Laboratorium 4
Metod Numerycznych

Instrukcja:
Na zajęciach należy wykonać poniższe zadania, dokonać testu na platformie github, a następnie sporządzić sprawozdanie zawierające odpowiedzi z komentarzami.

In [1]:
```python
import numpy as np
import scipy
import matplotlib
import matplotlib.pyplot as plt

from typing import Union, List, Tuple
```

Funkcje niezbędne do wykonania laboratorium:

Zadanie 1.

W celu wykonywania interpolacji należy przygotować funkcję wyliczającą wektor węzłów Czebyszewa (funkcja *chebyshev_nodes*) dany wzorem

$$x(k) = \cos\left(\frac{k\pi}{n}\right), \quad k = 0, 1, 2, \dots, n$$

Oraz wagi barycentryczne dla tego typu wezłów (funkcja def *bar_czeb_weights*):

$$w_j = (-1)^j \delta_j$$
$$\delta_j = \begin{cases} \frac{1}{2}, & j = 0 \text{ lub } j = n \\ 1, & j \in (0, n) \end{cases}$$

In [2]:
```
def chebyshev_nodes(n):
    if n <= 0:
        return None
    nodes = np.cos(np.pi*np.arange(n+1)/n)
    return nodes
def bar_czeb_weights(n):
    if n <= 0:
        return None

    delta_j = np.ones(n+1)
    delta_j[1:n:2] = -1  # Ustawienie wartości delta_j dla węzłów o nieparzy

    w_j = (-1) ** np.arange(n+1) * delta_j

    return w_j

n = 10
nodes = chebyshev_nodes(n)
weights = bar_czeb_weights(n)

print("Wezły czybyszewa:", nodes)
print("Wagi:", weights)
```

```
Wezły czybyszewa: [ 1.00000000e+00  9.51056516e-01  8.09016994e-01  5.8778
5252e-01
  3.09016994e-01  6.12323400e-17 -3.09016994e-01 -5.87785252e-01
 -8.09016994e-01 -9.51056516e-01 -1.00000000e+00]
Wagi: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Zadanie 2.
Do przeprowadzenia ćwiczenia należy zdefiniować następujące funkcje:

1. Funkcję ciągłą nieróżniczkowalną: $f(x) = \text{sgn}(x)x + x^2$
2. Funkcję różniczkowalną jednokrotnie: $f(x) = \text{sgn}(x)x^2$
3. Funkcję różniczkowalną trzykrotnie: $f(x) = |\sin(5x)|^3$
4. Trzy funkcje analityczne: $f(x) = \dfrac{1}{1 + ax^2}$ dla $a \in \{1, 25, 100\}$
5. Funkcję nieciągłą: $f(x) = \text{sgn}(x)$

Funkcje można zaimplementować w notaniku lub w pliku main. Do definicji funkcji w notatniku można użyć [wyrażenia lambda (https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions)](https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions).

In [3]:
```
def funkcja1(x):
    return np.sign(x)*x+x**2

def funkcja2(x):
    return np.sign(x)*x**2

def funkcja3(x):
    return (np.abs(np.sin(5*x)))**3

def funkcja4(x, a=1):
    return 1 / (1 + a*x**2)

def funkcja5(x, a=25):
    return 1 / (1 + a*x**2)

def funkcja6(x, a=100):
    return 1 / (1 + a*x**2)

def funkcja7(x):
    return np.sign(x)

x = 4.0
print("Funkcja 1: ", funkcja1(x))
print("Funkcja 2: ", funkcja2(x))
print("Funkcja 3: ", funkcja3(x))
print("Funkcja 4: ", funkcja4(x))
print("Funkcja 5: ", funkcja5(x))
print("Funkcja 6: ", funkcja6(x))
print("Funkcja 7: ", funkcja7(x))
```

```
Funkcja 1:  20.0
Funkcja 2:  16.0
Funkcja 3:  0.7609115933212749
Funkcja 4:  0.058823529411764705
Funkcja 5:  0.0024937655860349127
Funkcja 6:  0.0006246096189881324
Funkcja 7:  1.0
```

Zadanie 3.

Dla funkcji ciągłej nieróżniczkowalnej z [Zadania 2](#) przeprowadzić interpolację metodą [barycentryczną (https://people.maths.ox.ac.uk/trefethen/barycentric.pdf)](https://people.maths.ox.ac.uk/trefethen/barycentric.pdf) przy użyciu funkcji [*barycentric_interpolate (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.interpolate.barycentric_interpolate.html)*](https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.interpolate.barycentric_interpolate.html) z pakietu [*Scipy (https://scipy.org/)*](https://scipy.org/) oraz przy użyciu wzoru barycentrycznego podanego na wykładzie (funkcja barycentric_inte w main.py).

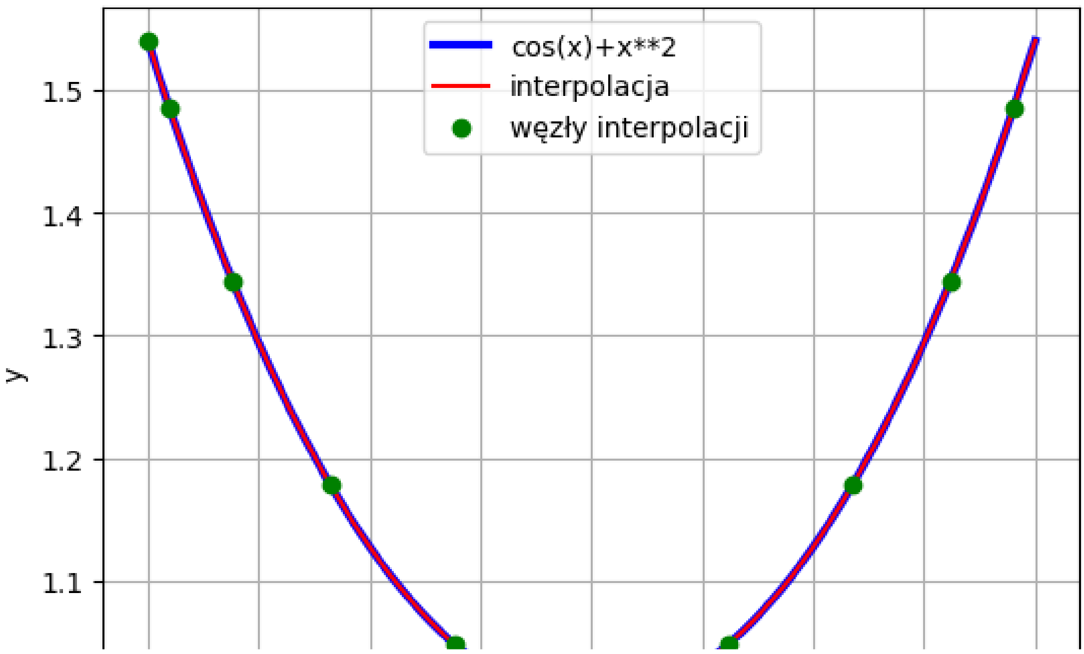
Wykonać w węzłach Czebyszewa interpolację rzędu 10, 100, 1000, 10000, 100000. Wyliczyć wartości wielomianu interpolacyjnego w równoodległych punktach w ilości 1000.

Wykreślić wykresy obrazujące wyniki interpolacji (wykres oryginalnej funkcji i funkcji interpolującej w 1000 punktów).

Przeanalizować czas obliczeń w zależności od rzędu interpolacji.

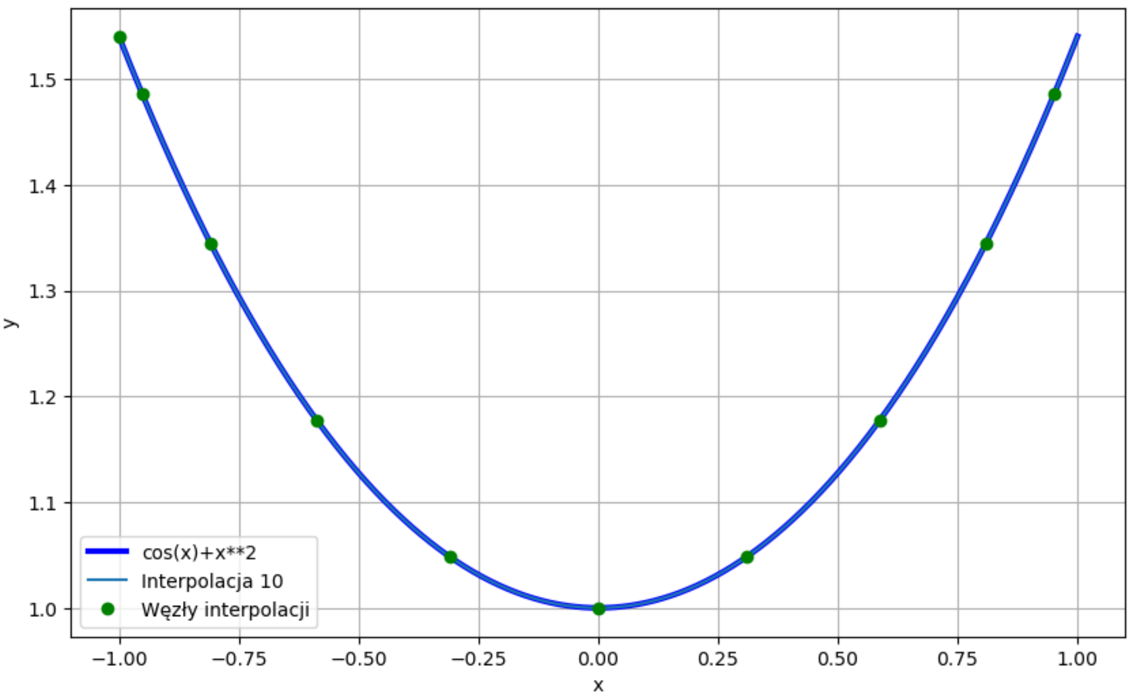
Przykład użycia funkcji barycentric_interpolate:

In [4]:
```
from scipy.interpolate import barycentric_interpolate

# funkcja do interpolacji
f = lambda x: np.cos(x)+x**2

# wektor współrzędnych x dla których chcemy wyliczyć wartości wielomianu int
x = np.linspace(-1,1,1000)

# węzły Czebyszewa
interpolation_nodes_number = 1e1
xch = np.cos(np.linspace(1,interpolation_nodes_number,int(interpolation_node

# interpoalcja metodą barycentryczną
yimp = barycentric_interpolate(xch,f(xch),x)

plt.plot(x,f(x),'b',  linewidth=3 ,label = 'cos(x)+x**2')
plt.plot(x,yimp,'r',label = 'interpolacja')
plt.plot(xch,f(xch),'go',label = 'węzły interpolacji')
plt.xlabel("x")
plt.ylabel("y")
plt.legend(loc = 0)
plt.grid()
plt.show()
```



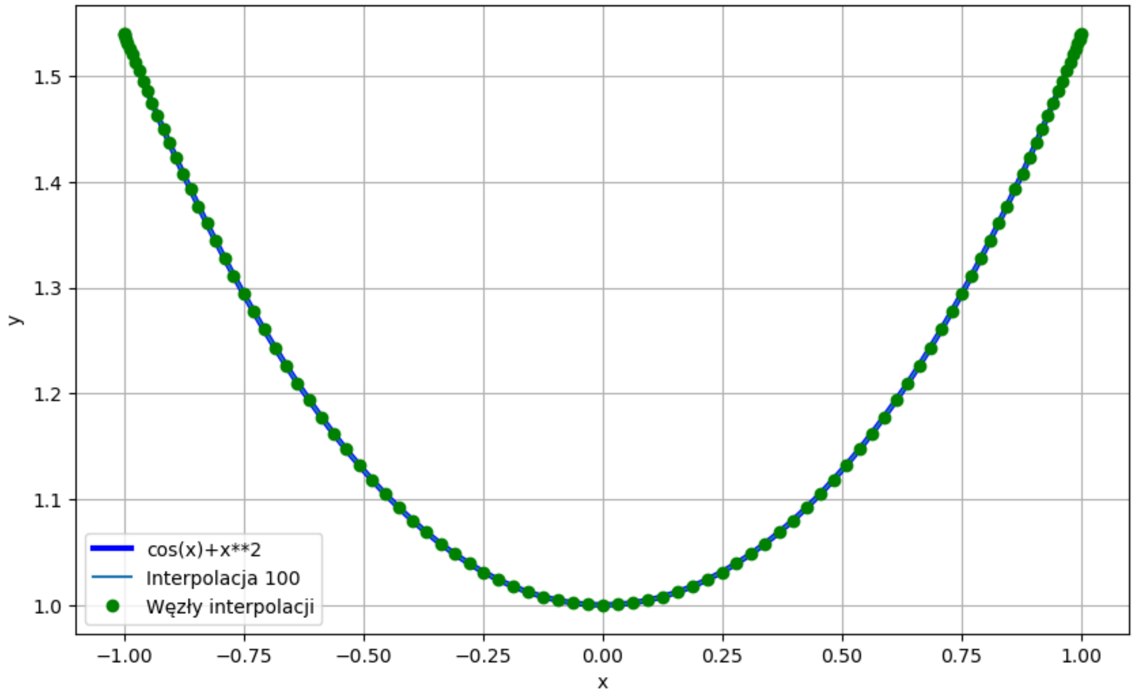
In [5]:
```
from scipy.interpolate import barycentric_interpolate
import time

# funkcja do interpolacji
f = lambda x: np.cos(x) + x**2

# wektor współrzędnych x dla których chcemy wyliczyć wartości wielomianu int
x = np.linspace(-1, 1, 1000)
```

In [6]:
```
# Rząd interpolacji
interpolation_nodes_number = 10

xch = np.cos(np.linspace(1, interpolation_nodes_number, interpolation_nodes_

start_time = time.time()
yimp = barycentric_interpolate(xch, f(xch), x)
end_time = time.time()

elapsed_time = end_time - start_time

plt.figure(figsize=(10, 6))
plt.plot(x, f(x), 'b', linewidth=3, label='cos(x)+x**2')
plt.plot(x, yimp, label=f'Interpolacja {interpolation_nodes_number}')
plt.plot(xch, f(xch), 'go', label='Węzły interpolacji')
plt.xlabel("x")
plt.ylabel("y")
plt.legend(loc=0)
plt.grid()
plt.show()

print(f'Czas obliczeń dla interpolacji rzędu {interpolation_nodes_number}: {
```



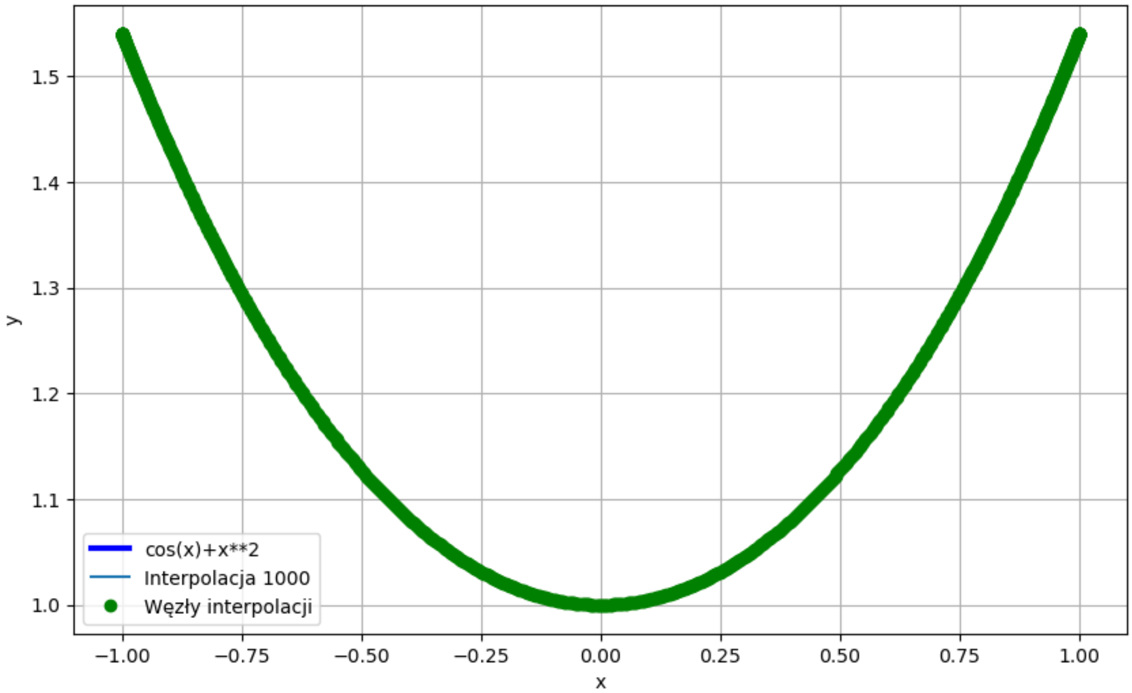

```
Czas obliczeń dla interpolacji rzędu 10: 0.000999450683359375 sekundy
```

In [7]:
```
# Rząd interpolacji
interpolation_nodes_number = 100

xch = np.cos(np.linspace(1, interpolation_nodes_number, interpolation_nodes_

start_time = time.time()
yimp = barycentric_interpolate(xch, f(xch), x)
end_time = time.time()

elapsed_time = end_time - start_time

plt.figure(figsize=(10, 6))
plt.plot(x, f(x), 'b', linewidth=3, label='cos(x)+x**2')
plt.plot(x, yimp, label=f'Interpolacja {interpolation_nodes_number}')
plt.plot(xch, f(xch), 'go', label='Węzły interpolacji')
plt.xlabel("x")
plt.ylabel("y")
plt.legend(loc=0)
plt.grid()
plt.show()

print(f'Czas obliczeń dla interpolacji rzędu {interpolation_nodes_number}: {
```



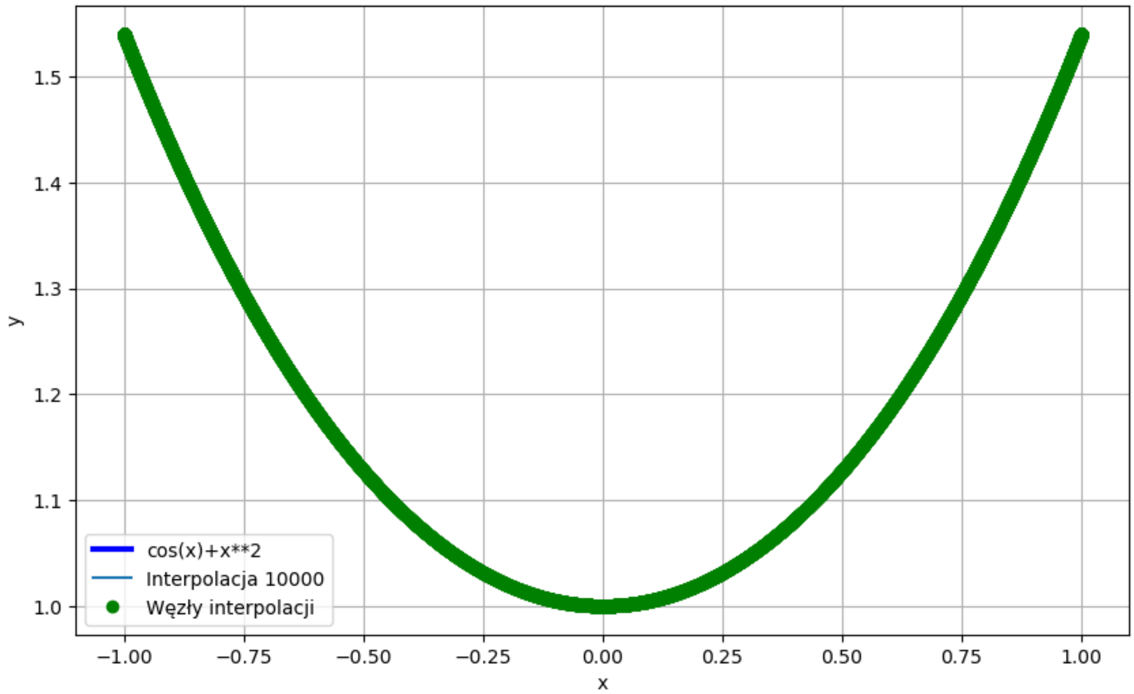

```
Czas obliczeń dla interpolacji rzędu 100: 0.0019998550415039062 sekundy
```

In [8]:
```
# Rząd interpolacji
interpolation_nodes_number = 1000

xch = np.cos(np.linspace(1, interpolation_nodes_number, interpolation_nodes_

start_time = time.time()
yimp = barycentric_interpolate(xch, f(xch), x)
end_time = time.time()

elapsed_time = end_time - start_time

plt.figure(figsize=(10, 6))
plt.plot(x, f(x), 'b', linewidth=3, label='cos(x)+x**2')
plt.plot(x, yimp, label=f'Interpolacja {interpolation_nodes_number}')
plt.plot(xch, f(xch), 'go', label='Węzły interpolacji')
plt.xlabel("x")
plt.ylabel("y")
plt.legend(loc=0)
plt.grid()
plt.show()

print(f'Czas obliczeń dla interpolacji rzędu {interpolation_nodes_number}: {
```



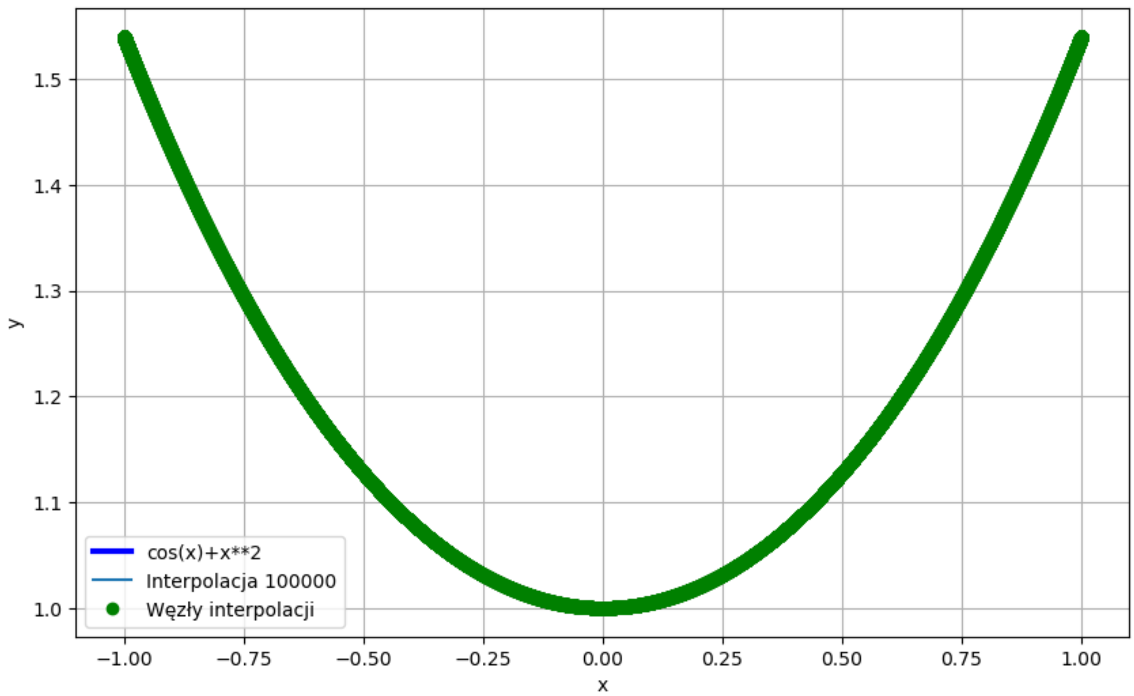

```
Czas obliczeń dla interpolacji rzędu 1000: 0.01600337028503418 sekundy
```

In [9]:
```
# Rząd interpolacji
interpolation_nodes_number = 10000

xch = np.cos(np.linspace(1, interpolation_nodes_number, interpolation_nodes_

start_time = time.time()
yimp = barycentric_interpolate(xch, f(xch), x)
end_time = time.time()

elapsed_time = end_time - start_time

plt.figure(figsize=(10, 6))
plt.plot(x, f(x), 'b', linewidth=3, label='cos(x)+x**2')
plt.plot(x, yimp, label=f'Interpolacja {interpolation_nodes_number}')
plt.plot(xch, f(xch), 'go', label='Węzły interpolacji')
plt.xlabel("x")
plt.ylabel("y")
plt.legend(loc=0)
plt.grid()
plt.show()

print(f'Czas obliczeń dla interpolacji rzędu {interpolation_nodes_number}: {
```




```
Czas obliczeń dla interpolacji rzędu 10000: 0.3901989459991455 sekundy
```

In [10]:
```
# Rząd interpolacji
interpolation_nodes_number = 100000

xch = np.cos(np.linspace(1, interpolation_nodes_number, interpolation_nodes_

start_time = time.time()
yimp = barycentric_interpolate(xch, f(xch), x)
end_time = time.time()

elapsed_time = end_time - start_time

plt.figure(figsize=(10, 6))
plt.plot(x, f(x), 'b', linewidth=3, label='cos(x)+x**2')
plt.plot(x, yimp, label=f'Interpolacja {interpolation_nodes_number}')
plt.plot(xch, f(xch), 'go', label='Węzły interpolacji')
plt.xlabel("x")
plt.ylabel("y")
plt.legend(loc=0)
plt.grid()
plt.show()

print(f'Czas obliczeń dla interpolacji rzędu {interpolation_nodes_number}: {
```




```
Czas obliczeń dla interpolacji rzędu 100000: 55.33916735649109 sekundy
```

Zadanie 4.
Do oceny jakości interpolacji stosuje się normę wektorową l_∞, dla różnicy wektorów definiujemy ją jako:

$$||\mathbf{x_1} - \mathbf{x_2}|| = \sup\{|\mathbf{x_1} - \mathbf{x_2}|\}$$

Zaimplementuj normę l_∞ jako funkcję w main *L_inf*. Za jej pomocą zbadaj jakość wszystkich przeprowadzonych interpolacji.

In [11]:
```
from scipy.interpolate import barycentric_interpolate

def L_inf(x1, x2):
    return np.max(np.abs(x1 - x2))

x_uniform = np.linspace(-1, 1, 1000)

rozniczka_nie = lambda x: np.sign(x) * x + x**2
rozniczka1 = lambda x: np.sign(x) * x**2
rozniczka3 = lambda x: np.abs(np.sin(5 * x))**3
analytic1 = lambda x: 1 / (1 + 1 * x**2)
analytic25 = lambda x: 1 / (1 + 25 * x**2)
analytic100 = lambda x: 1 / (1 + 100 * x**2)
nieciagla = lambda x: np.sign(x)

f = lambda x: np.cos(x) + x**2

interpolation_nodes_numbers = [10, 100, 1000, 10000, 100000]

for interpolation_nodes_number in interpolation_nodes_numbers:
    xch = np.cos(np.linspace(1, interpolation_nodes_number, interpolation_nc

    start_time = time.time()
    # Interpolacja metodą barycentryczną
    yimp = barycentric_interpolate(xch, f(xch), x_uniform)
    end_time = time.time()

    # Obliczenie błędów
    error_nieciagla = L_inf(f(x_uniform), nieciagla(x_uniform))
    error_rozniczka_nie = L_inf(f(x_uniform), rozniczka_nie(x_uniform))
    error_rozniczka1 = L_inf(f(x_uniform), rozniczka1(x_uniform))
    error_rozniczka3 = L_inf(f(x_uniform), rozniczka3(x_uniform))
    error_analytic1 = L_inf(f(x_uniform), analytic1(x_uniform))
    error_analytic25 = L_inf(f(x_uniform), analytic25(x_uniform))
    error_analytic100 = L_inf(f(x_uniform), analytic100(x_uniform))

    elapsed_time = end_time - start_time
    # Wyświetlenie wyników
    print(f'\nDla interpolacji o wartości {interpolation_nodes_number}:')
    print(f'Błąd w funkcji niestałej: {error_nieciagla}')
    print(f'Błąd w funkcji rozniczka_nie: {error_rozniczka_nie}')
    print(f'Błąd w funkcji rozniczka1: {error_rozniczka1}')
    print(f'Błąd w funkcji rozniczka3: {error_rozniczka3}')
    print(f'Błąd w funkcji analytic1: {error_analytic1}')
    print(f'Błąd w funkcji analytic25: {error_analytic25}')
    print(f'Błąd w funkcji analytic100: {error_analytic100}')
    print(f'\n Czas interpolacji: {elapsed_time} sekund')
```

```
Dla interpolacji o wartości 10:
Błąd w funkcji niestałej: 2.5403023058681398
Błąd w funkcji rozniczka_nie: 0.9989984979975388
Błąd w funkcji rozniczka1: 2.5403023058681398
Błąd w funkcji rozniczka3: 1.2240054184510198
Błąd w funkcji analytic1: 1.0403023058681398
Błąd w funkcji analytic25: 1.5018407674066012
Błąd w funkcji analytic100: 1.5304013157691299

 Czas interpolacji: 0.0010001659393310547 sekund

Dla interpolacji o wartości 100:
Błąd w funkcji niestałej: 2.5403023058681398
Błąd w funkcji rozniczka_nie: 0.9989984979975388
Błąd w funkcji rozniczka1: 2.5403023058681398
Błąd w funkcji rozniczka3: 1.2240054184510198
Błąd w funkcji analytic1: 1.0403023058681398
Błąd w funkcji analytic25: 1.5018407674066012
Błąd w funkcji analytic100: 1.5304013157691299

 Czas interpolacji: 0.0019998550415039062 sekund

Dla interpolacji o wartości 1000:
Błąd w funkcji niestałej: 2.5403023058681398
Błąd w funkcji rozniczka_nie: 0.9989984979975388
Błąd w funkcji rozniczka1: 2.5403023058681398
Błąd w funkcji rozniczka3: 1.2240054184510198
Błąd w funkcji analytic1: 1.0403023058681398
Błąd w funkcji analytic25: 1.5018407674066012
Błąd w funkcji analytic100: 1.5304013157691299

 Czas interpolacji: 0.015003204345703125 sekund

Dla interpolacji o wartości 10000:
Błąd w funkcji niestałej: 2.5403023058681398
Błąd w funkcji rozniczka_nie: 0.9989984979975388
Błąd w funkcji rozniczka1: 2.5403023058681398
Błąd w funkcji rozniczka3: 1.2240054184510198
Błąd w funkcji analytic1: 1.0403023058681398
Błąd w funkcji analytic25: 1.5018407674066012
Błąd w funkcji analytic100: 1.5304013157691299

 Czas interpolacji: 0.39632654190063477 sekund

Dla interpolacji o wartości 100000:
Błąd w funkcji niestałej: 2.5403023058681398
Błąd w funkcji rozniczka_nie: 0.9989984979975388
Błąd w funkcji rozniczka1: 2.5403023058681398
Błąd w funkcji rozniczka3: 1.2240054184510198
Błąd w funkcji analytic1: 1.0403023058681398
Błąd w funkcji analytic25: 1.5018407674066012
Błąd w funkcji analytic100: 1.5304013157691299

 Czas interpolacji: 55.75437521934509 sekund
```

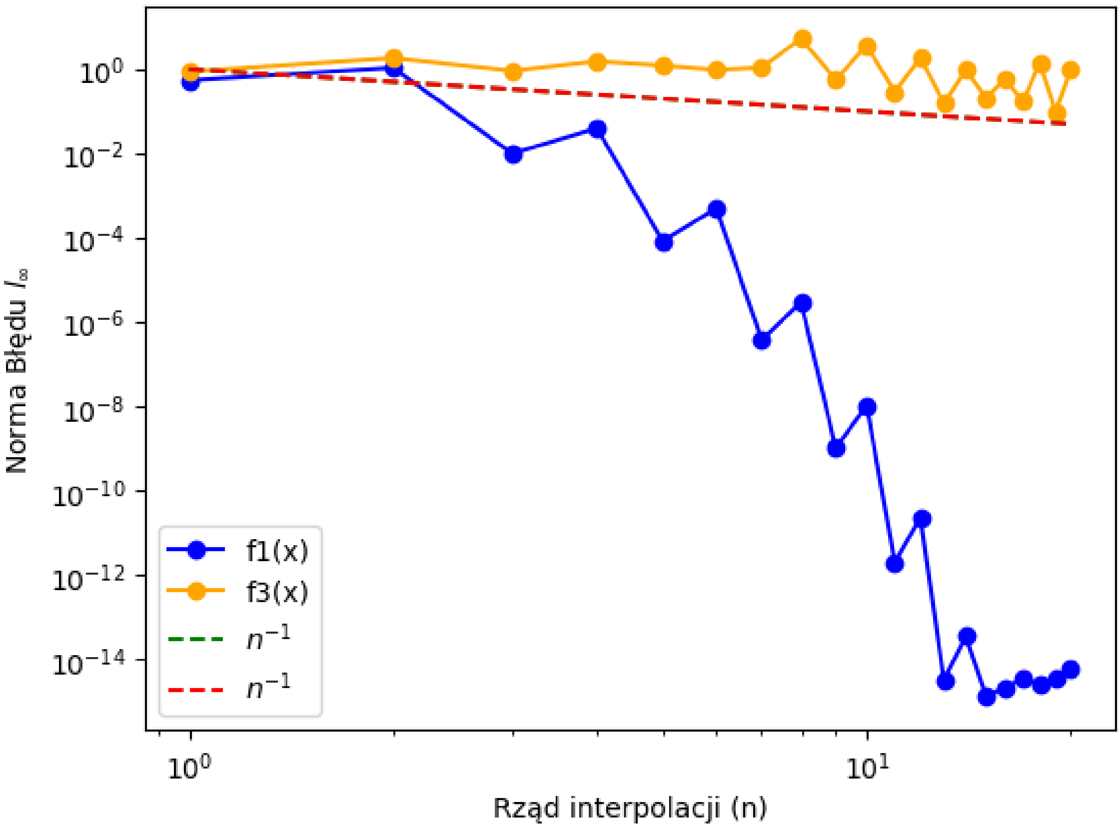
Zadanie 5.

Dla funkcji jednokrotnie i trzykrotnie różniczkowalnej z [Zadania 2](#). Przeanalizować w pętli jakość interpolacji dla różnych rzędów interpolacji n. W tym celu należy wyliczyć wartość funkcji i wielomianu interpolacyjnego w 1000 punktów i wyliczyć normę różnicy tych wektorów (normę błędu) dla każdego badanego rzędu. Maksymalny rząd należy przyjąć gdy błąd będzie na poziomie zera maszynowego. Dla każdej z funkcji sporządzić wykres w skali podwójnie logarytmicznej (obie osie), w którym oś argumentów to rząd interpolacji a oś wartości to odpowiadająca mu norma błędu. Dla porównania umieścić na wykresie dodatkowo wykres n^{-v}, gdzie v to rząd najwyższej pochodnej funkcji (zobacz wykład).

In [12]:
```
def L_inf(x1, x2):
    return np.max(np.abs(x1 - x2))

def find_highest_derivative_order(f):
    # Szukanie najwyższego rzędu pochodnej funkcji
    order = 0
    while True:
        try:
            f = np.gradient(f)
            order += 1
        except ValueError:
            break
    return order

def interpol_error(interpolation_function, k, f, x):
    xch = np.cos(np.linspace(1, k, k) * np.pi / k)
    yimp = interpolation_function(xch, f(xch), x)
    error = L_inf(f(x), yimp)
    return error

# Funkcje z Zadania 2
f1 = lambda x: np.cos(x) + x**2
f3 = lambda x: np.abs(np.sin(5 * x))**3

# Maksymalny rząd pochodnej
v1 = find_highest_derivative_order(f1)
v3 = find_highest_derivative_order(f3)

# Zakres rzędów interpolacji
max_interpolation_order = 20
interpolation_orders = np.arange(1, max_interpolation_order + 1)

errors_f1 = []
errors_f3 = []

# Obliczenia dla każdego rzędu interpolacji
for n in interpolation_orders:
    xch = np.cos(np.linspace(1, n, n) * np.pi / n)

    # Interpolacja metodą barycentryczną
    yimp_f1 = barycentric_interpolate(xch, f1(xch), x_uniform)
    yimp_f3 = barycentric_interpolate(xch, f3(xch), x_uniform)

    # Obliczenie błędów
    error_f1 = interpol_error(barycentric_interpolate, n, f1, x_uniform)
    error_f3 = interpol_error(barycentric_interpolate, n, f3, x_uniform)

    errors_f1.append(error_f1)
    errors_f3.append(error_f3)

# Wykres
plt.loglog(interpolation_orders, errors_f1, '-o', label='f1(x)', color='blue
plt.loglog(interpolation_orders, errors_f3, '-o', label='f3(x)', color='orar

# Wykres n^{-v}
plt.loglog(interpolation_orders, 1/np.power(interpolation_orders, v1), '--',
plt.loglog(interpolation_orders, 1/np.power(interpolation_orders, v3), '--',

plt.xlabel('Rząd interpolacji (n)')
plt.ylabel('Norma Błędu $l_\infty$')
plt.legend()
```

```
plt.show()
```

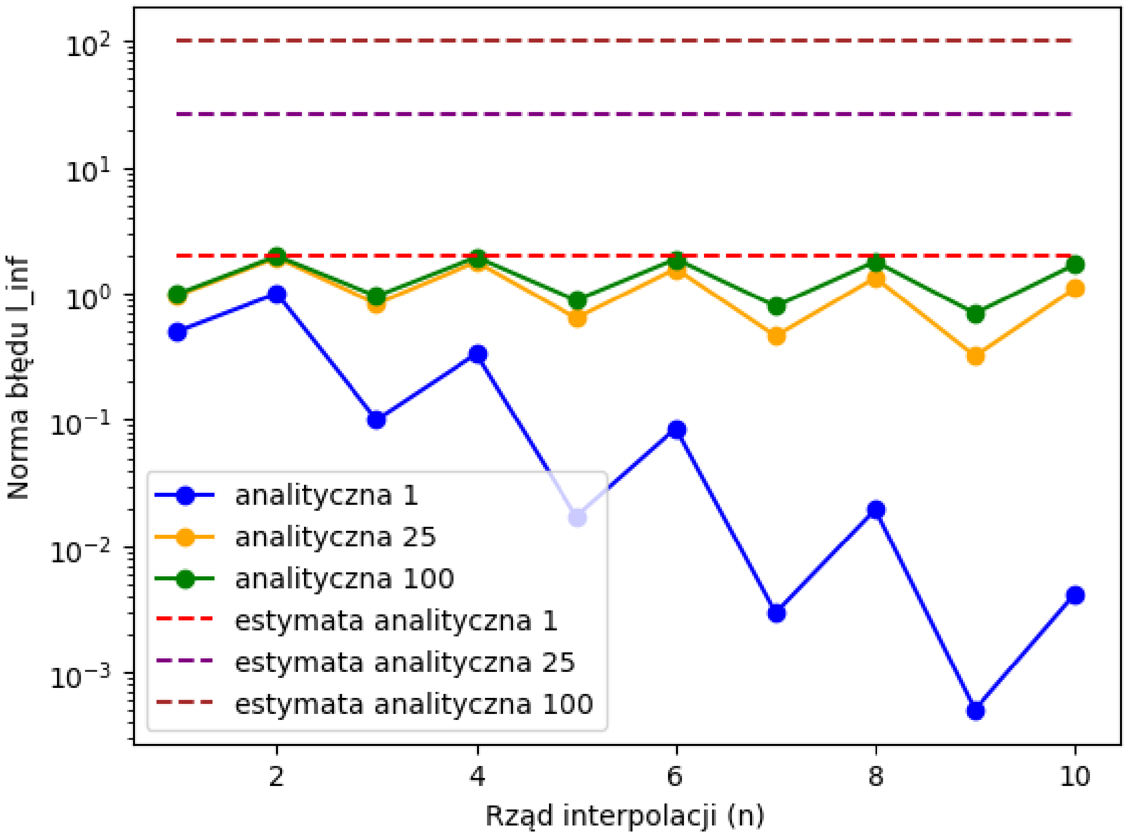
```
C:\Users\Piotrek\anaconda3\Lib\site-packages\scipy\interpolate\_polyint.p
y:567: RuntimeWarning: divide by zero encountered in scalar divide
  self._inv_capacity = 4.0 / (np.max(self.xi) - np.min(self.xi))
C:\Users\Piotrek\anaconda3\Lib\site-packages\scipy\interpolate\_polyint.p
y:574: RuntimeWarning: invalid value encountered in multiply
  dist = self._inv_capacity * (self.xi[i] - self.xi[permute])
```



Zadanie 6.

Przeprowadzić analogiczną analizę dla funkcji analitycznych z [Zadania 2](). Wykres sporządzić w skali pół logarytmicznej (tylko oś y). Dla porównania umieścić na wykresie dodatkowo wykres oszacowania dla interpolacji funkcji analitycznych (zobacz wykład). W tym celu należy wyliczyć maksimum funkcji na przedziale $[-1, 1]$ oraz największą elipsę, o ogniskach w punktach $(-1, j0)$ i $(1, j0)$, która nie zawiera pierwiastków mianownika funkcji.

In [13]:
```
analytic1 = lambda x: 1 / (1 + 1 * x**2)
analytic25 = lambda x: 1 / (1 + 25 * x**2)
analytic100 = lambda x: 1 / (1 + 100 * x**2)

def L_inf(x1, x2):
    return np.max(np.abs(x1 - x2))

def interp_error(interpolation_function, k, f, x):
    xch = np.cos(np.linspace(1, k, k) * np.pi / k)
    yimp = interpolation_function(xch, f(xch), x)
    error = L_inf(f(x), yimp)
    return error

def interp_estimate(f, x):
    max_value = np.max(np.abs(f(x)))
    return max_value * np.exp(np.max(np.abs(np.log(np.abs(f(x))))))

x = np.linspace(-1, 1, 1000)
interp_order = 10
interp_orders = np.arange(1, interp_order + 1)

errors_analytic1 = []
errors_analytic25 = []
errors_analytic100 = []
estimates_analytic1 = []
estimates_analytic25 = []
estimates_analytic100 = []

for n in interp_orders:
    error_analytic1 = interp_error(barycentric_interpolate, n, analytic1, x)
    error_analytic25 = interp_error(barycentric_interpolate, n, analytic25,
    error_analytic100 = interp_error(barycentric_interpolate, n, analytic10

    estimate_analytic1 = interp_estimate(analytic1, x)
    estimate_analytic25 = interp_estimate(analytic25, x)
    estimate_analytic100 = interp_estimate(analytic100, x)

    errors_analytic1.append(error_analytic1)
    errors_analytic25.append(error_analytic25)
    errors_analytic100.append(error_analytic100)

    estimates_analytic1.append(estimate_analytic1)
    estimates_analytic25.append(estimate_analytic25)
    estimates_analytic100.append(estimate_analytic100)

plt.semilogy(interp_orders, errors_analytic1, '-o', label='analityczna 1', c
plt.semilogy(interp_orders, errors_analytic25, '-o', label='analityczna 25',
plt.semilogy(interp_orders, errors_analytic100, '-o', label='analityczna 10

plt.semilogy(interp_orders, estimates_analytic1, '--', label='estymata anali
plt.semilogy(interp_orders, estimates_analytic25, '--', label='estymata anal
plt.semilogy(interp_orders, estimates_analytic100, '--', label='estymata ana

plt.xlabel('Rząd interpolacji (n)')
plt.ylabel('Norma błędu l_inf')
plt.legend()
plt.show()
```

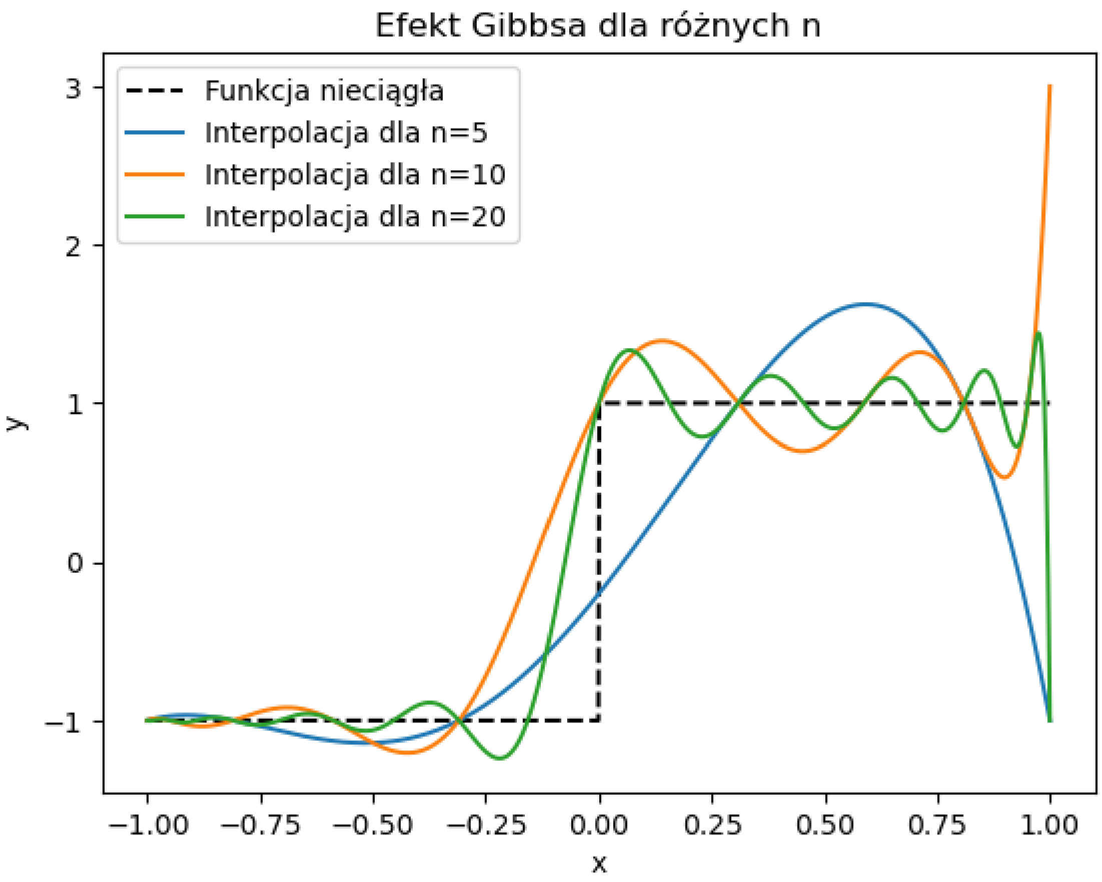


Zadanie 7.

Dla funkcji nieciągłej z [Zadania 2](#) przeanalizować efekt Gibbsa oddzielnie dla parzystych i nieparzystych n. Oddzielnie wyliczyć jaki jest minimalny błąd niezależny od rzędu. Dlaczego wartości dla parzystych i nieparzystych n się różnią?

Wskazówka: Wykonać wykres funkcji i jej funkcji interpolującej.

In [14]:
```
nieciagla = lambda x: np.sign(x)
x = np.linspace(-1, 1, 1000)
n_values = [5, 10, 20]

plt.plot(x, nieciagla(x), label='Funkcja nieciągła', linestyle='--', color='

for n in n_values:
    xch = np.cos(np.linspace(1, n, n) * np.pi / n)
    yimp = barycentric_interpolate(xch, nieciagla(xch), x)
    plt.plot(x, yimp, label=f'Interpolacja dla n={n}')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Efekt Gibbsa dla różnych n')
plt.legend()
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/)
- [Learn Python for Data Science (https://www.datacamp.com/learn-python-with-anaconda?utm_source=Anaconda_download&utm_campaign=datacamp_training&utm_medium=bar](https://www.datacamp.com/learn-python-with-anaconda?utm_source=Anaconda_download&utm_campaign=datacamp_training&utm_medium=bar)
- [Learn Python (https://www.learnpython.org/)](https://www.learnpython.org/)