In [ ]:
```
#Borsuk Piotr Technologie Przemysłu 4.0
```

# Laboratorium 2
# Metody Numeryczne

Biblioteki niezbędne do wykonania zadania:

In [2]:
```
#import main

import numpy as np
import scipy
import matplotlib
import matplotlib.pyplot as plt
import pickle
import string
import random
```

Niezbędne funkcje do laboratorium:

In [3]:
```
def compare_plot(x1: np.ndarray, y1: np.ndarray, x2: np.ndarray, y2: np.ndar
                 xlabel: str, ylabel: str, title: str, label1: str, label2:
    plt.figure(figsize=(10, 6))
    plt.plot(x1, y1, label=label1)
    plt.plot(x2, y2, label=label2)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()


def parallel_plot(x1, y1, x2, y2, x1label, y1label, x2label, y2label, title,
    if len(x1) != len(y1) or len(x2) != len(y2):
        print("Błąd: Wektory x i y muszą mieć taką samą długość.")
        return None

    if orientation not in ['-', '|']:
        print("Błąd: Parametr 'orientation' musi być '-' lub '|'.")
        return None

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

    if orientation == '-':
        plt.subplot(1, 2, 1)
    else:
        plt.subplot(2, 1, 1)

    plt.plot(x1, y1)
    plt.xlabel(x1label)
    plt.ylabel(y1label)
    plt.title(title)

    if orientation == '-':
        plt.subplot(1, 2, 2)
    else:
        plt.subplot(2, 1, 2)

    plt.plot(x2, y2)
    plt.xlabel(x2label)
    plt.ylabel(y2label)
    plt.title(title)

    plt.tight_layout()
    plt.show()


def log_plot(x, y, xlabel, ylabel, title, log_axis):
    plt.figure(figsize=(8, 6))

    if log_axis == 'xy':
        plt.loglog(x, y)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
    elif log_axis == 'x':
        plt.semilogx(x, y)
        plt.xlabel(xlabel + ' (log scale)')
        plt.ylabel(ylabel)
    elif log_axis == 'y':
        plt.semilogy(x, y)
```

```
        plt.xlabel(xlabel)
        plt.ylabel(ylabel + ' (log scale)')
    else:
        print("Błąd: Parametr 'log_axis' musi być 'x', 'y' lub 'xy'.")
        return None

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

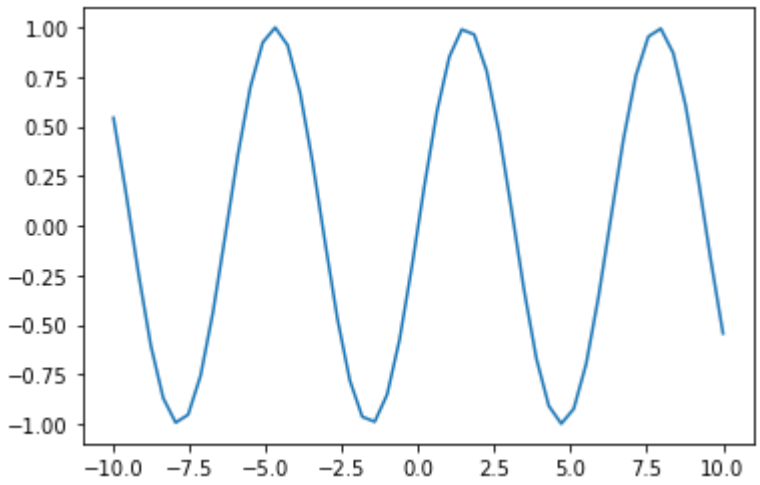
***Zadanie 1.***
Zdefiniuj w notatniku funkcję $f(x) = x^3 - 3x$ i sporządź jej wykres dla argumentów z przedziału:

1. $x \in < -1, 1 >$
2. $x \in < -5, 5 >$
3. $x \in < 0, 5 >$

***Wskazówki*** Jako argumentu funkcji można użyć numpy array.

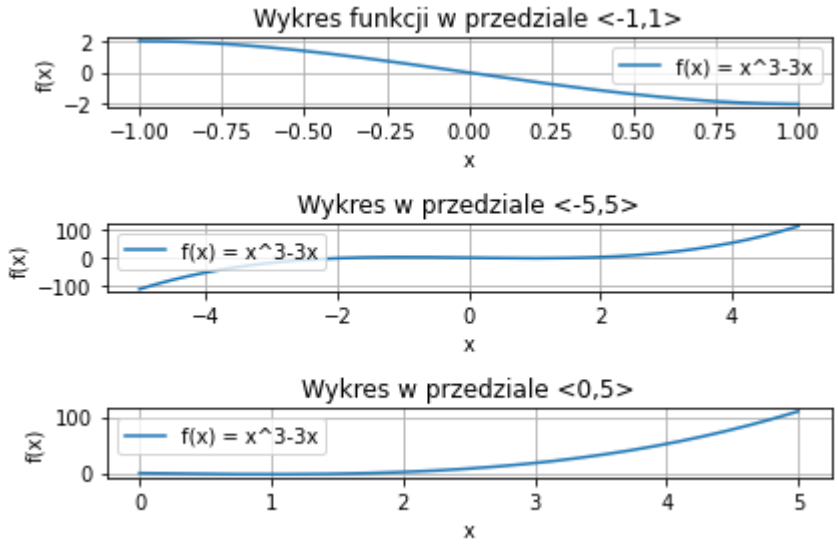
Do wizualizacji wyników w Pythonie używa się pakietu Matplotlib (https://matplotlib.org/). Działanie tego pakietu prezentuje przykład:

In [4]:
```
x = np.linspace(-10,10)
# inicjalizacja wektora f(x) = sin(x)
y = np.sin(x)
plt.plot(x, y, label ='sin(x)')
plt.show()
```



In [5]:
```
def f(y):
    return y**3-3*y

x_1 = np.linspace(-1,1,400)
plt.subplot(3,1,1)
plt.plot(x_1, f(x_1), label = 'f(x) = x^3-3x')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Wykres funkcji w przedziale <-1,1>')
plt.grid(True)
plt.legend()

x_2 = np.linspace(-5,5,400)
plt.subplot(3,1,2)
plt.plot(x_2, f(x_2), label = 'f(x) = x^3-3x')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Wykres w przedziale <-5,5>')
plt.grid(True)
plt.legend()

x_3 = np.linspace(0,5,400)
plt.subplot(3,1,3)
plt.plot(x_3, f(x_3), label = 'f(x) = x^3-3x')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Wykres w przedziale <0,5>')
plt.grid(True)
plt.legend()


plt.tight_layout()
plt.show
```

Out[5]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




Wspierając się dokumentacją Matplotlib (https://matplotlib.org/) dodaj do wykresu etykiety osi, tytuł, grida i legendę.

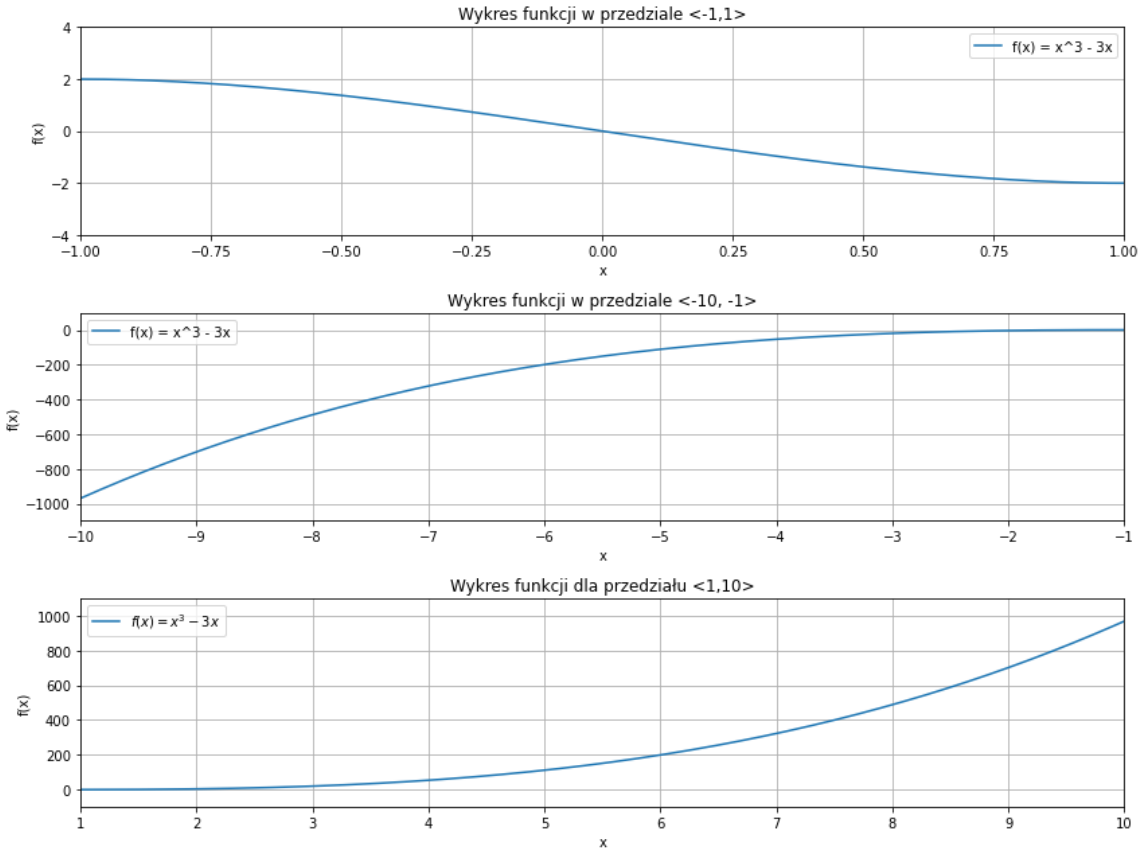
***Zadanie 2.***

Dla funkcji z zadania pierwszego używając wektora $x \in < -10, 10 >$ wyrysuj wykres dla argumentów:

1. $x \in < -1, 1 >$
2. $x \in < -10, -1 >$
3. $x \in < 1, 10 >$

Używając do tego funkcji: ylim (https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.ylim.html), xlim (https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.xlim.html), axis (https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.axis.html)

In [6]:
```
plt.figure(figsize=(12, 9))
x1 = np.linspace(-1, 1, 400)
y1 = f(x1)
plt.subplot(3, 1, 1)
plt.plot(x1, y1, label='f(x) = x^3 - 3x')
plt.title('Wykres funkcji w przedziale <-1,1>')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.xlim(-1, 1)
plt.ylim(-4, 4)

#################################################################
x2 = np.linspace(-10, -1, 400)
y2 = f(x2)

plt.subplot(3, 1, 2)
plt.plot(x2, y2, label='f(x) = x^3 - 3x')
plt.title('Wykres funkcji w przedziale <-10, -1>')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.xlim(-10, -1)
plt.ylim(-1100, 100)

#################################################################
x3 = np.linspace(1, 10, 400)
y3 = f(x3)

plt.subplot(3, 1, 3)
plt.plot(x3, y3, label='$f(x) = x^3 - 3x$')
plt.title('Wykres funkcji dla przedziału <1,10>')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.xlim(1, 10)
plt.ylim(-100, 1100)


plt.tight_layout()
plt.show()
```



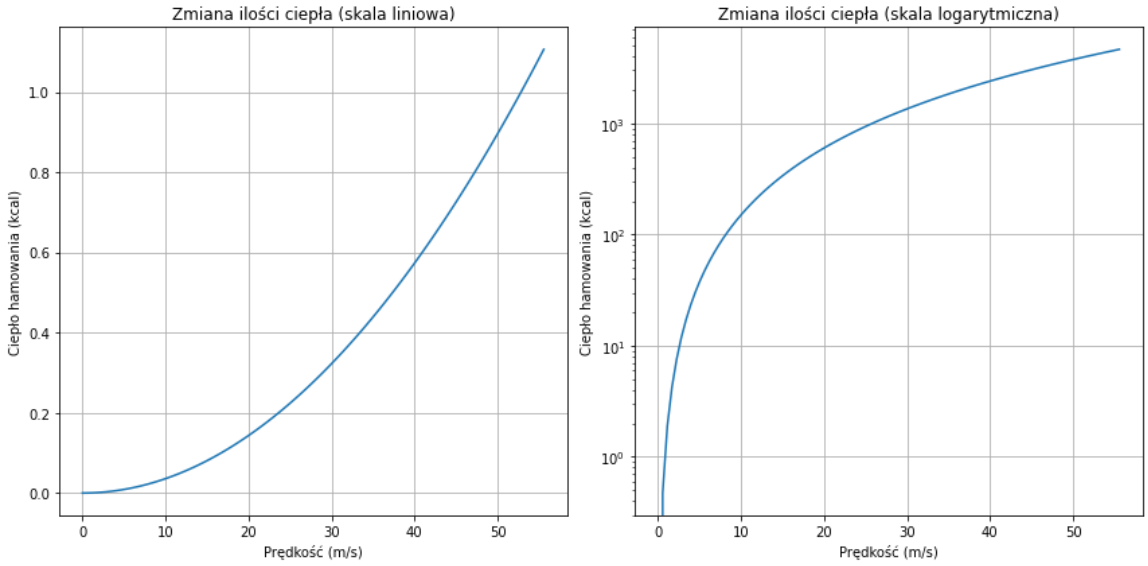

***Zadanie 3.*** Oblicz ilość ciepła, które wydzieli się podczas hamowania jeżeli opisuje je zależność:

$$Q = \frac{mv^2}{2}$$

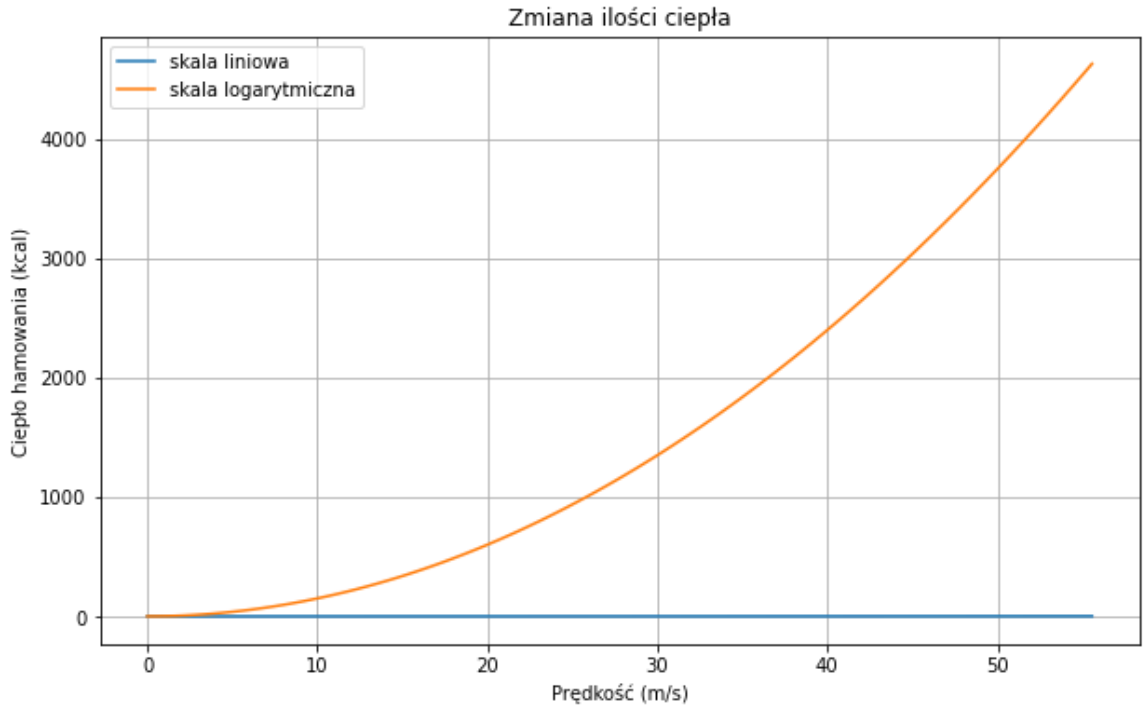
jeżeli $m = 2500g$, $v = 60km/h$, wypisz wynik oraz podaj w kilokaloriach i dżulach. Wykreśl wykresy które zobrazują zmianę ilości ciepła w procesie hamowania dla ciała o masie $3000g$ i prędkości $v \in < 200, 0 > km/h$, przedstawiając ciepło hamowania na pierwszym w skali liniowej na drugim w skali logarytmicznej.

***Wsakzówka:*** Do stworzenia wykresów z osią logarytmiczną skorzystaj z funkcji: semilogy (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.semilogy.html), xlim (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html?highlight=xlim#matplotlib.pyplot.xlim) i figure (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html?highlight=figure#matplotlib.pyplot.figure) z pakietu *Matplotlib*.

In [7]:
```
# Dane początkowe
m = 3.0 #masa w kilogramach
v_poczatkowe = 200.0
v_koncowe = 0.0

# Przeliczenie prędkości na m/s
v_poczatkowe = v_poczatkowe * 1000 / 3600
v_koncowe = v_koncowe * 1000 / 3600

# Zakres prędkości
v_wartosc = np.linspace(v_poczatkowe, v_koncowe, num=100)

# Obliczenie ciepła dla każdej prędkości
Q_wartosc = (m * v_wartosc**2) / 2

# Przeliczenie ciepła na kilokalorie
Q_wartosc_kcal = Q_wartosc / 4184

# Wykresy w skali liniowej i logarytmicznej
plt.figure(figsize=(12, 6))

# Wykres w skali liniowej
plt.subplot(1, 2, 1)
plt.plot(v_wartosc, Q_wartosc_kcal, label='Ciepło hamowania (kcal)')
plt.xlabel('Prędkość (m/s)')
plt.ylabel('Ciepło hamowania (kcal)')
plt.title('Zmiana ilości ciepła (skala liniowa)')
plt.grid()

# Wykres w skali logarytmicznej
plt.subplot(1, 2, 2)
plt.semilogy(v_wartosc, Q_wartosc, label='Ciepło hamowania (kcal)')
plt.xlabel('Prędkość (m/s)')
plt.ylabel('Ciepło hamowania (kcal)')
plt.title('Zmiana ilości ciepła (skala logarytmiczna)')
plt.grid()

plt.tight_layout()
plt.show()
```



In [8]:
```
# Dane początkowe
m = 3.0  # masa w kilogramach
v_poczatkowe = 200.0
v_koncowe = 0.0

# Przeliczenie prędkości na m/s
v_poczatkowe = v_poczatkowe * 1000 / 3600
v_koncowe = v_koncowe * 1000 / 3600

# Zakres prędkości
v_wartosc = np.linspace(v_poczatkowe, v_koncowe, num=100)

# Obliczenie ciepła dla każdej prędkości
Q_wartosc = (m * v_wartosc**2) / 2

# Przeliczenie ciepła na kilokalorie
Q_wartosc_kcal = Q_wartosc / 4184

compare_plot(v_wartosc, Q_wartosc_kcal, v_wartosc, Q_wartosc, 'Prędkość (m/s
             'Zmiana ilości ciepła', 'skala liniowa', 'skala logarytmiczna')
```

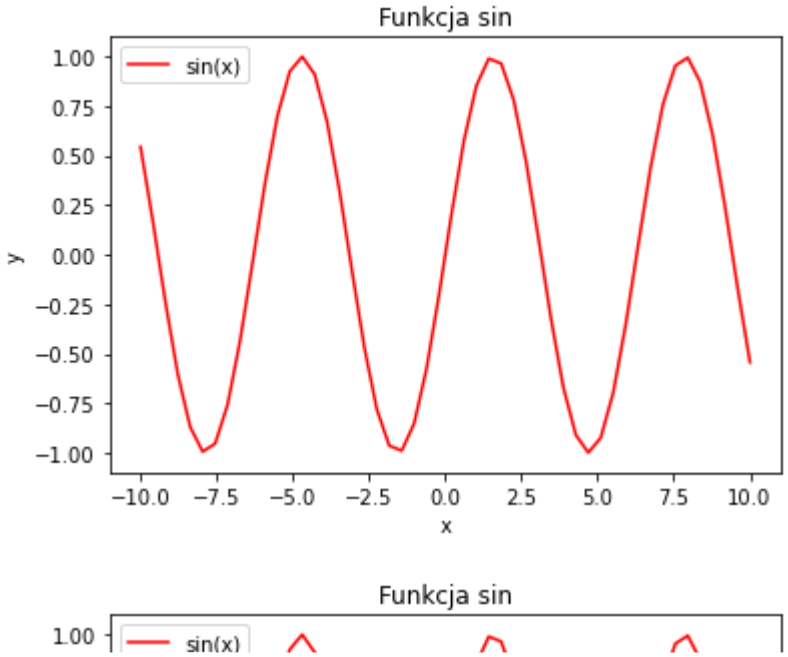



***Zadanie 4.***

Częstą praktyką szczególnie w dziedzinie analizy danych, statystyce, uczeniu maszynowym, itp. jest tworzenie na podstawie biblioteki Matplotlib (https://matplotlib.org/) własnych szablonów wykresów stworzonych na podstawie funkcji. Najprostszy przykład to przeładowanie funkcji plot tak by wykres miał czerwony kolor:

In [9]:
```
def my_plot(x,y,xlabel,ylabel,title,label):
    if x.shape != y.shape or  min(x.shape)==0:
        return None
    fig, ax = plt.subplots()
    ax.plot(x, y, 'r', label=label)
    ax.set(xlabel=xlabel, ylabel=ylabel, title=title)
    ax.legend()
    return fig

x = np.linspace(-10,10)
# inicjalizacja wektora f(x) = sin(x)
y = np.sin(x)
my_plot(x,y,'x','y','Funkcja sin', 'sin(x)')
```

Out[9]:





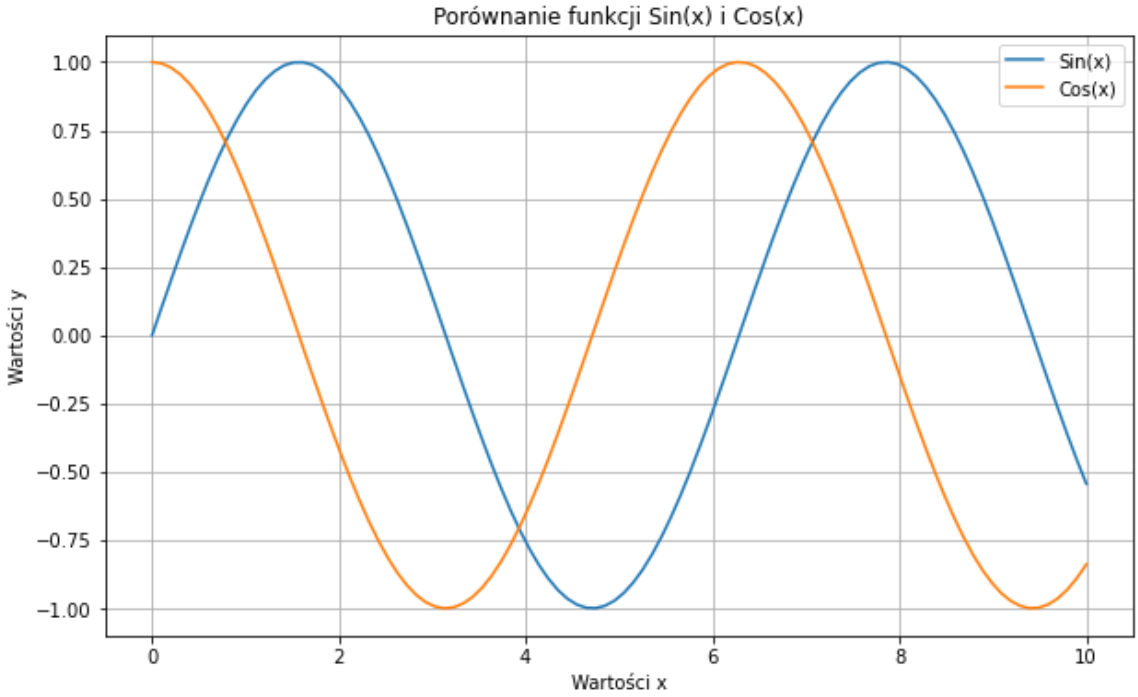


Zaimplementuj funkcję *compare_plot* w taki sposób by na jednym wykresie wyrysowane były dwie funkcje typu plot (https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot). Pierwsza ma rysować się w kolorze niebieski i grubość lini ma wynosić 4, druga natomiast w kolorze czerwonym o grubości lini 2. Domyślnie ma być dodawana legenda. Dodatkowo użytkownik ma mieć możliwość ustawienia parametrów:

- x1 - wektor wartości osi x dla pierwszego wykresu,
- y1 - wektor wartości osi y dla pierwszego wykresu,
- x2 - wektor wartości osi x dla drugiego wykresu,
- y2 - wektor wartości osi x dla drugiego wykresu,
- xlabel - opis osi x,
- ylabel - opis osi y,
- title - tytuł wykresu ,
- label1 - nazwa serii z pierwszego wykresu,
- label2 - nazwa serii z drugiego wykresu.

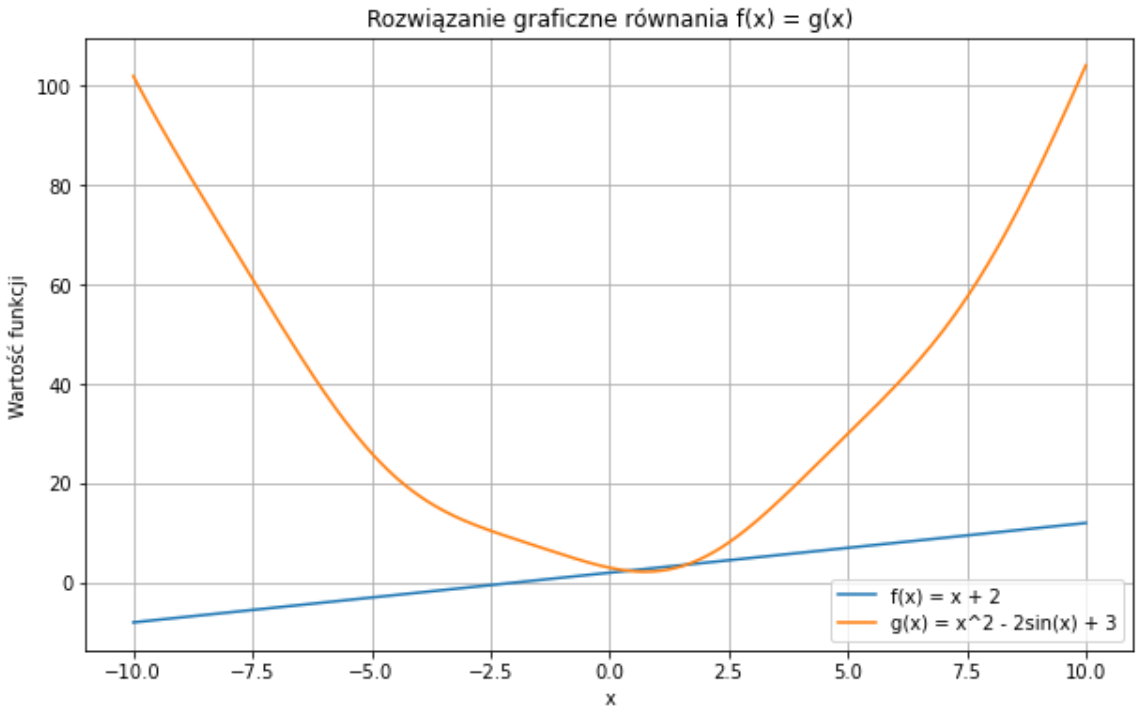
Jeżeli nie da się wyrysować danych należy zwrócić wartość None.

In [10]:
```
x1 = np.linspace(0, 10, 100)
y1 = np.sin(x1)
label1 = 'Sin(x)'

x2 = np.linspace(0, 10, 100)
y2 = np.cos(x2)
label2 = 'Cos(x)'

compare_plot(x1, y1, x2, y2, 'Wartości x', 'Wartości y', 'Porównanie funkcji
```




*** Zadanie 5 *** Za pomocą funkcji *compare_plot* rozwiąż graficznie równanie: $f(x) = g(x)$, gdzie:
$f(x) = x + 2$
$g(x) = x^2 - 2sin(x) + 3$

In [11]:
```
# Definicje funkcji f(x) i g(x)
def f(x):
    return x + 2

def g(x):
    return x**2 - 2 * np.sin(x) + 3

# Zakres wartości x
x = np.linspace(-10, 10, 500)

# Obliczenie wartości funkcji f(x) i g(x) na tym zakresie
y1 = f(x)
y2 = g(x)

compare_plot(x, y1, x, y2, 'x', 'Wartość funkcji', 'Rozwiązanie graficzne ró
```



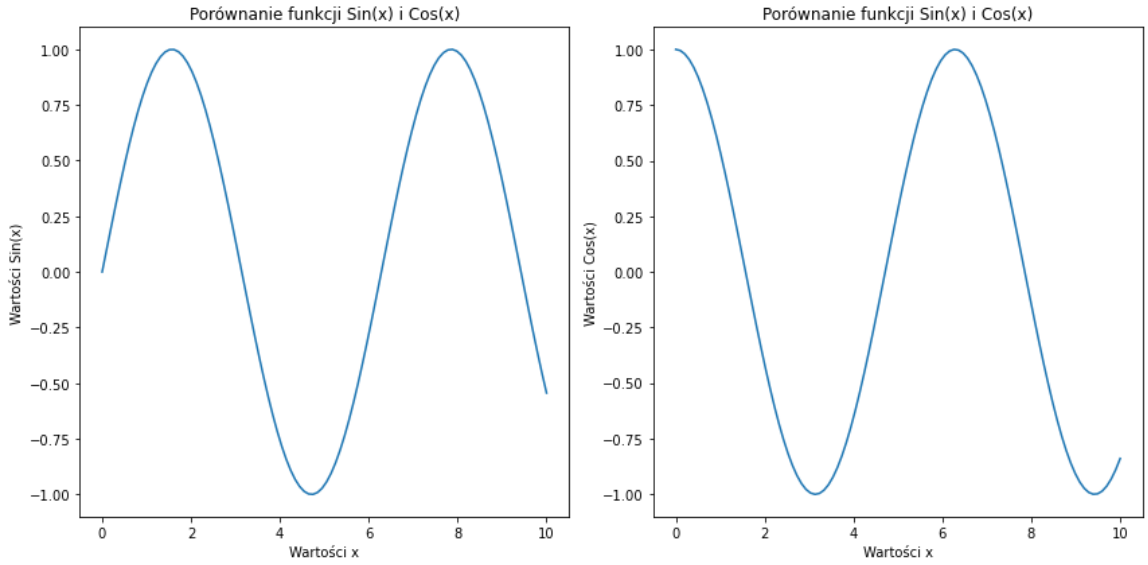

***Zadanie 6.***

Innym przydatnym sposobem prezentowania wykresów jest subplot (https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.subplot.html), czyli wyrysowanie obok siebie kilku wykresów. Zaimplementuj funkcję parallel_plot tak by obok siebie wyrysowane zostały dwa standardowe ploty i użytkownik mógł podać parametry:

- x1 - wektor wartości osi x dla pierwszego wykresu,
- y1 - wektor wartości osi y dla pierwszego wykresu,
- x2 - wektor wartości osi x dla drugiego wykresu,
- y2 - wektor wartości osi x dla drugiego wykresu,
- x1label - opis osi x dla pierwszego wykresu,
- y1label - opis osi y dla pierwszego wykresu,
- x2label - opis osi x dla drugiego wykresu,
- y2label - opis osi y dla drugiego wykresu,
- title - tytuł wykresu,
- orientation - parametr przyjmujący wartość '-' jeżeli subplot ma posiadać dwa wiersze albo '|' jeżeli ma posiadać dwie kolumny.

Jeżeli nie da się wyrysować danych należy zwrócić wartość None

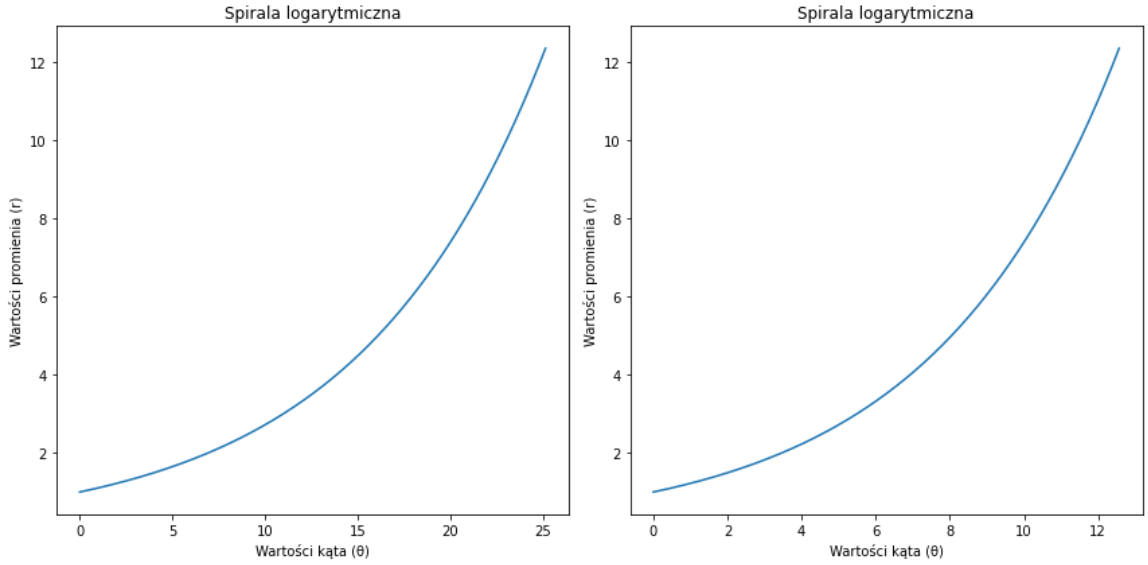
In [12]:
```
# Przykład użycia funkcji parallel_plot
x1 = np.linspace(0, 10, 100)
y1 = np.sin(x1)
x1label = 'Wartości x'
y1label = 'Wartości Sin(x)'

x2 = np.linspace(0, 10, 100)
y2 = np.cos(x2)
x2label = 'Wartości x'
y2label = 'Wartości Cos(x)'

title = 'Porównanie funkcji Sin(x) i Cos(x)'
orientation = '-'

parallel_plot(x1, y1, x2, y2, x1label, y1label, x2label, y2label, title, ori
```




***Zadanie 7.***

Za pomocą funkcji *parallel_plot* i przedstaw na jednym z nich Spirale logarytmiczną (https://pl.wikipedia.org/wiki/Spirala_logarytmiczna) w szerokim przedziale, a w drugim w okolicy zera.

In [13]:
```
# Spirala logarytmiczna w szerokim przedziale
theta1 = np.linspace(0, 8 * np.pi, 1000)
r1 = np.exp(0.1 * theta1)

# Spirala logarytmiczna w okolicy zera
theta2 = np.linspace(0, 4 * np.pi, 1000)
r2 = np.exp(0.2 * theta2)

x1label = 'Wartości kąta (θ)'
y1label = 'Wartości promienia (r)'

x2label = 'Wartości kąta (θ)'
y2label = 'Wartości promienia (r)'

title = 'Spirala logarytmiczna'

orientation = '-'

parallel_plot(theta1, r1, theta2, r2, x1label, y1label, x2label, y2label, ti
```




***Zadanie 8.***

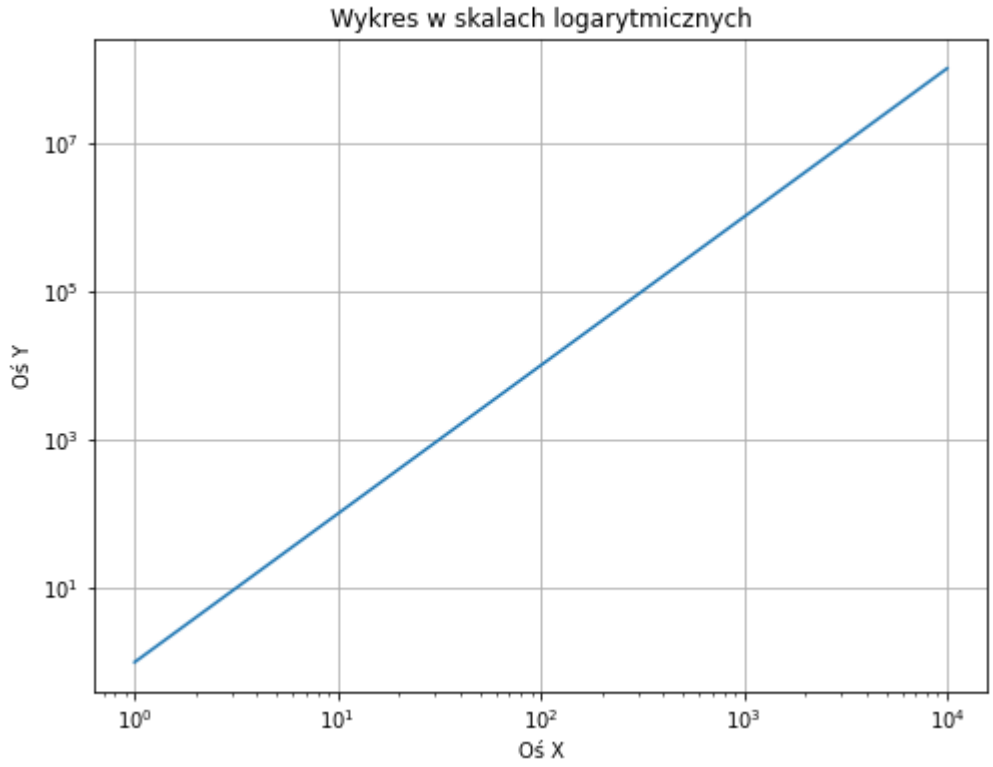
Zaimplementuj funkcję *log_plot* która będzie tworzyć wykres w skalach logaritmicznych. Skale logarytmiczne mają być ustawione zgodnie z parameterem *log_axis* gdzie wartość:

- 'x' oznacza skale logarytmiczną na osi x,
- 'y' oznacza skale logarytmiczną na osi y,
- 'xy' oznacza skale logarytmiczną na obu osiach.

Dodatkowo użytkownik ma mieć możliwość ustawienia parametrów:

- x - wektor wartości osi x,
- y - wektor wartości osi y,
- xlabel - opis osi x,
- ylabel - opis osi y,
- title - tytuł wykresu.

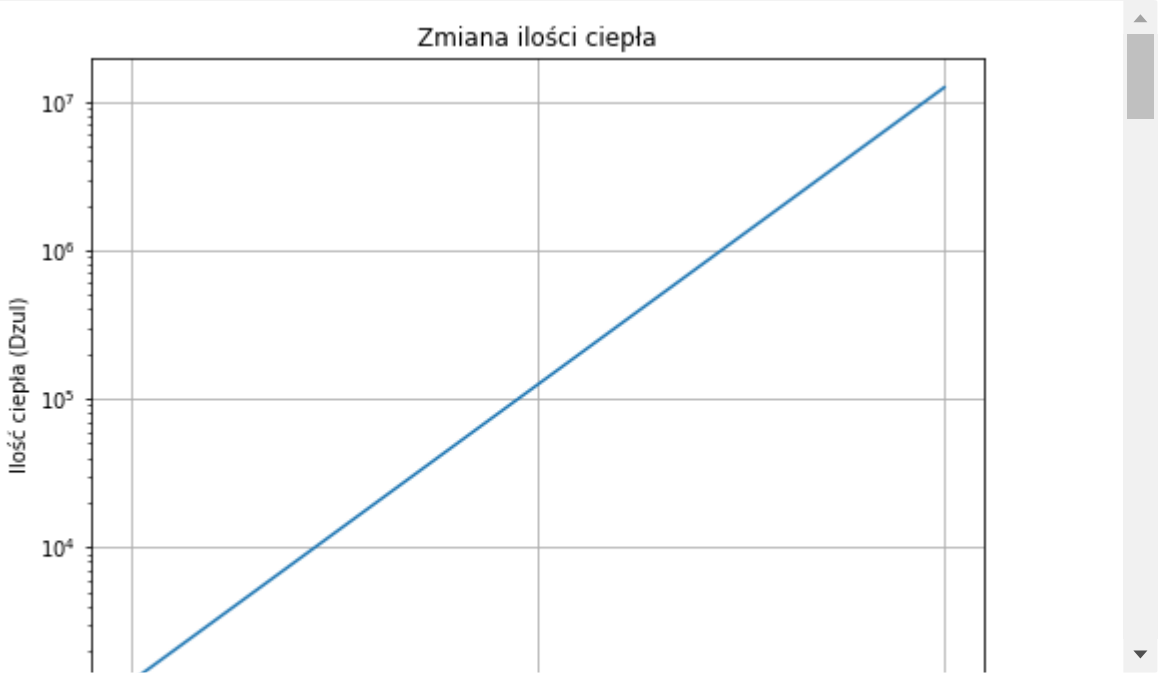
In [14]:
```
# Przykład użycia funkcji log_plot
x = [1, 10, 100, 1000, 10000]
y = [1, 100, 10000, 1000000, 100000000]

xlabel = 'Oś X'
ylabel = 'Oś Y'
title = 'Wykres w skalach logarytmicznych'
log_axis = 'xy'  # Skala logarytmiczna na obu osiach

log_plot(x, y, xlabel, ylabel, title, log_axis)
```




**Zadanie 9.**

Przy pomocy funkcji *log_plot* przedstaw różne warianty funkcji z zadania 3.

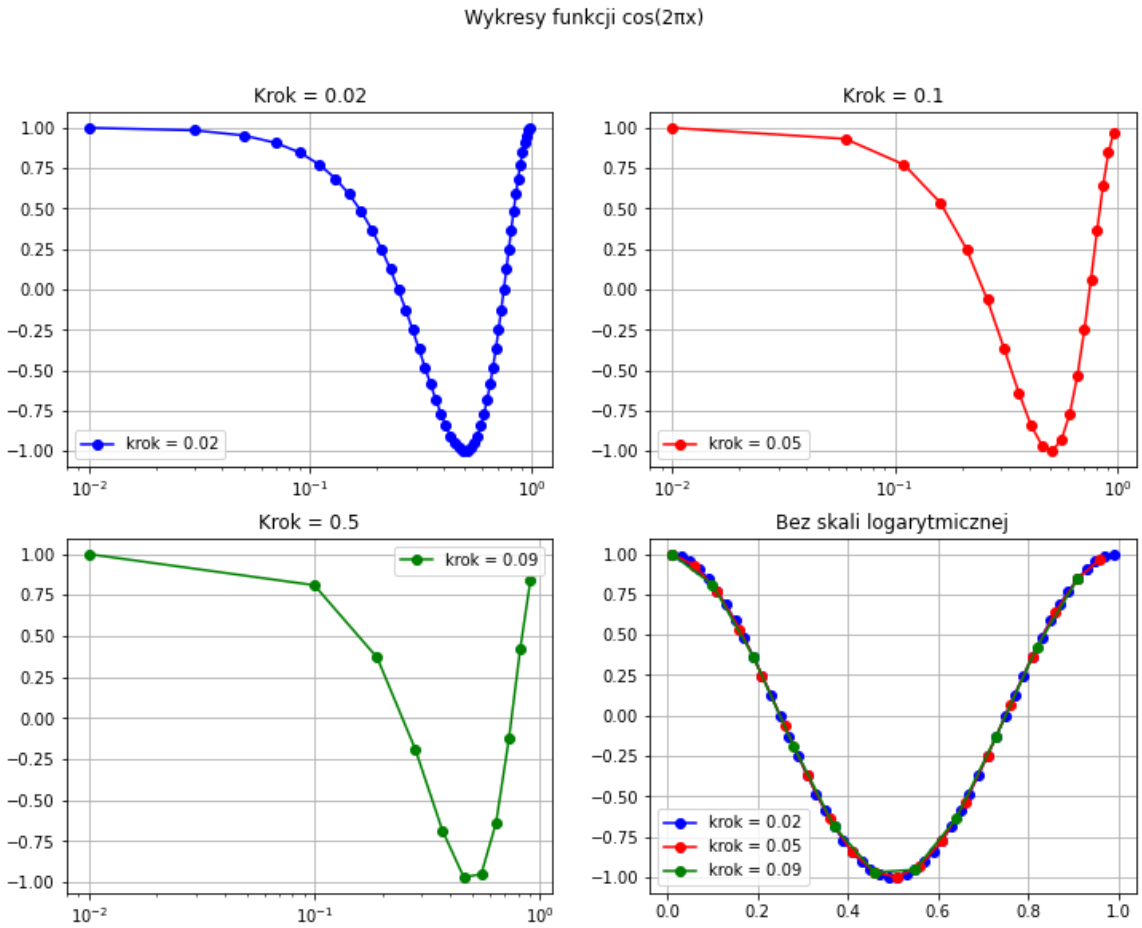
In [15]:
```
# Warianty funkcji z zadania 3
m = 2500  # masa w gramach
v = np.linspace(1, 100, 100)  # prędkości od 1 do 100

# Obliczenie ilości ciepła
Q = (m * v**2) / 2
Q_kcal = Q / 4184  # przeliczenie na kilokalorie

# Wykresy w różnych wariantach
log_plot(v, Q, 'Prędkość (m/s)', 'Ilość ciepła (Dzul)', 'Zmiana ilości ciepł
log_plot(v, Q_kcal, 'Prędkość (m/s)', 'Ilość ciepła (Dzul)', 'Zmiana ilości
log_plot(v, Q, 'Prędkość (m/s)', 'Ilość ciepła (Dzul)', 'Zmiana ilości ciepł
log_plot(v, Q_kcal, 'Prędkość (m/s)', 'Ilość ciepła (kcal)', 'Zmiana ilości
log_plot(v, Q, 'Prędkość (m/s)', 'Ilość ciepła (Dzul)', 'Zmiana ilości ciepł
log_plot(v, Q_kcal, 'Prędkość (m/s)', 'Ilość ciepła (kcal)', 'Zmiana ilości
```




**Zadanie 10.**

Wykorzystując funkcje *semilogx* narysuj funkcję $cos(2\pi x)$ w zakresie co najmniej od $-10^2$ do $10^0$. Uzyj do tego trzech róznych wartości kroków, gdzie jeden będzie równy około 0.02, a reszta nieznacznie większa, dla generowania wartości funkcji (za pomocą *np.arrange*), a następnie narysuj wykresy według schematu: węzły (punkty, w których znamy wartosc funckji) zaznacz pomoca kropek, połączonych liniami przerywanymi. Zauwaz jak wykresy zachowują się w miarę zwiększania kroku. Dla porównania narysuj takze wykres niestosujac skali logarytmicznej. Wykres umieść w siatce 2x2 wykorzystując *subplot*. Nie zapomnij o oznaczeniu tytułów osi i wykresów.

In [16]:
```
# Zakres wartości x
x_range = np.array([10**-2, 1])

# Trzy różne wartości kroku
step1 = 0.02
step2 = 0.05
step3 = 0.09

# Generowanie wartości x
x1 = np.arange(x_range[0], x_range[1], step1)
x2 = np.arange(x_range[0], x_range[1], step2)
x3 = np.arange(x_range[0], x_range[1], step3)

# Obliczanie wartości funkcji
y1 = np.cos(2 * np.pi * x1)
y2 = np.cos(2 * np.pi * x2)
y3 = np.cos(2 * np.pi * x3)

# Wykresy na skali logarytmicznej
plt.figure(figsize=(12, 9))

# Wykres 1
plt.subplot(221)
plt.semilogx(x1, y1, 'bo-', label='krok = 0.02')
plt.title('Krok = 0.02')
plt.grid()
plt.legend()

# Wykres 2
plt.subplot(222)
plt.semilogx(x2, y2, 'ro-', label='krok = 0.05')
plt.title('Krok = 0.1')
plt.grid()
plt.legend()

# Wykres 3
plt.subplot(223)
plt.semilogx(x3, y3, 'go-', label='krok = 0.09')
plt.title('Krok = 0.5')
plt.grid()
plt.legend()

# Wykres bez skali logarytmicznej
plt.subplot(224)
plt.plot(x1, y1, 'bo-', label='krok = 0.02')
plt.plot(x2, y2, 'ro-', label='krok = 0.05')
plt.plot(x3, y3, 'go-', label='krok = 0.09')
plt.title('Bez skali logarytmicznej')
plt.grid()
plt.legend()

plt.suptitle('Wykresy funkcji cos(2πx)')
plt.show()
```




Materiały uzupełniające:

- Scipy Lecture Notes (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)
- Python Tutorial - W3Schools (https://www.w3schools.com/python/default.asp)
- NumPy (https://www.numpy.org)
- Matplotlib (https://matplotlib.org/)
- Anaconda (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
- Learn Python (https://www.learnpython.org/)
- Wujek Google (https://google.pl) i Ciocia Wikipedia (https://pl.wikipedia.org/wiki/Wikipedia:Strona_g%C5%82%C3%B3wna)

In [ ]: