Homework Problem 21 - 1D wavefunction expansions

Problem 21 - 20 pts (5 pts per part)

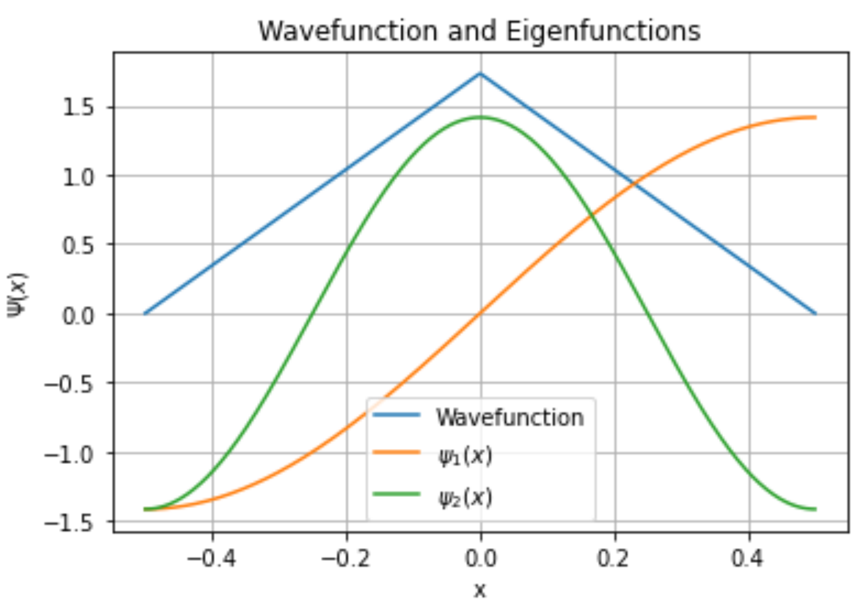
In [1]:
```
# Load the standard set of libraries for basic numerical compu
import numpy as np
%matplotlib inline
# I chose inline because it is more straightforward to print a
import matplotlib
import matplotlib.pyplot as plt
# Don't forget that it is ok to grab code for plotting and inte
# (the tutorial is useful).
# Useful commands include numpy.sum(f*dx) and numpy.absolute(x,
# Good Python/Numpy reference:
```

The spatial part of a wavefunction for a particle in a square well with infinite walls is $\Psi(x) = \sqrt{12/a^2}(a/2 - |x|)$ for $a/2 < x < a/2$ and it is zero elsewhere. It can be written as the sum of the eigenfunctions of the particle in a box problem: $\Psi(x,t) = c_1\psi_1 + c_2\psi_2 + \ldots$.

$\psi_n = \sqrt{(2/a)}sin(k_n x)$ for n=even.and $\psi_n = \sqrt{(2/a)}cos(k_n x)$ for n=odd where $k_n = n\pi/a$.

A) Plot the wavefunction $\Psi(x)$ and the eigenfunctions ψ_1 and ψ_2. Make an argument for why some of the c's are zero, and specify which ones are zero.

In [22]:
```
def wavefunction(x, a):
    return np.sqrt(12 / a**2) * (a / 2 - np.abs(x))

def eigenfunction1(x, a):
    return np.sqrt(2 / a) * np.sin((np.pi * x / a))

def eigenfunction2(x, a):
    return np.sqrt(2 / a) * np.cos((2 * np.pi * x / a))

a = 1
x_values = np.linspace(-a / 2, a / 2, 400)

y_wavefunction = wavefunction(x_values, a)
y_eigenfunction1 = eigenfunction1(x_values, a)
y_eigenfunction2 = eigenfunction2(x_values, a)

plt.plot(x_values, y_wavefunction, label='Wavefunction')
plt.plot(x_values, y_eigenfunction1, label=r'$\psi_1(x)$')
plt.plot(x_values, y_eigenfunction2, label=r'$\psi_2(x)$')

plt.xlabel('x')
plt.ylabel(r'$\Psi(x)$')
plt.title('Wavefunction and Eigenfunctions')

plt.grid(True)
plt.legend()
plt.show()
```



The Coefficents for the odd eigenfunctions will be 0 because the wavefunction is an even function.

B) Calculate the coefficients for c_1 and c_2 by numerical integration of the product of wavefunctions. Use a small step-size (e.g.- 0.001 a). Are the coefficients you calculated reasonably consistent with your expectations from your argument above?

In [45]:
```
def integrand(x, a, is_sin=True):
    if is_sin:
        return np.sqrt(12 / a**2) * (a / 2 - np.abs(x)) * np.sc
    else:
        return np.sqrt(12 / a**2) * (a / 2 - np.abs(x)) * np.sc

def trapezoidal_rule(func, a, n, is_sin=True):
    # Calculate the width of each subinterval
    h = a / n

    # Initialize the integral value
    integral = 0.5 * (func(-a / 2, a, is_sin) + func(a / 2, a,

    # Sum the function values at the endpoints of each subinter
    for i in range(1, n):
        integral += func(-a / 2 + i * h, a, is_sin)

    # Multiply by the width of each subinterval
    integral *= h

    return integral

def solve_integral(a, n):
    # Use the trapezoidal rule to compute the integrals for bo
    integral_sin = trapezoidal_rule(integrand, a, n, is_sin=Tru
    integral_cos = trapezoidal_rule(integrand, a, n, is_sin=Fal
    return integral_sin, integral_cos
a = 1  # Value of well width
n = 100000  # Number of intervals for the trapezoidal rule
result_sin, result_cos = solve_integral(a, n)
print("Approximate value of c1:", result_sin)
print("Approximate value of c2:", result_cos)
```

```
Approximate value of c1: -9.264231390490249e-17
Approximate value of c2: 0.49637040028041307
```

These are close to what I expected. c1, the odd eigenfunction coefficent, is close to 0, and c2, the first even eigenfunction coefficent is not 0.

C) Calculate the coefficients for the first three non-zero coefficients by numerically integrating the product of wavefunctions.

In [57]:

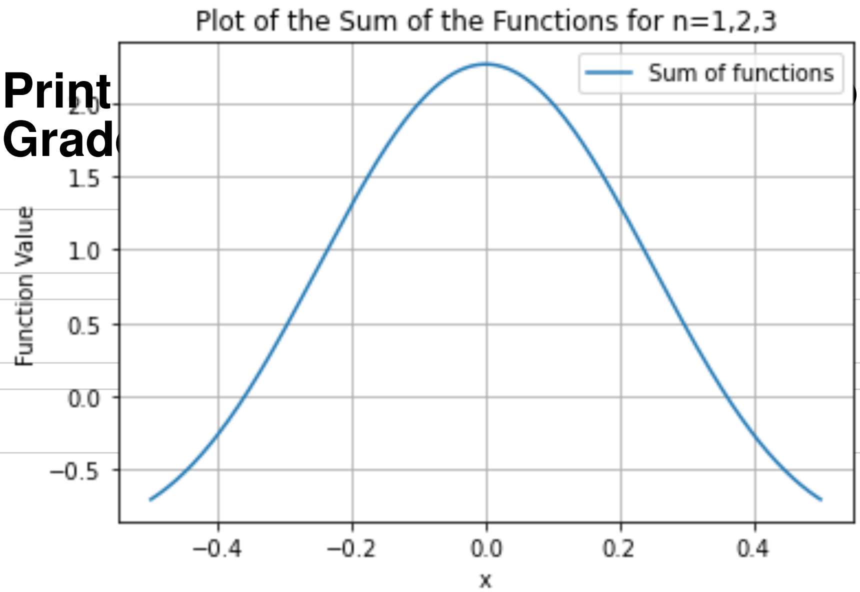
```
def integrand(x, a, n):

    return np.sqrt(12 / a**2) * (a / 2 - np.abs(x)) * np.sqrt(2

def simpsons_rule(func, a, n):
    # Define the number of intervals for Simpson's rule
    num_intervals = 1000  # You can adjust this value as needed

    # Calculate the width of each subinterval
    h = a / num_intervals

    # Initialize the integral value
    integral = func(-a / 2, a, n) + func(a / 2, a, n)

    # Sum the function values at the endpoints of each subinter
    for i in range(1, num_intervals):
        x = -a / 2 + i * h
        if i % 2 == 0:
            integral += 2 * func(x, a, n)
        else:
            integral += 4 * func(x, a, n)

    # Multiply by the width of each subinterval and divide by 3
    integral *= h / 3

    return integral

def solve_integral(a, n):

    # Use Simpson's rule to compute the integral
    integral = simpsons_rule(integrand, a, n)
    return integral

# Example usage:
a = 1  # Value of parameter 'a'
for n in [1, 2, 3]:
    result = solve_integral(a, n)
    print(f"Approximate value of the coefficent for n={n*2}: {r
```

```
Approximate value of the coefficent for n=2: 0.99274080023261
82
Approximate value of the coefficent for n=4: 0.49637040010422
12
Approximate value of the coefficent for n=6: 0.11030453334485
336
```

D) Plot the sum of the first three non-zero terms times their eigenfunctions.

In [59]:
```
def combined_function(x, a, coefficients):
    combined_values = np.zeros_like(x)
    for n, coefficient in enumerate(coefficients, start=1):
        combined_values += coefficient * np.sqrt(2 / a) * np.co
    return combined_values

# Define parameters
a = 1 # Value of parameter 'a'
coefficients = [0.9927408002342233, 0.4963704001171262, 0.11030

# Generate x values
x_values = np.linspace(-a / 2, a / 2, 400)

# Calculate the combined function values
combined_values = combined_function(x_values, a, coefficients)

# Plot the combined function
plt.plot(x_values, combined_values, label='Sum of functions')

# Add labels and title
plt.xlabel('x')
plt.ylabel('Function Value')
plt.title('Plot of the Sum of Functions for n=1,2,3')

# Add grid
plt.grid(True)

# Add legend
plt.legend()

# Show
```

Out[59]: <matplotlib.legend.Legend at 0x7f6f22689d60>



Print... mit it in Grad...

In [ ]:

In [ ]:

In [ ]: