Laboratorium
Metod Numerycznych
Całkowanie numeryczne ¶

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy.polynomial.legendre import leggauss
from scipy.integrate import quad
import scipy


from typing import Union, List, Tuple

def rectangular_rule(func, a, b, n):
    h = (b - a) / n  # Szerokość podprzedziału
    result = 0

    for i in range(n):
        x_i = a + i * h  # Lewy brzeg podprzedziału
        result += h * func(x_i)

    return result


def trapezoidal_rule(func, a, b, n):
    h = (b - a) / n  # Szerokość podprzedziału
    result = 0

    for i in range(n):
        x_i = a + i * h  # Lewy brzeg podprzedziału
        x_next = a + (i - 1) * h  # Prawy brzeg podprzedziału
        result += 0.5 * h * (func(x_i) + func(x_next))

    return result



def custom_integration(func, a, b, order):
    """
    Własna funkcja całkująca, wykorzystująca kwadraturę Gaussa-Legendre'a.

    :param func: Funkcja do zintegrowania.
    :param a: Dolna granica całkowania.
    :param b: Górna granica całkowania.
    :param order: Rząd kwadratury.
    :return: Przybliżona wartość całki.
    """
    # Przeskalowanie przedziału do (-1, 1)
    scaled_func = lambda x: func(0.5 * (b - a) * x + 0.5 * (b + a))

    # Uzyskanie węzłów i wag z kwadratury Gaussa-Legendre'a
    nodes, weights = leggauss(order)

    # Obliczenie wartości funkcji w przeskalowanych punktach
    values = scaled_func(nodes)

    # Obliczenie całki przy użyciu kwadratury Gaussa-Legendre'a
    result = np.dot(weights, values) * 0.5 * (b - a)

    return result

def error_vs_neval(func, a, b, true_value, max_order=20):
    """
    Funkcja obliczająca błąd całkowania w zależności od liczby wywołań funkc
```

```
    :param func: Funkcja do zintegrowania.
    :param a: Dolna granica całkowania.
    :param b: Górna granica całkowania.
    :param true_value: Dokładna wartość całki do porównania.
    :param max_order: Maksymalny rząd kwadratury do rozważenia.
    :return: Listy z błędami względnymi i bezwzględnymi oraz liczbą wywołań
    """
    relative_errors = []
    absolute_errors = []
    neval_values = []

    for order in range(1, max_order + 1):
        result_gaussian = custom_integration(func, a, b, order)

        # Obliczenie błędów
        absolute_error = np.abs(result_gaussian - true_value)
        relative_error = absolute_error / np.abs(true_value)

        # Zapisanie błędów i liczby wywołań funkcji podcałkowej
        absolute_errors.append(absolute_error)
        relative_errors.append(relative_error)
        neval_values.append(order)

    return relative_errors, absolute_errors, neval_values
```

Całkowanie numeryczne to dziedzina matematyki numerycznej zajmująca się przybliżonym obliczaniem wartości całek matematycznych, zwłaszcza tych, które nie mają rozwiązania analitycznego lub gdy obliczenia analityczne są trudne lub niemożliwe do przeprowadzenia. Całkowanie numeryczne jest szeroko stosowane w praktyce, szczególnie w obszarach takich jak nauki przyrodnicze, inżynieria, ekonometria, czy analiza danych.

Podstawowe metody całkowania numerycznego obejmują:

Metoda prostokątów

Metoda prostokątów: Polega na przybliżeniu całki poprzez sumowanie wartości funkcji w punktach lewego brzegu podprzedziałów i pomnożenie tego przez szerokość podprzedziału.

$\int_{x_0}^{x_n} f(x)dx \approx h \sum_{i=0}^{n-1} f(x_i)$

gdzie: $h = \frac{x_n - x_0}{n}$

Metoda trapezów

Metoda trapezów: Podobnie jak metoda prostokątów, ale używa trapezów zamiast prostokątów. Jest bardziej dokładna dla funkcji, które nie są zupełnie równoległe osi x.

$\int_{x_0}^{x_n} f(x)dx \approx \sum_{i=0}^{n-1} \frac{h}{2}(f(x_{i+1}) + f(x_i))$

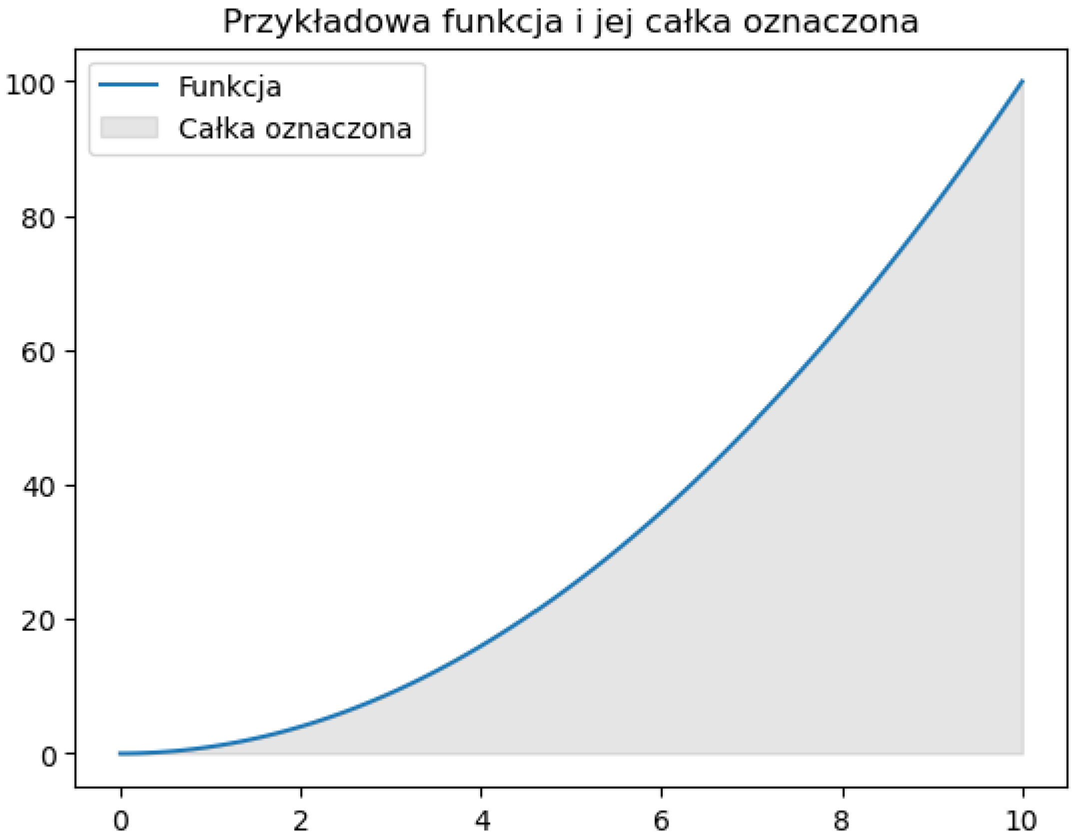
Zadanie 1.

Zaimplementuj metodę prostokątów oraz metodę trapezów zgodnie z opisem funkcji u góry.

In [14]:

```
def przyklad(x):
    return x**2  # Przykładowa funkcja kwadratowa

# Dolna i górna granica całkowania
a = 0
b = 10

# Liczba podprzedziałów
n = 1000

# Metoda prostokątów
wynik_prostokatow = rectangular_rule(przyklad, a, b, n)
print("Metoda prostokątów:", wynik_prostokatow)

# Metoda trapezów
wynik_trapezow = trapezoidal_rule(przyklad, a, b, n)
print("Metoda trapezów:", wynik_trapezow)

# Do porównania, użyjmy scipy.integrate.quad, która oblicza dokładną wartość
exact_result, _ = quad(przyklad, a, b)
print("Dokładna wartość:", exact_result)

# Wykres funkcji
x_values = np.linspace(a, b, 100)
y_values = przyklad(x_values)

plt.plot(x_values, y_values, label='Funkcja')
plt.fill_between(x_values, y_values, alpha=0.2, color='gray', label='Całka o

plt.title('Przykładowa funkcja i jej całka oznaczona')
plt.legend()
plt.show()
```

```
Metoda prostokątów: 332.8334999999999
Metoda trapezów: 332.33450000000005
Dokładna wartość: 333.33333333333326
```



Zadanie 2.

Wyznacz numerycznie wartość całki:

$$\int_{a}^{b} \log(1 + \tan(x))dx$$

Wykonaj następujące czynności:

1. Czy możliwe jest całkowanie podanej funkcji na dowolnym przedziale? Swoją odpowiedź uzasadnij.
2. Oblicz całkę numerycznie na przedziale $a = 0$, $b = \frac{\pi}{4}$ przy pomocy następujących metod zaimplementowanych w zadaniu 1

In [15]:
```
# Podana funkcja
def log_tan(x):
    return np.log(1 + np.tan(x))

# Dolna i górna granica całkowania
a = 0
b = np.pi / 4

# Liczba podprzedziałów
n = 1000

# Metoda prostokątów
wynik_prostokatow = rectangular_rule(log_tan, a, b, n)
print("Metoda prostokątów:", wynik_prostokatow)

# Metoda trapezów
wynik_trapezow = trapezoidal_rule(log_tan, a, b, n)
print("Metoda trapezów:", wynik_trapezow)
```

```
Metoda prostokątów: 0.2719260630266622
Metoda trapezów: 0.2716538645231376
```

Odpowiedzi:

1. Funkcja podcałkowa to log(1+tan(x)). Całkowanie tej funkcji na dowolnym przedziale może być możliwe, ale istnieje pewien warunek, który musi być spełniony. Funkcja tan(x) ma miejsca, gdzie staje się nieskończona, a dokładniej, tam gdzie x przyjmuje wartości (2k + 1) · π/2 , gdzie k jest liczbą całkowitą. W tych punktach tan(x) staje się nieskończona, co prowadzi do log(1+tan(x)) przyjmujące wartość nieskończoną.

Zadanie 3.

Dla funkcji

$$f(x) = e^{x^2}$$

wyznacz numerycznie wartośc całki na przedziale $a = 0, b = 1$ w taki sam sposób jak w zadaniu 2.

In [16]:
```
# Nowa funkcja
def exp_square(x):
    return np.exp(x**2)

# Dolna i górna granica całkowania
a = 0
b = 1

# Liczba podprzedziałów
n = 1000

# Metoda prostokątów
wynik_prostokatow = rectangular_rule(exp_square, a, b, n)
print("Metoda prostokątów:", wynik_prostokatow)

# Metoda trapezów
wynik_trapezow = trapezoidal_rule(exp_square, a, b, n)
print("Metoda trapezów:", wynik_trapezow)
```

```
Metoda prostokątów: 1.461793058039848
Metoda trapezów: 1.4609366318345505
```

Zadanie 4.

Dane są funkcje:

$$f(x) = e^{-x^2}$$

$$g(x) = \frac{1}{x^2 + 4}$$

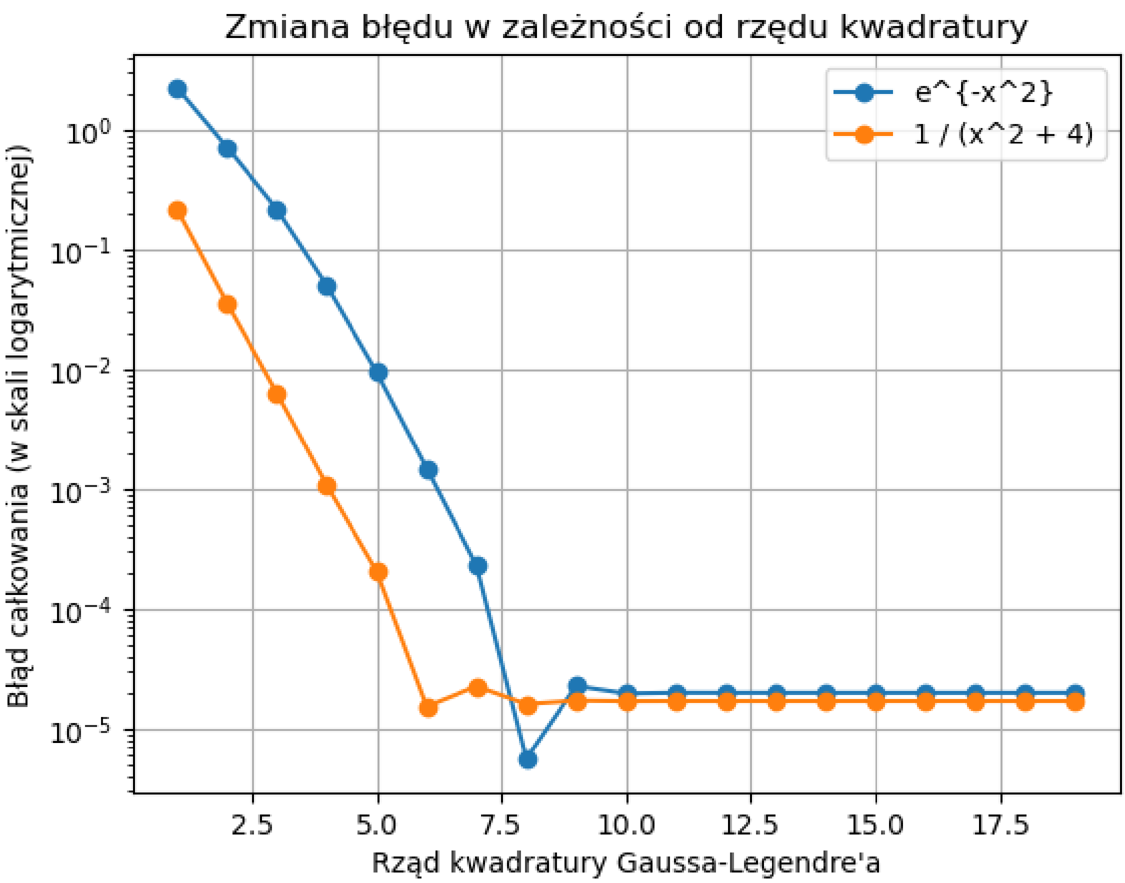
Napisz funkcję całkującą *custom_integration*, która wykorzystuje kwadraturę Gaussa-Legendre'a zaimplementowaną w bibliotece numpy.polynomial.legendre. Funkcja powinna przyjmować funkcję do zintegrowania, przedział całkowania oraz rząd kwadratury. Przedział ten zostanie przeskalowany do standardowego przedziału (-1, 1) używając wag i węzłów uzyskanych z funkcji leggauss. Następnie funkcja powinna obliczyć wartość całki przy użyciu kwadratury Gaussa-Legendre'a.

W celu oceny dokładności metody, należy porównać wyniki uzyskane za pomocą tej funkcji z wynikami uzyskanymi za pomocą metody trapezów (np.trapz) na wybranych funkcjach na różnych przedziałach. Równocześnie, należy analizować, jak błąd całkowania zmienia się wraz ze wzrostem rzędu kwadratury. Do tego celu należy użyć funkcji semilogy z biblioteki matplotlib w celu stworzenia wykresu w skali logarytmicznej.

In [17]:
```
# Funkcje do testowania
func_e = lambda x: np.exp(-x**2)
func_g = lambda x: 1 / (x**2 + 4)

# Przedział całkowania
a = -2
b = 2

# Rząd kwadratury
order = 5

# Testujemy funkcję custom_integration dla funkcji e^{-x^2}
result_gaussian_e = custom_integration(func_e, a, b, order)
print("Wynik Gaussa-Legendre'a dla e^{-x^2}:", result_gaussian_e)

# Porównujemy wyniki z metodą trapezów dla funkcji e^{-x^2}
x_values_e = np.linspace(a, b, 100)
y_values_e = func_e(x_values_e)
result_trapezoidal_e = np.trapz(y_values_e, x_values_e)
print("Wynik metody trapezów dla e^{-x^2}:", result_trapezoidal_e)

# Tworzymy wykres zmiany błędu w zależności od rzędu kwadratury dla e^{-x^2}
order_range_e = range(1, 20)
errors_e = []

for order_e in order_range_e:
    result_gaussian_e = custom_integration(func_e, a, b, order_e)
    error_e = np.abs(result_gaussian_e - result_trapezoidal_e)
    errors_e.append(error_e)

# Tworzymy wykres w skali logarytmicznej dla e^{-x^2}
plt.semilogy(order_range_e, errors_e, marker='o', label='e^{-x^2}')

# Testujemy funkcję custom_integration dla funkcji 1 / (x^2 + 4)
result_gaussian_g = custom_integration(func_g, a, b, order)
print("Wynik Gaussa-Legendre'a dla 1 / (x^2 + 4):", result_gaussian_g)

# Porównujemy wyniki z metodą trapezów dla funkcji 1 / (x^2 + 4)
x_values_g = np.linspace(a, b, 100)
y_values_g = func_g(x_values_g)
result_trapezoidal_g = np.trapz(y_values_g, x_values_g)
print("Wynik metody trapezów dla 1 / (x^2 + 4):", result_trapezoidal_g)

# Tworzymy wykres zmiany błędu w zależności od rzędu kwadratury dla 1 / (x^2
order_range_g = range(1, 20)
errors_g = []

for order_g in order_range_g:
    result_gaussian_g = custom_integration(func_g, a, b, order_g)
    error_g = np.abs(result_gaussian_g - result_trapezoidal_g)
    errors_g.append(error_g)

# Tworzymy wykres w skali logarytmicznej dla 1 / (x^2 + 4)
plt.semilogy(order_range_g, errors_g, marker='o', label='1 / (x^2 + 4)')

plt.xlabel('Rząd kwadratury Gaussa-Legendre\'a')
plt.ylabel('Błąd całkowania (w skali logarytmicznej)')
plt.title('Zmiana błędu w zależności od rzędu kwadratury')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

```
Wynik Gaussa-Legendre'a dla e^{-x^2}: 1.7735688756358607
Wynik metody trapezów dla e^{-x^2}: 1.7641428535931651
Wynik Gaussa-Legendre'a dla 1 / (x^2 + 4): 0.7855855855855857
Wynik metody trapezów dla 1 / (x^2 + 4): 0.7853811583299719
```



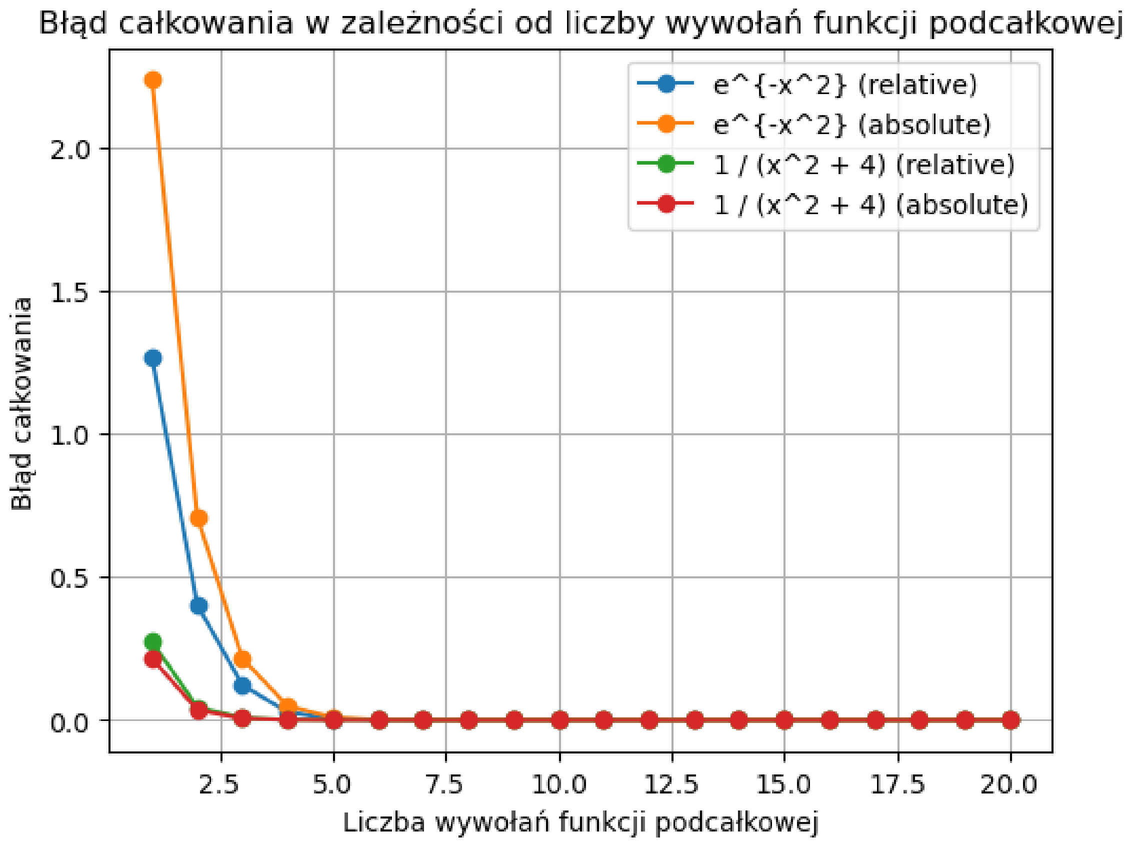
Zadanie 5

Wyznacz wartość całki wykorzystując funkcje z *zadania 4* oraz całkowanie adaptacyjne (jest to funkcja *scipy.integrate.quad*).

Zbadaj jak zmienia się błąd całkowania zmieniając błąd względny i bezwzględny w funkcji *quad*. Wybierz 10 różnych wartości.

Utwórz wykres błędu całki w zależności od liczby wywołań funkcji podcałkowej. Liczba wywołań jest w argumencie wyjściowym infodict jako 'neval'.

In [18]:
```
# Testujemy funkcję error_vs_neval dla funkcji e^{-x^2}
true_value_e, _ = quad(func_e, a, b)
relative_errors_e, absolute_errors_e, neval_values_e = error_vs_neval(func_e

# Tworzymy wykres błędu całki w zależności od liczby wywołań funkcji podcałk
plt.plot(neval_values_e, relative_errors_e, marker='o', label='e^{-x^2} (rel
plt.plot(neval_values_e, absolute_errors_e, marker='o', label='e^{-x^2} (abs

# Testujemy funkcję error_vs_neval dla funkcji 1 / (x^2 + 4)
true_value_g, _ = quad(func_g, a, b)
relative_errors_g, absolute_errors_g, neval_values_g = error_vs_neval(func_g

# Tworzymy wykres błędu całki w zależności od liczby wywołań funkcji podcałk
plt.plot(neval_values_g, relative_errors_g, marker='o', label='1 / (x^2 + 4)
plt.plot(neval_values_g, absolute_errors_g, marker='o', label='1 / (x^2 + 4)

plt.xlabel('Liczba wywołań funkcji podcałkowej')
plt.ylabel('Błąd całkowania')
plt.title('Błąd całkowania w zależności od liczby wywołań funkcji podcałkowe
plt.legend()
plt.grid(True)
plt.show()
```



Materiały uzupełniające:

- [Scipy Lecture Notes (http://www.scipy-lectures.org/index.html)](http://www.scipy-lectures.org/index.html)
- [NumPy for Matlab users (https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html#numpy-for-matlab-users)](https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html#numpy-for-matlab-users)
- [Python Tutorial - W3Schools (https://www.w3schools.com/python/default.asp)](https://www.w3schools.com/python/default.asp)
- [NumPy (https://www.numpy.org)](https://www.numpy.org)
- [Matplotlib (https://matplotlib.org/)](https://matplotlib.org/)
- [Anaconda (https://www.anaconda.com/)](https://www.anaconda.com/)