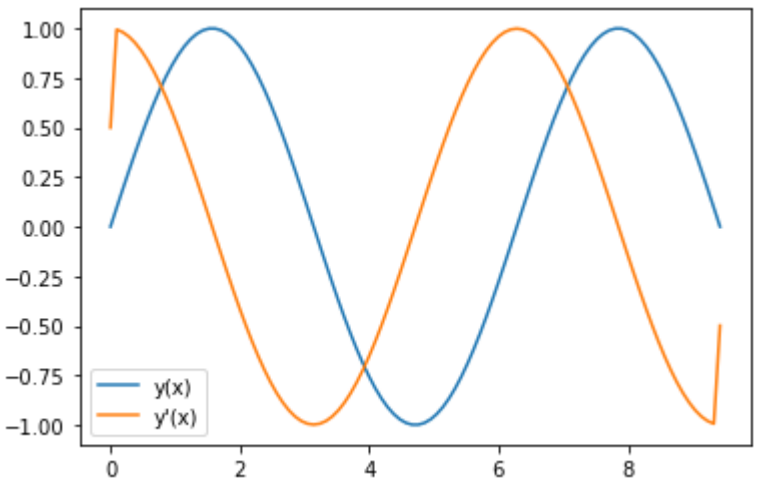
In [11]:
```python
# Ejemplo de cómo utilizar la función forwarderiv

# array definitions (0,3 Pi)
nsize = 100
xmin=0
xmax=3*np.pi
x = np.linspace(xmin,xmax,nsize)

# derivative of Sin(x)
y = np.sin(x)
yp = forwarderiv(y,x)

# Plot
plt.plot(x,y,label='y(x)');
plt.plot(x,yp,label="y'(x)");
plt.legend();

# check results
ydif = yp - np.cos(x)
```



Ejercicio 1: Derivadas

- Resolver los problemas que hay en los bordes
- Graficar los errores en función del tamaño del grid (*nsize*)
- Repetir reemplazando la derivada *forward* por *backward* y *central difference*.

- Escribir una función que calcule la derivada segunda y comprobar los resultados.

2) Integrales

Para calcular el área bajo la curva $y(x)$ podemos hacer una aproximación de rectángulos:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} f_i\,\Delta x$$

También podemos hacer una aproximación de trapecios:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{N-1} \frac{f_{i+1}+f_i}{2}\,\Delta x = \\ = \frac{1}{2}f_0 + \sum_{i=1}^{N-1} f_i + \frac{1}{2}f_N$$

In [12]:
```
# Función que integra una función y(x)
def integraterect(y,x):

    # grid (se asume uniforme)
    h = x[1]-x[0]
    n = len(x)

    # Integral
    S = np.sum(y) * h

    return S
```

In [13]:
```
# Test: Integral de Sin(x) enter 0 y 3 Pi:

yint = integraterect(y,x)
print(yint)

```

```
1.9984892721876022
```

Ejercicio 2: Integrales

- Graficar los errores en función del tamaño del grid (*nsize*)
- Repetir reemplazando la integral por rectángulos por la integral por trapecios.

3) Hamiltoniano

$$\hat{H}\,\varphi(x) = -\frac{1}{2}\frac{d^2\,\varphi(x)}{dx} + V(x)\,\varphi(x)$$

$$\mathbf{H} = \begin{pmatrix} \frac{1}{(\Delta x)^2} + V_0 & -\frac{1}{2(\Delta x)^2} & 0 & 0 & \dots \\ -\frac{1}{2(\Delta x)^2} & \frac{1}{(\Delta x)^2} + V_1 & -\frac{1}{2(\Delta x)^2} & 0 & \dots \\ 0 & -\frac{1}{2(\Delta x)^2} & \frac{1}{(\Delta x)^2} + V_2 & -\frac{1}{2(\Delta x)^2} & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & -\frac{1}{2(\Delta x)^2} & \frac{1}{(\Delta x)^2} + V_N \end{pmatrix}$$

$$\mathbf{H}\,\vec{f} = E\,\vec{f}$$

Ejercicio 3: Hamiltoniano

- Escribir una función que calcule el Hamiltoniano de un oscilador armónico unidimensional

4) Solución del Hamiltoniano: Autofunciones y Energías

In [ ]:
```
# Una vez que se construye el Hamiltoniano (Matriz H)
# se lo diagonaliza de la siguiente manera:

from numpy.linalg import eigh


# Eigenvalues (E) and Eigenvectors (U)
E,U = eigh(H)
```

Ejercicio 4: Estados estacionarios del pozo finito ("*Jellium*")

- Comprobar que obtiene las soluciones correctas para un Oscilador Harmónico.
- Si la función eigh no normaliza los vectores, normalizarlos.
- Reemplazar el potencial $V(x)$ por un Jellium que contenga 2 estados ligados.
- Diagonalizar y verificar que las autofunciones y energías obtenidas son correctas.

In [ ]:
```

```