Derivatives (Symbolic & Numeric) (Mr. P Solver)

Video Link: https://youtu.be/DeeoiE22bZ8 (https://youtu.be/DeeoiE22bZ8)

Codes: https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEp2TkRlUzI0V2dpUU5EbkZlUXdQQXFTa05LQXxBQ3 (https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEp2TkRlUzI0V2dpUU5EbkZlUXdQQXFTa05LQXxBQ3

In [1]:
```
import numpy as np
import scipy as sp
import sympy as smp
import matplotlib.pyplot as plt
```

In [2]:
```
from scipy.misc import derivative
```

Symbolic Case

Here the function is known and we want to calculate the derivative using Sympy.

Example:

$$f(x) = e^{-ax^2} \cdot \sin(bx) \cdot \ln(c\sin(x)/x)$$

In [3]:
```
x, a, b, c = smp.symbols('x a b c')
f1 = smp.exp(-a* x**2)* smp.sin(b*x)* smp.log(c*smp.sin(x)/x)
f1
```

Out[3]:
$$e^{-ax^2} \log\left(\frac{c\sin(x)}{x}\right) \sin(bx)$$

In [4]:
```
f1p = smp.diff(f1,x)
f1p
```

Out[4]:
$$-2axe^{-ax^2} \log\left(\frac{c\sin(x)}{x}\right) \sin(bx) + be^{-ax^2} \log\left(\frac{c\sin(x)}{x}\right) \cos(bx) + \frac{x\left(\frac{c\cos(x)}{x} - \frac{c\sin(x)}{x^2}\right)e^{-}}{c\sin(x)}$$

In [5]:
```
f1pp = smp.diff(f1,x,2)  # 2nd order derivative
f1pp
```

Out[5]:

$$\left(-4abx\log\left(\frac{c\sin(x)}{x}\right)\cos(bx) - \frac{4ax\left(\cos(x) - \frac{\sin(x)}{x}\right)\sin(bx)}{\sin(x)} + 2a\left(2ax^2 - 1\right)\log\left(\frac{c\sin(\right.\right.$$

$$\left.\left. + \frac{2b\left(\cos(x) - \frac{\sin(x)}{x}\right)\cos(bx)}{\sin(x)} - \frac{\left(\frac{\left(\cos(x) - \frac{\sin(x)}{x}\right)\cos(x)}{\sin(x)} + \sin(x) - \frac{\cos(x) - \frac{\sin(x)}{x}}{x} + \frac{2\cos(x)}{x} - \frac{2\,si}{}\right.\right.}{\sin(x)}$$

In [6]:
```
f1p.subs([(a,1),(b,2),(c,3),(x,1)])
```

Out[6]:

$$-\frac{2\log(3\sin(1))\sin(2)}{e} + \frac{2\log(3\sin(1))\cos(2)}{e} + \frac{(-3\sin(1) + 3\cos(1))\sin(2)}{3e\sin(1)}$$

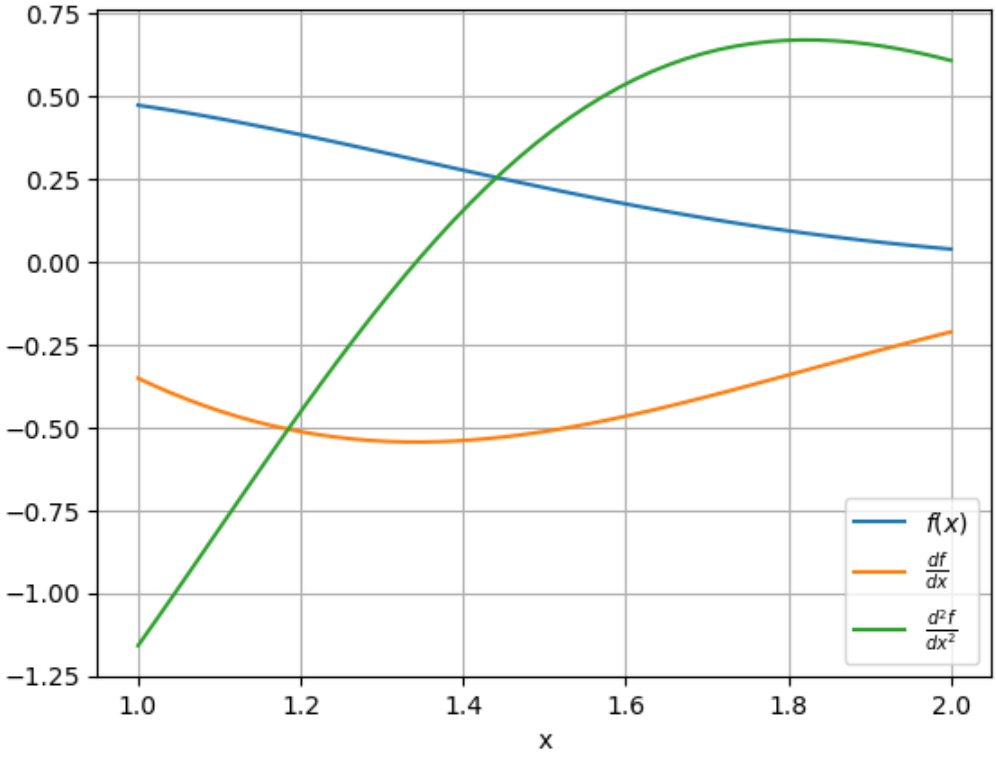
In [7]:
```
f1p.subs([(a,1),(b,2),(c,3),(x,1)]).evalf()
```

Out[7]:

$$-1.02277462733248$$

Plotting

In [8]:
```
f1f = smp.lambdify([x,a,b,c], f1)
f1pf = smp.lambdify([x,a,b,c], f1p)
f1ppf = smp.lambdify([x,a,b,c], f1pp)

x1 = np.linspace(1,2,50)
y1 = f1f(x1, a=0.5, b=1, c=3)
yp1 = f1pf(x1, a=0.5, b=1, c=3)
ypp1 = f1ppf(x1, a=0.5, b=1, c=3)

plt.plot(x1,y1,label='$f(x)$')
plt.plot(x1,yp1,label=r'$\frac{df}{dx}$')
plt.plot(x1,ypp1,label=r'$\frac{d^2f}{dx^2}$')
plt.xlabel('x')
plt.legend()
plt.grid()
plt.show()
```

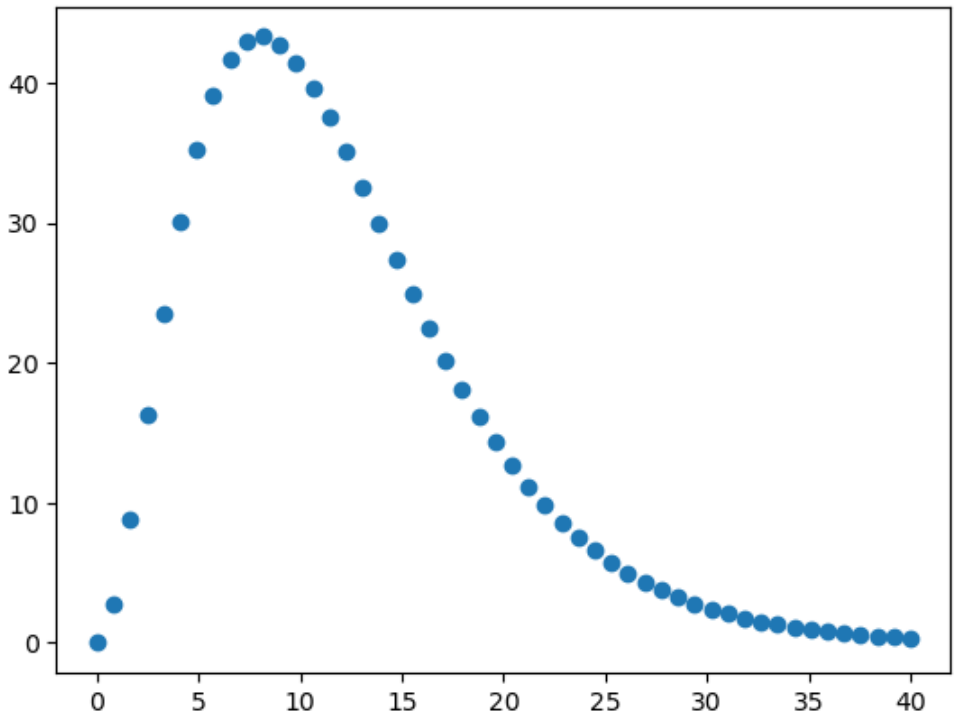


In [ ]:

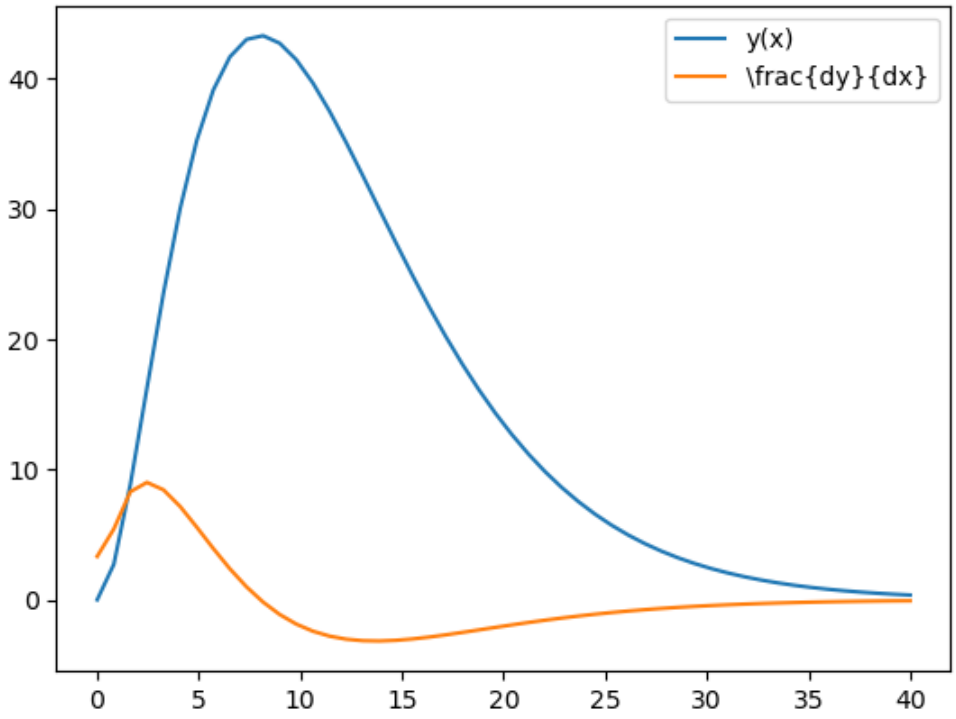
Numerical Case

In [9]:
```
# creating data
xdata = np.linspace(0,40,50)
ydata = 5*xdata**2*np.exp(-0.25*xdata)
plt.scatter(xdata,ydata)
```

Out[9]: <matplotlib.collections.PathCollection at 0x1558ba3d850>



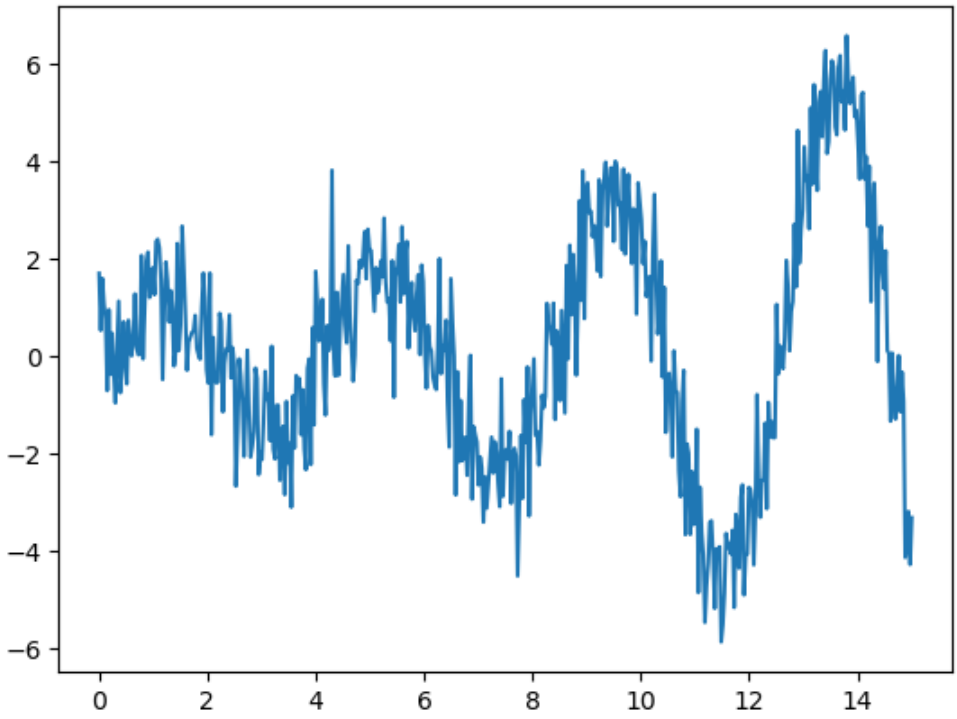
In [10]:
```
dydx = np.gradient(ydata,xdata)

plt.plot(xdata, ydata, label='y(x)')
plt.plot(xdata,dydx, label=r'\frac{dy}{dx}')
plt.legend()
plt.show()
```



`np.gradient` works good if the data is smooth but not if the data is noisy. There, before using `np.gradient`, we makes a smooth curve by the following method.

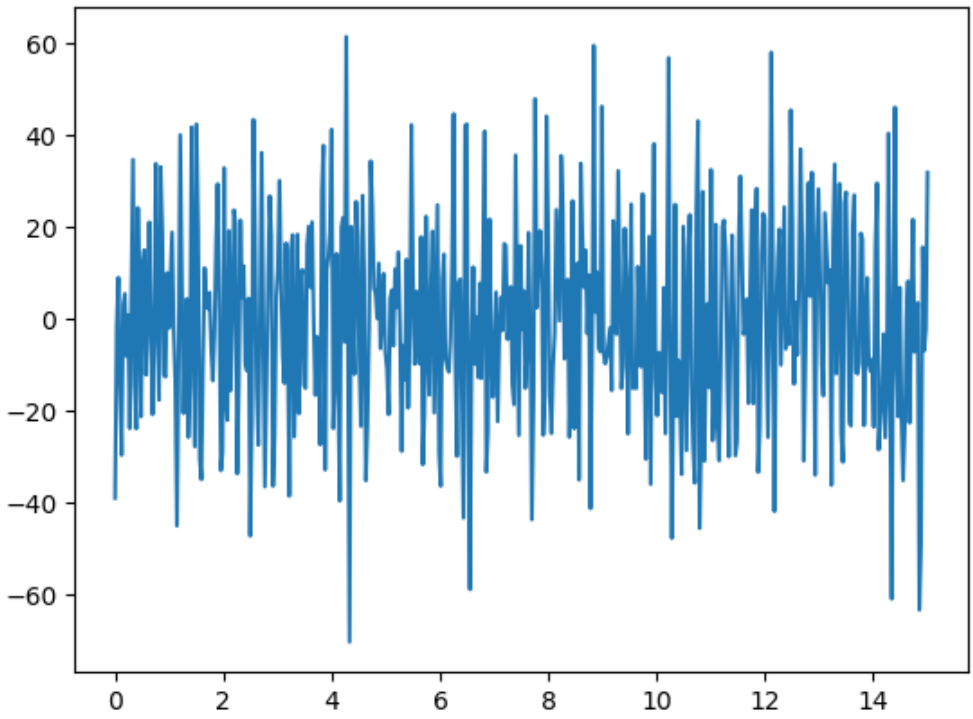
In [11]:
```
xdata = np.linspace(0.001,15,500)
ydata = np.exp(xdata/8)*np.sin(1.5*xdata) +0.9*np.random.randn(len(xdata))
plt.plot(xdata,ydata)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x1558baceaf0>]
```



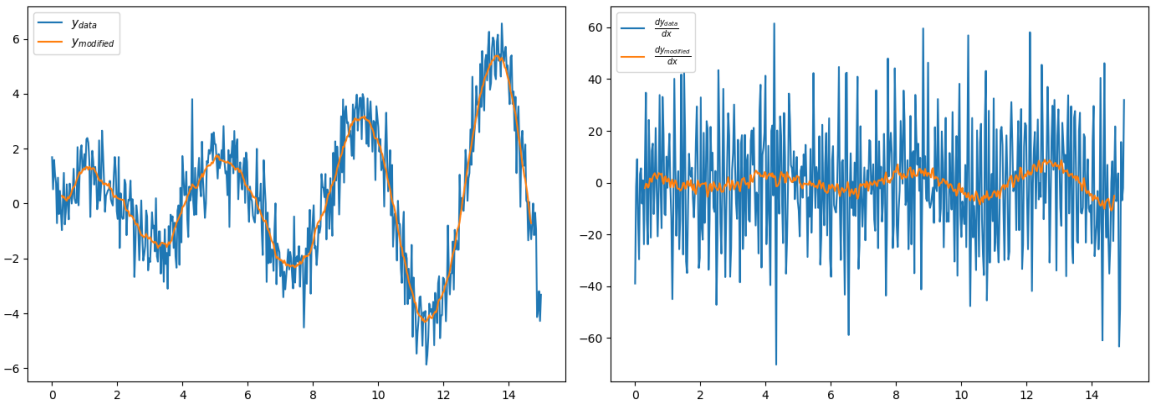
In [12]:
```
dydx = np.gradient(ydata,xdata)
plt.plot(xdata,dydx)
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x1558bb39730>]
```



We can get a smooth graph by using `np.convolve` .

In [13]:
```
filt = np.ones(21)/21
y1 = np.convolve(ydata,filt,mode='valid')
x1 = xdata[10:-10]  # making the length equal to len(y1)
dy1dx = np.gradient(y1,x1)

fig, ax = plt.subplots(1,2, figsize = (14,5))
ax1 = ax[0]
ax1.plot(xdata,ydata, label='$y_{data}$')
ax1.plot(x1,y1, label='$y_{modified}$')
ax2 = ax[1]
ax2.plot(xdata,dydx, label=r'$\frac{dy_{data}}{dx}$')
ax2.plot(x1,dy1dx, label=r'$\frac{dy_{modified}}{dx}$')
[i.legend() for i in ax]
fig.tight_layout()
plt.show()
```



In [ ]:

Have your own data and watch this section.

Quasi-Symbolic Case

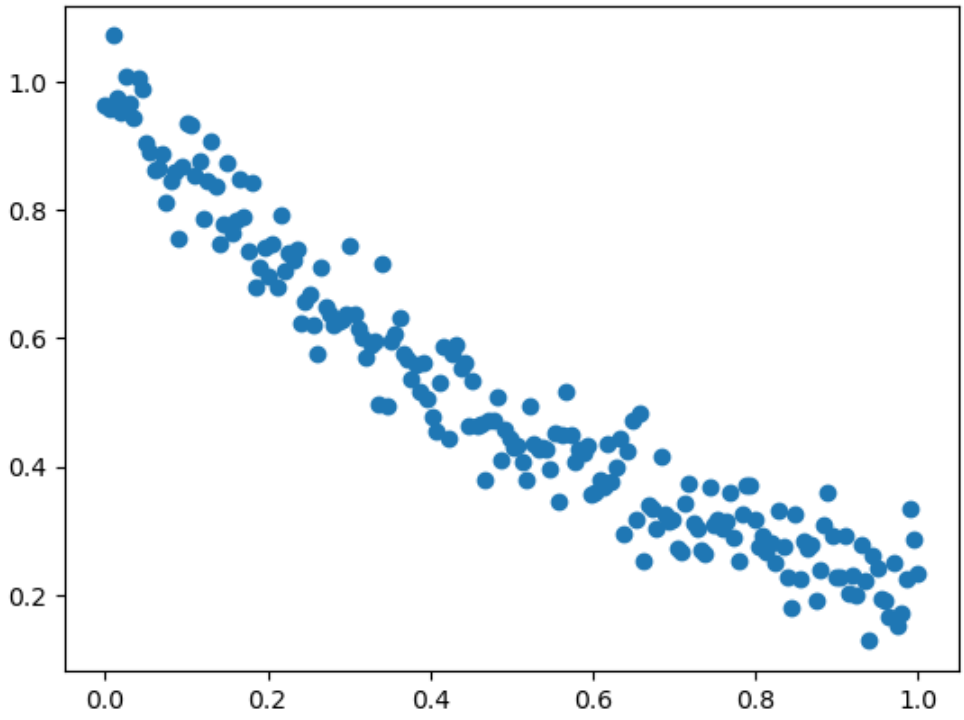
We don't know the function but the function is given by a typical expression.

Example:

$$f(u) = \max\left\{\left|e^{-x_i u^2} - y_i\right|\right\}$$

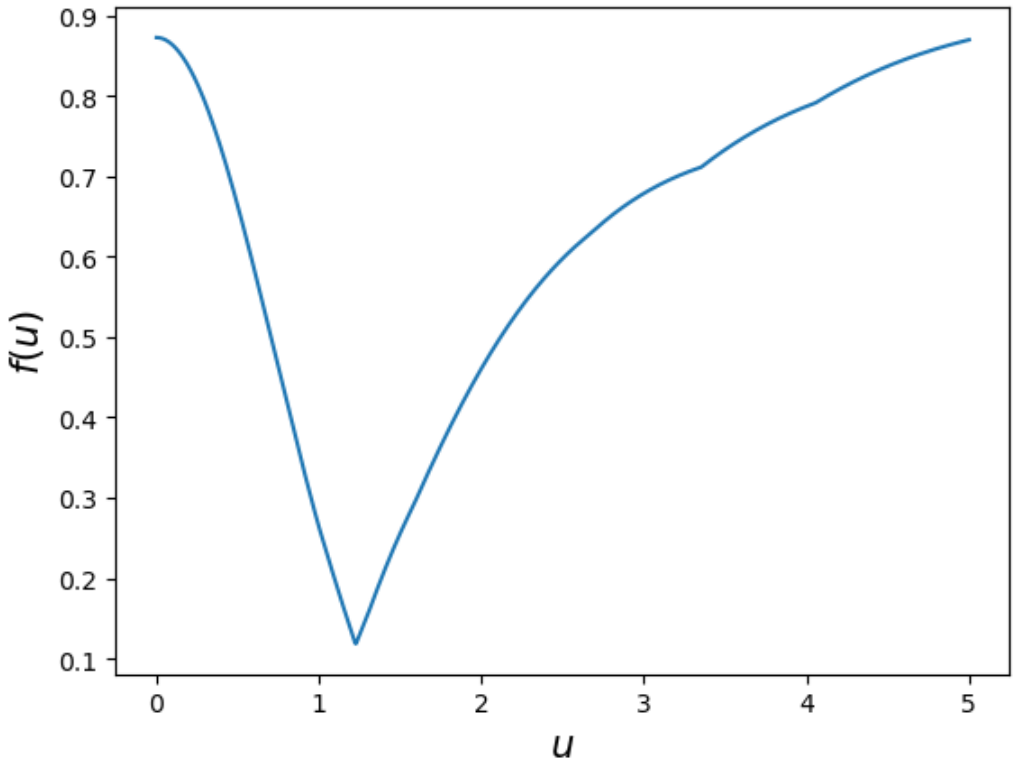
In [14]:
```
# creating data
x = np.linspace(0,1,200)
y = np.exp(-1.5*x) + 0.05*np.random.randn(len(x))
plt.scatter(x,y)
```

Out[14]: <matplotlib.collections.PathCollection at 0x1558b756eb0>



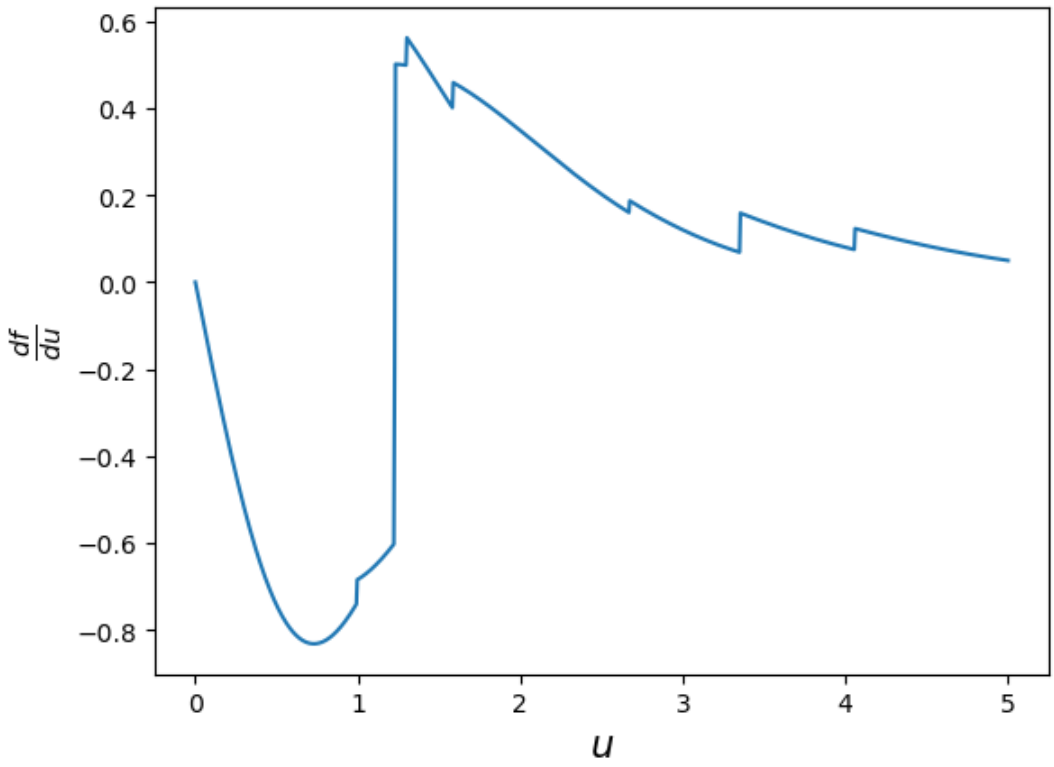
Defining the Function and plotting

In [15]:
```
def f(u):
    return max(abs(np.exp(-x*u**2)-y))

u = np.linspace(0,5,1000)
fu = np.vectorize(f)(u)

plt.plot(u,fu)
plt.xlabel('$u$', fontsize=15)
plt.ylabel('$f(u)$', fontsize=15)
plt.show()
```



Derivative of the function and plotting that.

In [16]:
```
dfdu = np.vectorize(derivative)(f,u,dx=1e-6)
plt.plot(u,dfdu)
plt.xlabel('$u$', fontsize=15)
plt.ylabel(r'$\frac{df}{du}$', fontsize=15)
plt.show()
```



In [ ]:

In [ ]: