# Laboratorium 9
# Metody Numeryczne

Instrukcja:
Na zajęciach należy wykonać poniższe zadania, a następnie sporządzić sprawozdanie zawierające odpowiedzi (w postaci kodu) z komentarzami w środowisku Jupyter Notebook i umieścić je na platformie e-learningowej.

***Materiały przygotowujące:***

In [2]:

```
import numpy as np
import scipy
import matplotlib
import matplotlib.pyplot as plt
import math
from scipy.optimize import root, fsolve, root_scalar
import time
import typing
import types
import pickle
from inspect import isfunction


from typing import Union, List, Tuple

def fun(x):
    return None

def dfun(x):
    return None

def ddfun(x):
    return None

def bisection(a: Union[int,float], b: Union[int,float], f: typing.Callable[[
    '''funkcja aproksymująca rozwiązanie równania f(x) = 0 na przedziale [a,

    Parametry:
    a - początek przedziału
    b - koniec przedziału
    f - funkcja dla której jest poszukiwane rozwiązanie
    epsilon - tolerancja zera maszynowego (warunek stopu)
    iteration - ilość iteracji

    Return:
    float: aproksymowane rozwiązanie
    int: ilość iteracji
    '''
    return None


def difference_quotient(f: typing.Callable[[float], float],x: Union[int,floa
    '''Funkcja obliczająca iloaz różnicowy zadanej funkcji
    Parametry:

    f - funkcja dla której jest poszukiwane rozwiązanie
    x - argument funkcji la której jest
    h - krok różnicy wykorzystywanej do wyliczenia ilorazu różnicowego

    return:
    diff - wartość ilorazu różnicowego

    '''
    return None

def newton(f: typing.Callable[[float], float], df: typing.Callable[[float],
    ''' Funkcja aproksymująca rozwiązanie równania f(x) = 0 metodą Newtona.
    Parametry:
    f - funkcja dla której jest poszukiwane rozwiązanie
    df - pochodna funkcji dla której jest poszukiwane rozwiązanie
```

```
    ddf - druga pochodna funkcji dla której jest poszukiwane rozwiązanie
    a - początek przedziału
    b - koniec przedziału
    epsilon - tolerancja zera maszynowego (warunek stopu)
    Return:
    float: aproksymowane rozwiązanie
    int: ilość iteracji
    '''
    return None
```

***Temat główny:***

Znajdź miejsca zerowe funkcji:

$$f(x) = e^{-2x} + x^2 - 1$$

,

metodami:

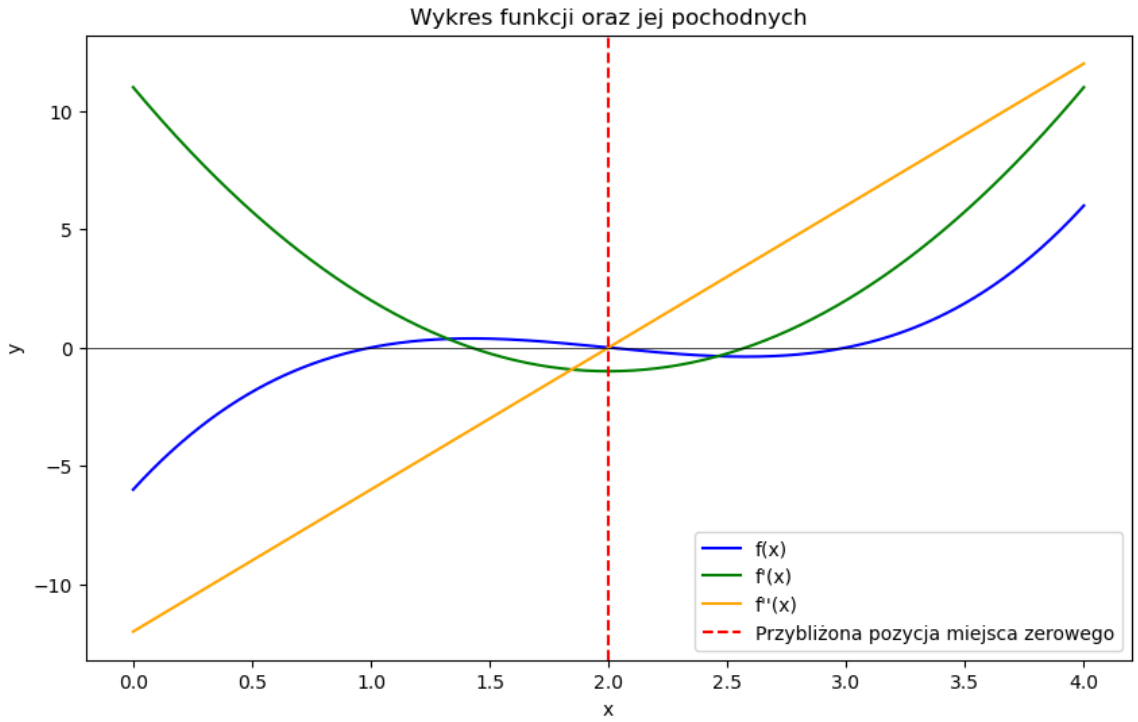
- Bisekcji,
- Newtona

Funkcja i jej pochodne zostały zaimplementowane w *main.py*

***Zadanie 1.***
Wykonaj wykres funkcji oraz jej pierwszej i drugiej pochodnej (obliczonej analitycznie) na jednym rysunku w przedziale pozwalającym na zgrubne określenie miejsc zerowych. Wykres powinien być odpowiednio opisany. Określ przedziały, w którym znajdują się miejsca zerowe naszej funkcji.

W jaki sposób (wykorzystując pierwszą i drugą pochodną) można znaleźć miejsca zerowe funkcji?

In [3]:
```
def function(x):
    return x**3 - 6*x**2 + 11*x - 6

def first_derivative(x):
    return 3*x**2 - 12*x + 11

def second_derivative(x):
    return 6*x - 12

x_vals = np.linspace(0, 4, 400)

y_vals = function(x_vals)
y_prime_vals = first_derivative(x_vals)
y_double_prime_vals = second_derivative(x_vals)

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(x_vals, y_vals, label='f(x)', color='blue')
ax.plot(x_vals, y_prime_vals, label="f'(x)", color='green')
ax.plot(x_vals, y_double_prime_vals, label="f''(x)", color='orange')

ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(2, color='red', linestyle='--', label='Przybliżona pozycja miejsc

ax.set_title('Wykres funkcji oraz jej pochodnych')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()

plt.show()
```




1. Miejsca zerowe funkcji można znaleźć, posługując się pierwszą pochodną, gdzie punkty, w których pochodna zmienia znak, są potencjalnymi miejscami zerowymi.
2. W tych punktach pierwsza pochodna osiąga ekstremum, czyli przybliżone miejsce zerowe można znaleźć, gdzie pochodna pierwsza osiąga minimum lub maksimum lokalne.

3. Następnie, wykorzystując drugą pochodną, sprawdzamy, czy w tych miejscach istnieją punkty przegięcia, czyli gdzie druga pochodna zmienia znak.
4. Miejsca, w których pierwsza pochodna osiąga ekstremum lokalne i druga pochodna zmienia znak, są kandydatami na rzeczywiste miejsca zerowe funkcji.
5. Ostateczne znalezienie miejsc zerowych wymaga dokładniejszych metod, takich jak metoda Newtona czy bisekcji, jednak początkowe przybliżenia można uzyskać analizując pochodne funkcji.

***Zadanie 2.***
Najprostszą metodą do wyznaczenia miejsca zerowego funkcji nieliniowej jest metoda bisekcji. Zaimplementuj metodę bisekcji (https://en.wikipedia.org/wiki/Bisection_method)

Pamiętaj, że gwarancją zbieżności działania funkcji są założenia:

1. funkcja jest ciągła na danym przedziale $[a, b]$
2. wartość funkcji na końcach przedziału przyjmuje przeciwne znaki (tzn. $f(a) * f(b) < 0$)

In [4]:
```
def bisection_method(func, a, b, tol=1e-6, max_iter=100):
    if func(a) * func(b) >= 0:
        raise ValueError("Warunek func(a) * func(b) < 0 nie jest spełniony.

    iteration = 0
    while (b - a) / 2 > tol and iteration < max_iter:
        c = (a + b) / 2
        if func(c) == 0:
            return c
        elif func(c) * func(a) < 0:
            b = c
        else:
            a = c
        iteration += 1

    return (a + b) / 2

def sample_function(x):
    return x**2 - 4

zero_location = bisection_method(sample_function, 0, 3)
print(f"Miejsce zerowe funkcji: {zero_location}")
```

```
Miejsce zerowe funkcji: 2.000000238418579
```

***Zadanie 3.***
Inną metodą, wykorzystywaną do poszukiwania miejsca zerowego funkcji jest metoda Newtona. Wykorzystuję ona wartość pierwszej pochodnej do wyznaczenia wartości.

1. zaimplementuj iloraz różnicowy.
2. wygeneruj wektor 10 elementowy
3. sprawdź działanie funkcji dla danego wektora oraz $h = 0.00001$ oraz z wartościami uzyskanymi z funkcją wyliczoną analityczną.

Zaimplementuj metodę Newtona (https://en.wikipedia.org/wiki/Newton%27s_method)

Gwarancja zbieżności:

1. funkcja jest ciągła na danym przedziale $[a, b]$
2. Pierwsza i druga pochodna istnieją i są ciągłe w przedziale domkniętym $[a, b]$
3. funkcja na końcach przedziału przyjmuje przeciwne znaki
4. pierwsza i druga pochodna mają stały znak (brak ekstremów lokalnych i punktów przegięcia)

In [5]:
```
def finite_difference_quotient(f, x, h=1e-5):
    return (f(x + h) - f(x)) / h

def sample_function(x):
    return x**3 - 6*x**2 + 11*x - 6

def analytical_derivative(x):
    return 3*x**2 - 12*x + 11

vector = np.linspace(0, 5, 10)

finite_differences = finite_difference_quotient(sample_function, vector, h=1

analytical_values = analytical_derivative(vector)
print(f"Wartości ilorazu różnicowego:\n {finite_differences}")
print(f"Wartości uzyskane z funkcji analitycznej: \n {analytical_values}")

def newton_method(f, df, x0, tol=1e-6, max_iter=100):
    iteration = 0
    x = x0

    while abs(f(x)) > tol and iteration < max_iter:
        x = x - f(x) / df(x)
        iteration += 1

    return x

zero_location_newton = newton_method(sample_function, analytical_derivative,
print(f"Miejsce zerowe funkcji (metoda Newtona): {zero_location_newton}")
```

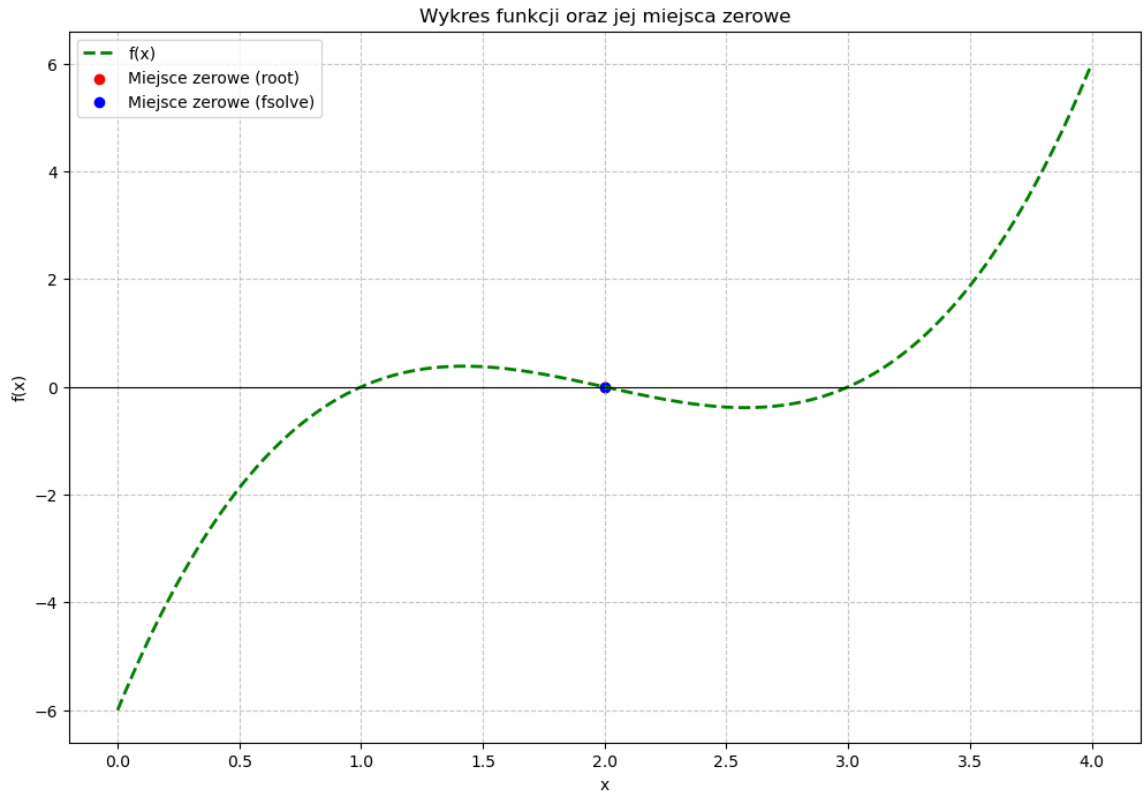
```
Wartości ilorazu różnicowego:
 [10.99994     5.25921593  1.3703437  -0.66667667 -0.85184519  0.81483815
  4.33337333  9.70376037 16.92599926 26.00009   ]
Wartości uzyskane z funkcji analitycznej:
 [11.          5.25925926  1.37037037 -0.66666667 -0.85185185  0.81481481
  4.33333333  9.7037037  16.92592593 26.        ]
Miejsce zerowe funkcji (metoda Newtona): 2
```

***Zadania 4.***

Dla głównej funkcji z zadania 1 znajdź miejsca zerowe przy użyciu:

- funkcji root dostępnej w pakiecie scipy.optimize
- funkcji fsolve dostępnej w pakiecie scipy.optimize

In [6]:
```
root_zero_location = root(sample_function, x0=2)
fsolve_zero_location = fsolve(sample_function, x0=2)

x_vals = np.linspace(0, 4, 400)
y_vals = sample_function(x_vals)

plt.figure(figsize=(12, 8))

plt.plot(x_vals, y_vals, label='f(x)', color='green', linestyle='--', linewi
plt.axhline(0, color='black', linestyle='-', linewidth=0.8)

plt.scatter(root_zero_location.x, [0], color='red', marker='o', label='Miejs
plt.scatter(fsolve_zero_location, [0], color='blue', marker='o', label='Mie

plt.title('Wykres funkcji oraz jej miejsca zerowe')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```




***Zadanie 5.***

Dla głównej funkcji oraz przedziału zdefiniowanego z zadania 1 znajdź miejsca zerowe przy użyciu:

- metody bisekcji
- metody Newtona

z tolerancją $10^{-10}$

Zbadaj dokładność(względem rozwiązania z zadania 4) i czas obliczeń metod w zależności od liczby iteracji. Wyniki przedstaw na wykresach.

In [7]:
```
def bisection(f, a, b, tol=1e-10, max_iter=100):
    iterations_bisection = []
    times_bisection = []
    fsolve_root = fsolve(f, x0=2)
    fsolve_iterations = len(fsolve_root)

    for i in range(1, max_iter + 1):
        iter_start_time = time.time()
        _ = root_scalar(f, bracket=[a, b], method='bisect', options={'maxite
        iter_end_time = time.time()

        iterations_bisection.append(i)
        times_bisection.append(iter_end_time - iter_start_time)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(iterations_bisection, label='Bisection', marker='o')
    plt.axhline(fsolve_iterations, linestyle='--', color='red', label='fsolv
    plt.title('Number of Iterations')
    plt.xlabel('Iterations')
    plt.ylabel('Iterations')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(times_bisection, label='Bisection', marker='o')
    plt.axhline(max(times_bisection), linestyle='--', color='red', label='fs
    plt.title('Computation Time')
    plt.xlabel('Iterations')
    plt.ylabel('Time (s)')
    plt.legend()

    plt.tight_layout()
    plt.show()

bisection(sample_function, 0, 4)
```