Physics 3800 - Gaussian Integration

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
sp.init_printing(use_latex='mathjax')
from ipywidgets import interactive, fixed

from scipy.special import *
import scipy.integrate as integrate
```

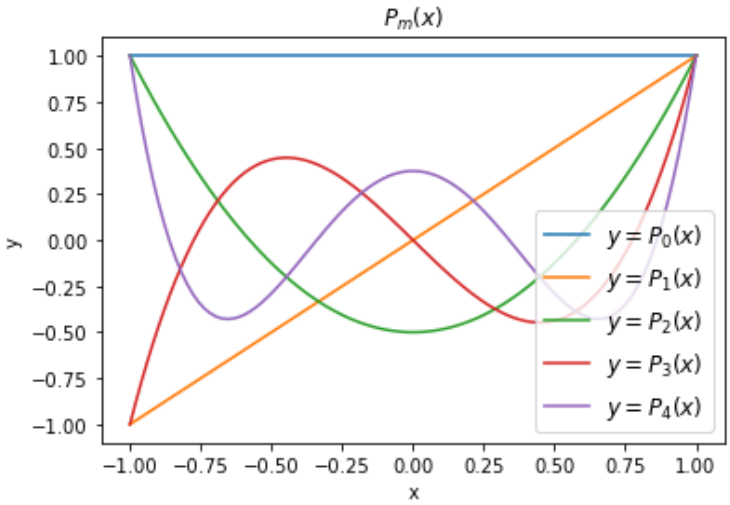
Legendre Polynomial Coefficients

In [2]:
```
legendre(4)
```

Out[2]:
```
poly1d([ 4.37500000e+00,  4.85722573e-16, -3.75000000e+00,  2.42861287e-16,
         3.75000000e-01])
```

Graphs of Legendre Polynomials

In [3]:
```
x = np.linspace(-1,1,100)

plt.title(f'$P_m(x)$')

for m in range(5):
    y = eval_legendre(m, x)
    plt.plot(x,y, label=f'$y=P_{m}(x)$')

plt.legend(fontsize=12)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Roots

In [4]:
```
roots, weights = roots_legendre(4)
roots
```

Out[4]:
```
array([-0.86113631, -0.33998104,  0.33998104,  0.86113631])
```

In [5]:
```
eval_legendre(4,roots)
```

Out[5]:
```
array([ 0.00000000e+00,  5.55111512e-17,  5.55111512e-17, -5.55111512e-1
7])
```

Construct matrix

Each row n ($n = 0$ to 3) is $P_n(x)$ evaluated at each of the four roots of $P_4(x)$. Also need inverse of P, P^{-1}, and the weights, which are the first row of $2P^{-1}$. We compare this with what the *scipy* function *roots_legendre* reports as the weights for Gauss-Legendre integration.

In [6]:
```
P = np.eye(4,4)

for n in range(4):
    P[n]=eval_legendre(n,roots)

P = np.transpose(P)
P
```

Out[6]:
```
array([[ 1.        , -0.86113631,  0.61233362, -0.30474698],
       [ 1.        , -0.33998104, -0.32661934,  0.411728  ],
       [ 1.        ,  0.33998104, -0.32661934, -0.411728  ],
       [ 1.        ,  0.86113631,  0.61233362,  0.30474698]])
```

In [7]:
```
Pinv = np.linalg.inv(P)
Pinv
```

Out[7]:
```
array([[ 0.17392742,  0.32607258,  0.32607258,  0.17392742],
       [-0.44932566, -0.33257549,  0.33257549,  0.44932566],
       [ 0.53250804, -0.53250804, -0.53250804,  0.53250804],
       [-0.371027  ,  0.93977247, -0.93977247,  0.371027  ]])
```

In [8]:
```
w = 2*Pinv[0]
w
```

Out[8]:
```
array([0.34785485, 0.65214515, 0.65214515, 0.34785485])
```

In [9]:
```
weights
```

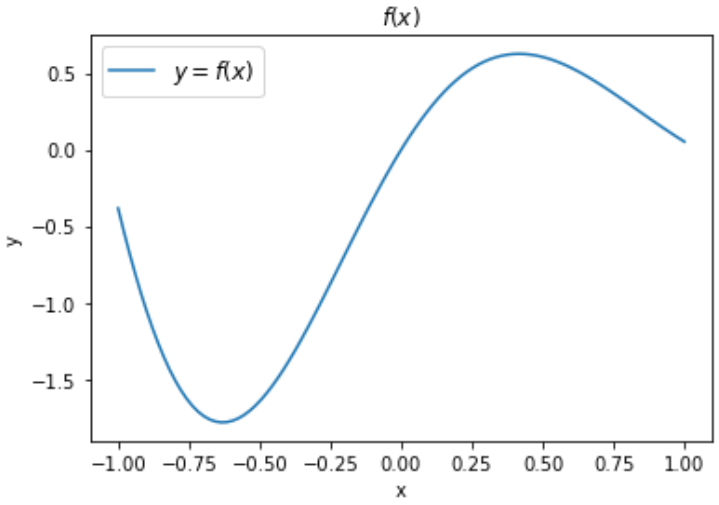
Out[9]:
```
array([0.34785485, 0.65214515, 0.65214515, 0.34785485])
```

Integrate a function

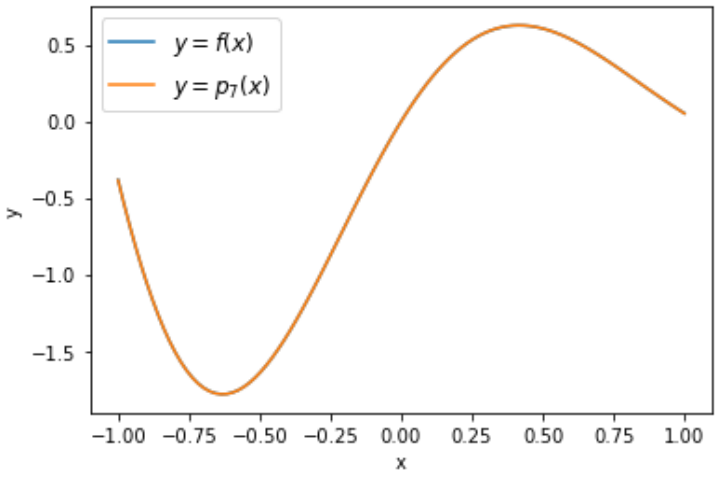
A function that can be well fit with a polynomial of order $2n - 1 = 2(4) - 1 = 7$. Use library function to calculate integral and compare with Gauss-Legendre method.

In [10]:
```
def f(x):
    return np.exp(-x)*np.sin(3*x)
```

In [11]:
```
x = np.linspace(-1,1,100)

plt.title(f'$f(x)$')

y = f(x)
plt.plot(x,y, label=f'$y=f(x)$')

plt.legend(fontsize=12)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



In [12]:
```
p7=np.polyfit(x,y,7)
ypoly7=np.polyval(p7,x)

plt.plot(x,y, label=f'$y=f(x)$')
plt.plot(x,ypoly7, label=f'$y=p_7(x)$')


plt.legend(fontsize=12)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



In [13]:
```
val,error = integrate.quad(f,-1,1)
print(val, error)
```

```
-0.7416161285463809 1.708125114500024e-14
```

In [14]:
```
scipy_GaussLegendre, tmp = integrate.fixed_quad(f,-1,1,n=4)
print(scipy_GaussLegendre)
```

```
-0.7431706299286471
```

In [15]:
```
gauss_fxi = f(roots)
our_GaussLegendre=np.dot(gauss_fxi,weights)
print(our_GaussLegendre)
```

```
-0.7431706299286476
```

In [16]:
```
Percent_error = abs((val-our_GaussLegendre)/val)*100
print(Percent_error)
```

```
0.20960997508422305
```

Only 0.2% error with 4 points!