Szeregi potęgowe.

Jak się okazuje, własności różniczkowania i całkowania granicy ciagu czy sumy szeregu funkcyjnego "wyraz po wyrazie" jest bardzo ważna w zastosowaniach (o czym wkrótce). To oznacza, że pewna klasa szeregów jest szczególnie użyteczna, bo będzie można je różniczkować i całkować bez dodatkowych założeń. To **szeregi potęgowe**.

Powody zainteresowania nimi w informatyce są jeszcze co najmniej dwa: ich sumy częściowe to wielomiany, oraz obejmują klasę szeregów Taylora, które już poznaliśmy, a które pozwalały np. na dobrą analizę błędów obliczeniowych.

> **Szeregiem potęgowym** nazywamy szereg funkcyjny postaci
>
> $$S(x) = \sum_{n=0}^{\infty} a_n \cdot (x - x_0)^n,$$

gdzie (a_n) jest ciągiem liczbowym. Przykład już znamy: $\sum_{n=0}^{\infty} \frac{x^n}{n!}$. Jest to szereg zbieżny bezwzględnie niemal jednostajnie na $\mathbb{R}$ do funkcji e^x. Jest niemal jednostajnie zbieżny, ale nie jest zbieżny jednostajnie na $\mathbb{R}$.

Inny klasyczny przykład to szereg geometryczny z ilorazem x.

Okazuje się (twierdzenie Cauchy'ego-Hadamarda), że szeregi potęgowe zawsze są zbieżne niemal jednostajnie w symetrycznym przedziale postaci $(x_0 - r, x_0 + r)$, gdzie r nazywamy promieniem zbieżności i pozwala się on wyliczyć z ciągu współczynników (a_n). Zaczniemy od wzorów na promień zbieżności.

Definicja.

> Wartość (liczbę nieujemną lub ∞) zadaną wzorem
>
> $$r = \frac{1}{\limsup_n \sqrt[n]{|a_n|}}$$

> nazywamy *promieniem zbieżności* szeregu potęgowego $\sum_{n=0}^{\infty} a_n (x - x_0)^n$.

Uwaga: dla funkcji o wartościach zespolonych mamy zbiór na płaszczyźnie zespolonej $\{z : |z - z_0| < r\}$ i to tłumaczy nazwę "koło zbieżności" ...

> **Promieniem zbieżności** szeregu potęgowego $\sum_{n=0}^{\infty} a_n (x - x_0)^n$ nazywamy kres górny r zbioru tych $|x - x_0|$, dla których szereg $\sum_{n=0}^{\infty} a_n (x - x_0)^n$ jest zbieżny.

Gdy zbiór ten jest nieograniczony, umawiamy się przyjąć $r = \infty$. Przedział $(x_0 - r, x_0 + r)$ (koło $|x - x_0| < r$) nazywamy przedziałem zbieżności (kołem zbieżności) szeregu potęgowego o promieniu zbieżności r.

Twierdzenie Cauchy'ego - Hadamarda podaje wzór na promień zbieżności.

> Twierdzenie.
>
> Niech $\lambda = \overline{\lim_{n\to\infty}} \sqrt[n]{|a_n|}$. Wtedy promieniem zbieżności szeregu potęgowego $\sum_{n=0}^{\infty} a_n(x-x_0)^n$ jest $r = \frac{1}{\lambda}$, przy czym gdy $\lambda = 0$ przyjmujemy $r = \infty$, a gdy $\lambda = \infty$, przyjmujemy $r = 0$.

Te ostatnie zastosowanie ma swoją podstawę w obserwacji, że znane nam funkcje (elementarne) mają swoje reprezentacje w postaci szeregów potęgowych i wygodnie jest zastępować je szeregami (co pozwala oszacować z dowolną dokładnością ich wartości!!).

Można m.in. wykazać, że (wzór Taylora!! - samodzielnie (lub na ćwiczeniach) sprawdzić!):

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$

Przypomnijmy: to są tak naprawdę szeregi Taylora tych funkcji. Ale także trzeba przypomnieć: one dobrze przybliżają funkcje jedynie LOKALNIE.

A do czego jescze mogą się przydać? Np. do kontroli znaku wartości tych funkcji:

$$\cos 2 \leq 1 - \frac{2^2}{2!} + \frac{2^4}{4!} = 1 - 2 + \frac{2}{3} < 0.$$

Inne przykłady - samodzielnie lub na ćwiczeniach. A na *laboratorium* (zastanówmy się jak to zrobić na ćwiczeniach):

> Sprawdzić znak $\sin 132567$...

Szczególny przypadek - szeregi potęgowe.

> Wniosek.
>
> Dany jest szereg potęgowy $\sum_{n=0}^{\infty} a_n(x-x_0)^n$. Niech r oznacza jego promień zbieżności i niech $r > 0$.
>
> Wtedy w przedziale zbieżności $(x_0 - r, x_0 + r)$ można ten szereg zarówno całkować, jak i różniczkować, wyraz po wyrazie.

Kryterium d'Alemberta...

Można też zbadać zbieżność szeregu potęgowego za pomocą kryterium ilorazowego, czyli sprawdzenia, czy stosunek dwóch kolejnych wyrazów w szeregu dąży do stałej wartości. W Pythonie można to zrobić np. w następujący sposób:

In [12]:
```python
import sympy as sp

def ratio_test(formula):
    """
    Test ilorazu dla szeregu a na podstawie wzoru ogólnego.
    Zwraca True jeśli szereg jest zbieżny, False jeśli jest rozbieżny, None jeśli n
    """
    n = sp.symbols('n')
    an = sp.sympify(formula)

    limit = sp.limit(an/(an.subs(n, n+1)), n, sp.oo)

    if sp.limit(an/(an.subs(n, n+1)), n, sp.oo) == 1:
        return None, "nie można jednoznacznie określić"
    elif limit < 1:
        return True, "zbieżny"
    else:
        return False, "rozbieżny"

formula = input("Podaj wzór ogólny dla a_n: ")
convergence, result = ratio_test(formula)
print(f"Szereg jest {result}.")


# Wyświetlenie granicy ilorazu
print("Granica ilorazu dla n -> oo:", sp.limit(sp.sympify(formula)/(sp.sympify(form
print(f"Jak widać wyniki bywają błędne...")
```

```
Szereg jest rozbieżny.
Granica ilorazu dla n -> oo: oo
Jak widać wyniki bywają błędne...
```

Rodzaje zbieżności szeregu potęgowego.

> **Twierdzenie.** Jeśli $0 < r < \infty$, to szereg potęgowy $\sum_{n=0}^{\infty} a_n(x - x_0)^n$ jest bezwzględnie niemal jednostajnie zbieżny w przedziale $(x_0 - r, x_0 + r)$.

> Natomiast w każdym punkcie zbioru $(-\infty, x_0 - r) \cup (x_0 + r, \infty)$ jest on rozbieżny. Na końcach przedziału (czyli dla $x = x_0 - r$ i $x = x_0 + r$) zbieżność może, ale nie musi zachodzić (może też zachodzić w tylko jednym z końców).
>
> Jeśli $r = \infty$ to szereg jest niemal jednostajnie zbieżny na całym $\mathbb{R}$.
>
> Jeśli zaś $r = 0$, to szereg jest rozbieżny w każdym punkcie zbioru $(-\infty, x_0) \cup (x_0, \infty)$, tu jednak mamy: każdy szereg potęgowy jest zbieżny (i ma sumę a_0) w punkcie x_0.

Zastosowania w informatyce.

Szereg potęgowy to narzędzie matematyczne o wielu zastosowaniach w informatyce. Oto kilka przykładów:

1. Aproksymacja funkcji: Szeregi potęgowe mogą być stosowane do aproksymacji funkcji z pewnej klasy funkcji analitycznych wokół danego punktu. Ta metoda jest szczególnie

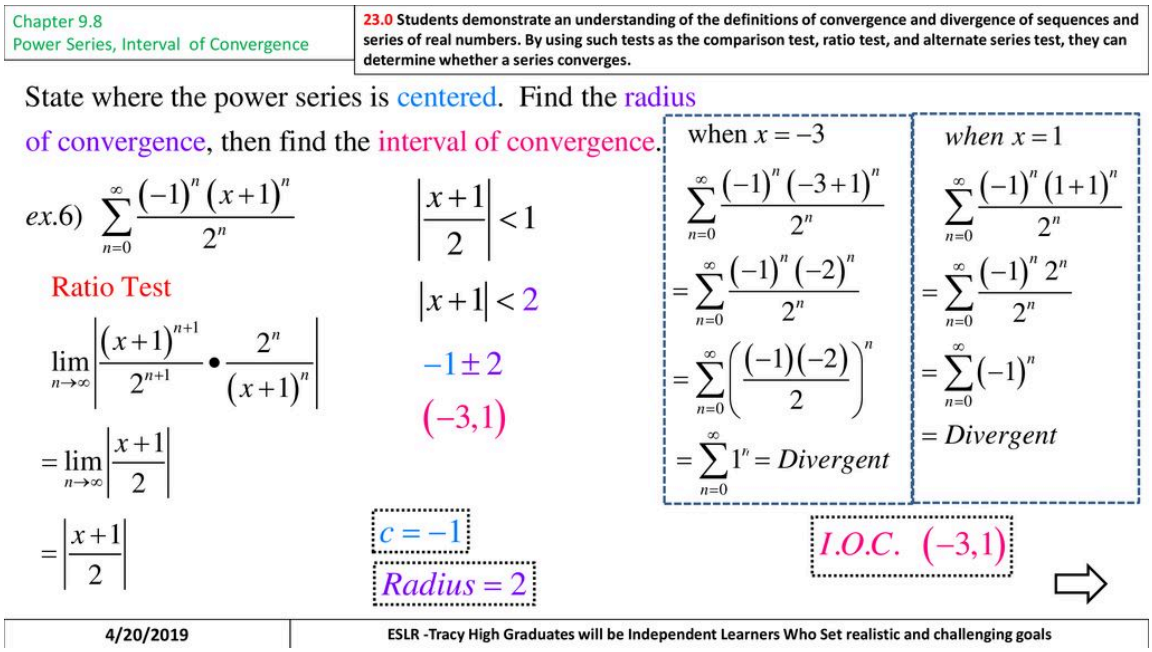
przydatna, gdy trudno jest obliczyć wartości dokładne funkcji w danym punkcie, ale można łatwo obliczyć jej wartości dla punktów wokół niego. Przykłady funkcji aproksymowanych w ten sposób to funkcje trygonometryczne, funkcje wykładnicze itp.

2. Obliczenia numeryczne: Szeregi potęgowe mogą być używane do wyznaczania wartości funkcji, które są trudne lub niemożliwe do obliczenia dokładnie w sposób analityczny. Przykłady takich funkcji to funkcje trygonometryczne, logarytmy, funkcje Bessela, funkcje Gamma i wiele innych. W przypadku funkcji, dla których nie istnieją analityczne wzory, szeregi potęgowe mogą być stosowane do obliczania ich wartości w pewnych punktach.

3. Symulacje numeryczne: Szeregi potęgowe mogą być stosowane do generowania ciągów liczb losowych o określonej rozkładzie prawdopodobieństwa. Zwykle wymaga to aproksymacji funkcji rozkładu prawdopodobieństwa za pomocą szeregu potęgowego i generowania liczb losowych poprzez odwrócenie tej funkcji. Przykładem takiego zastosowania może być symulacja Monte Carlo, która jest szeroko stosowana w różnych dziedzinach nauki, takich jak fizyka, chemia, inżynieria i ekonomia.

4. Optymalizacja: Szeregi potęgowe mogą być stosowane do optymalizacji funkcji. Istnieją różne metody optymalizacji, które wykorzystują szeregi potęgowe, takie jak metoda Newtona-Raphsona, metoda quasi-Newtona i metoda siecznych. Te metody mogą być stosowane do rozwiązywania problemów optymalizacji w różnych dziedzinach, takich jak projektowanie algorytmów, optymalizacja kosztów w produkcji i wielu innych.

> Twierdzenie.
>
> Jeśli istnieje granica (właściwa lub nie) $\lim_n \left|\frac{a_n}{a_{n+1}}\right|$, to jest ona równa promieniowi zbieżności.



In [14]:
```
import sympy as sp
from sympy import Sum, oo

x = sp.symbols('x')
```

```
# Definicja szeregu
n = sp.symbols('n', integer=True)  # Definicja symbolu całkowitoliczbowego
series = Sum((-1)**(n+1)/n * x**n, (n, 1, oo))

# Wyświetlenie sumy częściowej dla n = 5
print(series.subs(n, 5).doit())

# Sprawdzenie zbieżności dla x=1
if series.subs(x, 1).doit().is_finite:
    print("Szereg jest zbieżny dla x=1")
else:
    print("Szereg jest rozbieżny dla x=1")

# Sprawdzenie zbieżności dla x=-1
if series.subs(x, -1).doit().is_finite:
    print("Szereg jest zbieżny dla x=-1")
else:
    print("Szereg jest rozbieżny dla x=-1")
```

```
-Piecewise((-log(x + 1), (x <= 1) & (x > -1)), (Sum((-1)**n*x**n/n, (n, 1, oo)), T
rue))
Szereg jest zbieżny dla x=1
Szereg jest rozbieżny dla x=-1
```

Funkcje zespolone.

Przypomnienie: szeregi potęgowe zespolone, są to funkcje argumentu zespolonego postaci $\sum_{n=0}^{\infty} a_n z^n$.

Promień zbieżności r, gdzie $\frac{1}{r} = \limsup |a_n|^n$.Wówczas $\sum_{n=0}^{\infty} a_n z^n$ jest zbieżna dla wszystkich $z : |z| < r$. Co więcej szereg funkcyjny $\sum_{n=0}^{\infty} a_n z^n$ jest zbieżny niemal jednostajnie na $K(0, r)$.

Różniczkując $\sum_{n=0}^{\infty} a_n z^n$ wyraz po wyrazie dostajemy szereg potęgowy

$$\sum_{n=0}^{\infty} n a_n z^{n-1}.$$

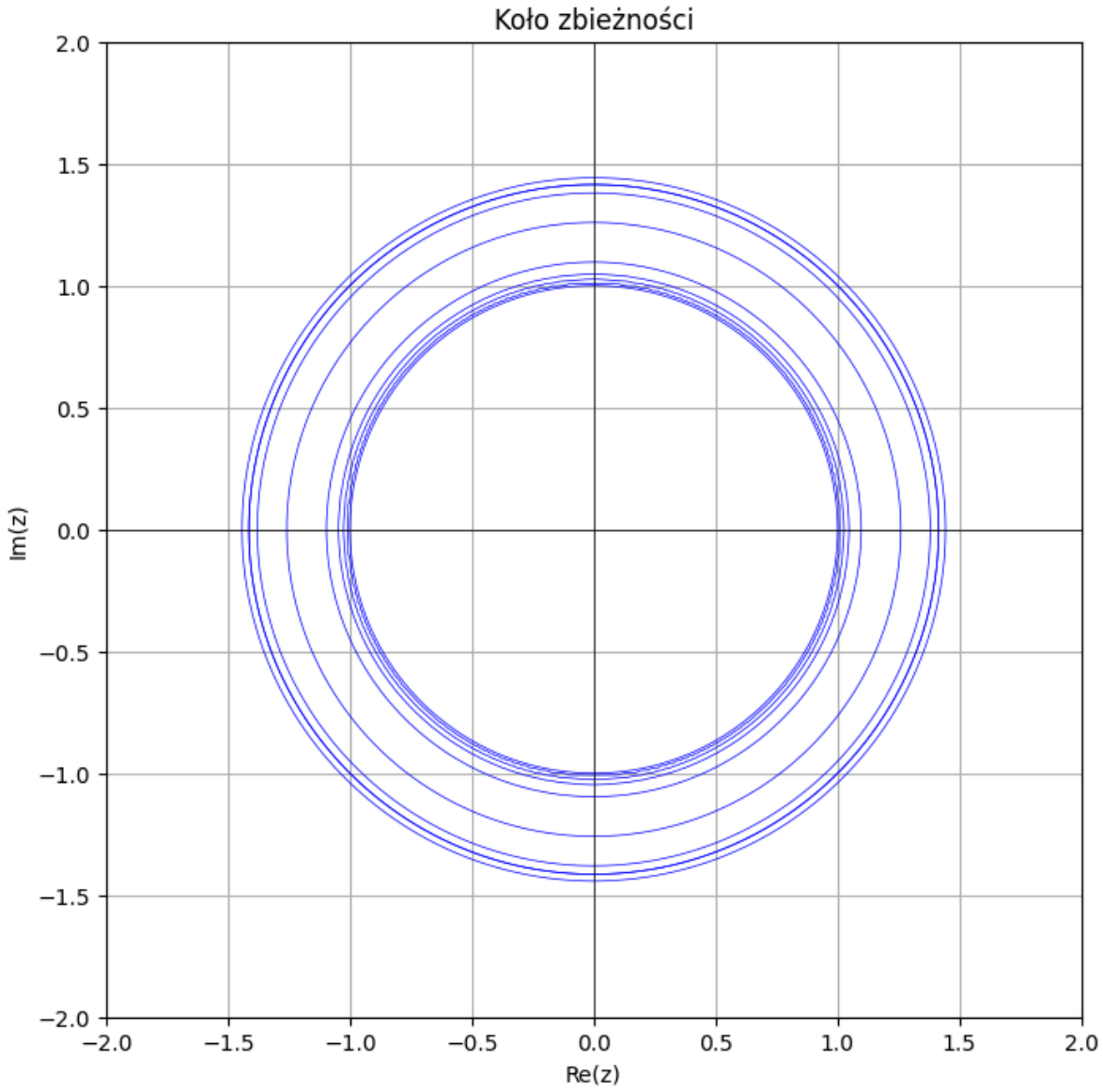
Zróżniczkowany szereg potęgowy ma taki sam promień zbieżności jak $\sum_{n=0}^{\infty} a_n z^n\}$, gdyż $\sqrt[n]{n|a_n|} = \sqrt[n]{n} \cdot \sqrt[n]{|a_n|}$ oraz $\lim_{n\to\infty} \sqrt[n]{n} = 1$.

Ciągi funkcyjne zespolone.

Wizualizacja przykładu dla $f_n(z) = \frac{z^n}{n}$.

In [22]:
```
import numpy as np
import matplotlib.pyplot as plt

def fn(z, n):
    """Funkcja zdefiniowana jako f_n(z) = z^n / n"""
    return z**n / n

def convergence_circle(radius, n_values):
    """Generowanie koła zbieżności dla danego promienia i wartości n"""
    x = np.linspace(-radius, radius, 400)
    y = np.linspace(-radius, radius, 400)
```

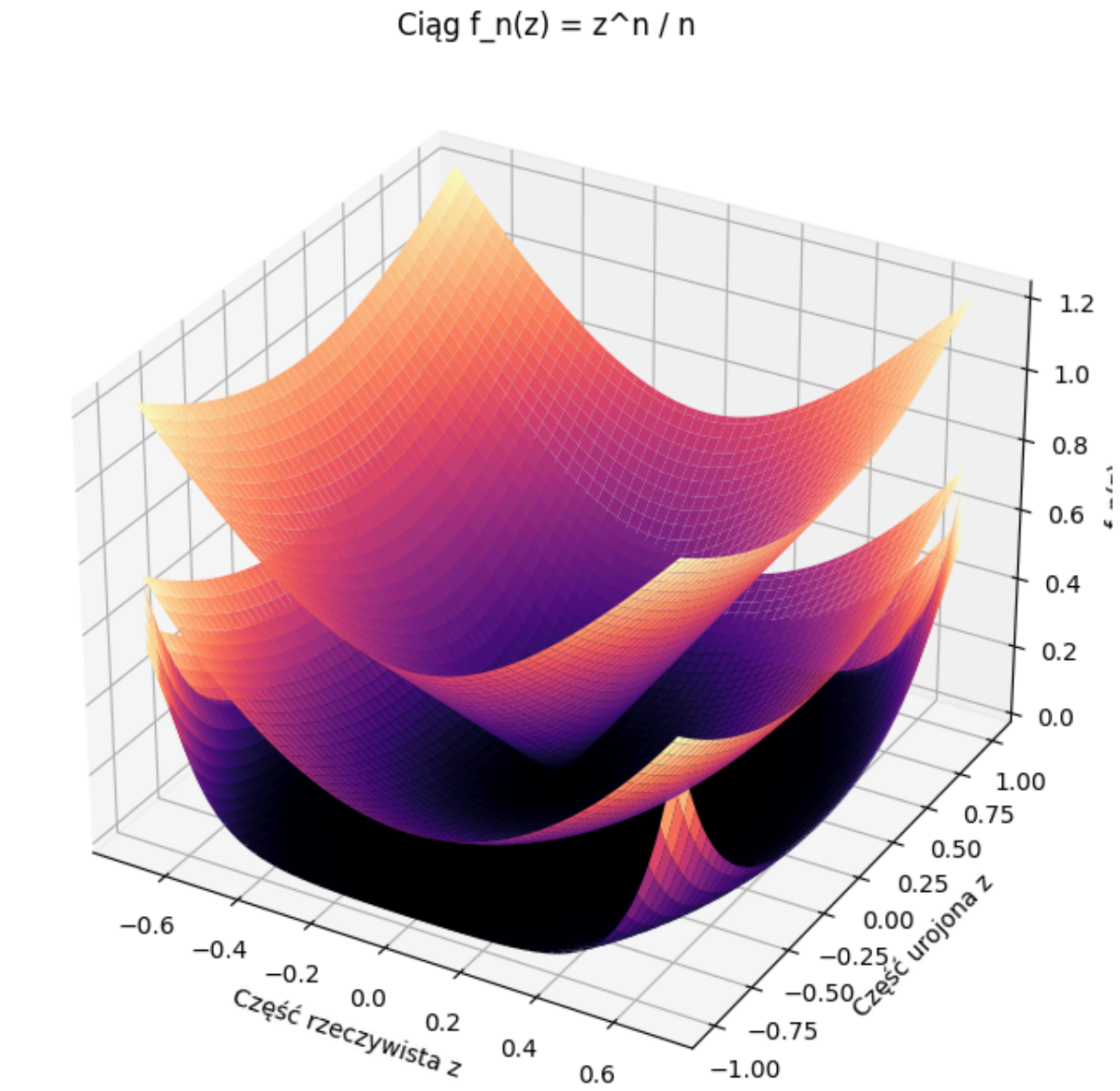
```
    X, Y = np.meshgrid(x, y)
    Z = X + 1j*Y

    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_aspect('equal', 'box')

    for n in n_values:
        F = fn(Z, n)
        ax.contour(X, Y, np.abs(F), levels=[1], colors='b', linewidths=0.5)
        #ax.text(0.1, 0.1, f'n = {n}', fontsize=10, transform=ax.transAxes)

    ax.set_xlim(-radius, radius)
    ax.set_ylim(-radius, radius)
    ax.axhline(0, color='black',linewidth=0.5)
    ax.axvline(0, color='black',linewidth=0.5)
    ax.set_xlabel('Re(z)')
    ax.set_ylabel('Im(z)')
    ax.set_title('Koło zbieżności')
    plt.grid()
    plt.show()

# Definicja promienia i wartości n
radius = 2
n_values = [1, 2, 3, 4, 5, 10, 40, 100, 200, 500]

# Wywołanie funkcji generującej wizualizację
convergence_circle(radius, n_values)
```



In [23]:

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Funkcja zwracająca wartości ciągu f_n(z) dla danego z
def fn(z, n):
    return z ** n / n

# Wygeneruj zestaw wartości x, y i z
x = np.linspace(-0.7, 0.7, 100)
y = np.linspace(-1, 1, 100)
X, Y = np.meshgrid(x, y)
Z = np.zeros((len(x), len(y), 10))
for i in range(len(x)):
    for j in range(len(y)):
        z = x[i] + 1j * y[j]
        for n in range(1, 11):
            Z[i, j, n-1] = np.abs(fn(z, n))

# Stwórz wykres funkcji sin(z) w przestrzeni 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

#ax = fig.gca(projection='3d')
for n in range(10):
```

```
    surf = ax.plot_surface(X, Y, Z[:, :, n], cmap='magma')
ax.set_title('Ciąg f_n(z) = z^n / n')
ax.set_xlabel('Część rzeczywista z')
ax.set_ylabel('Część urojona z')
ax.set_zlabel('f_n(z)')
plt.show()
```



Zespolone szeregi potęgowe.

Szczególnie ciekawe wyniki da się uzyskać korzystając z zespolonych szeregów potęgowych.

1. Szereg $\sum_{n=0}^{\infty} z^n$ ma promień zbieżności równy 1. Gdy $|z| = 1$ jest rozbieżny (gdyż nie spełnia warunku koniecznego zbieżności szeregu).

2. Do szeregu $\sum_{n=0}^{\infty} \frac{z^n}{n}$ możemy zastosować także kryterium ilorazowe:

$$\frac{z^{n+1}/(n+1)}{z^n/n} = z \cdot \frac{n}{n+1} \to z.$$

Gdy $z = 1$ to jest rozbieżny, ale gdy $|z| = 1$ and $z \neq 1$ jest zbieżny (z kryterium Abela).

3. Szereg $\sum_{n=1}^{\infty} \frac{z^n}{n^2}$ jest zbieżny dla $|z| \leq 1$ i rozbieżny dla $|z| > 1$. Co jeśli $|z| = 1$?

4. Promień zbieżności szeregu $\frac{z^n}{2^n}$ wynosi 2, gdyż $\sqrt[n]{|a_n|} = \frac{1}{2}$ dla każdego n. Stąd $\limsup \sqrt[n]{|a_n|} = \frac{1}{2}$, czyli $1/\limsup \sqrt[n]{|a_n|} = 2$.

Klasyczne funkcje zespolone.

Możemy poprawnie zdefiniować rozszerzenie funkcji wykładniczej e^x do dziedziny zespolonej:

$$e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!}.$$

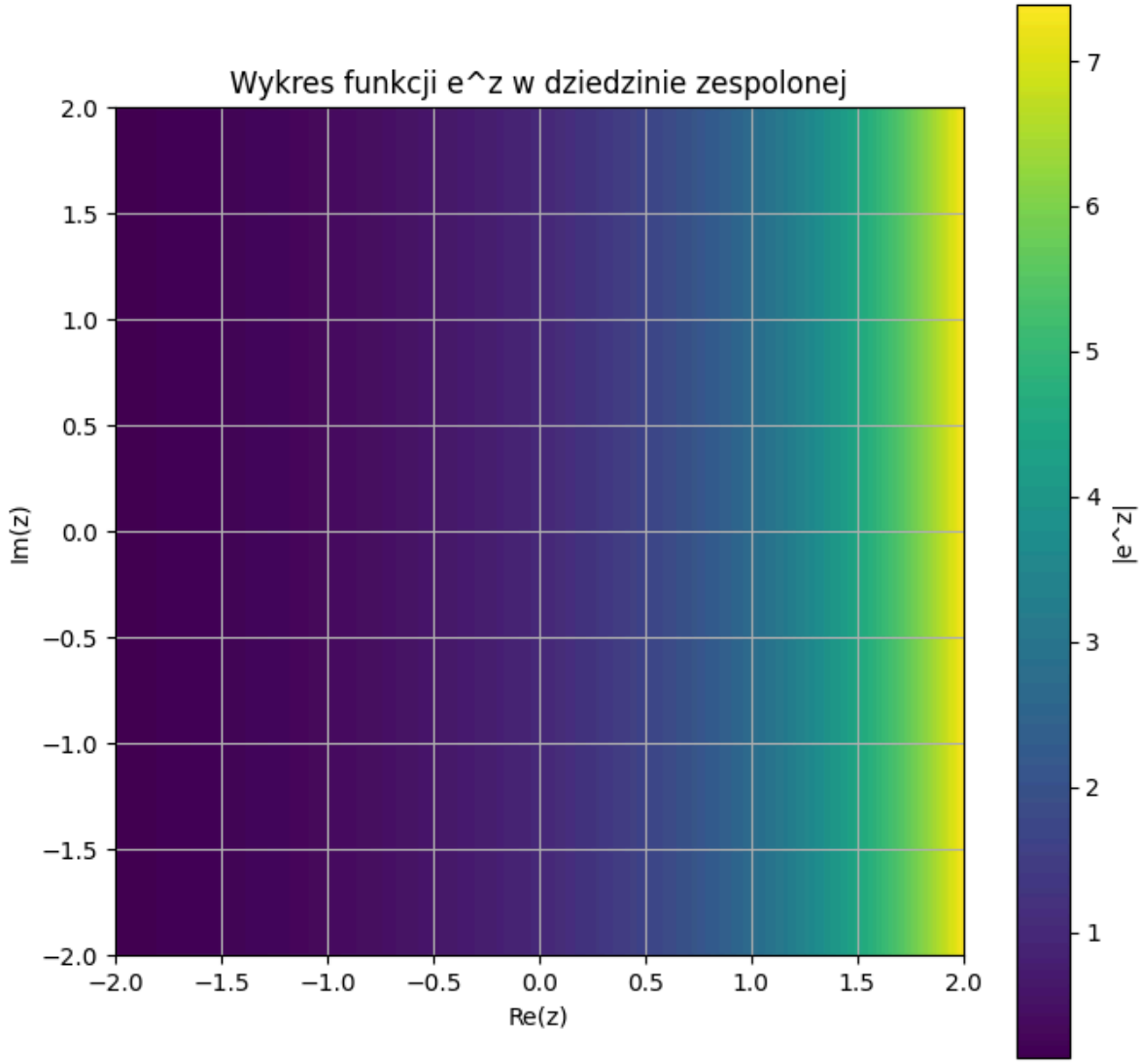
Warto sprawdzić (np. z kryterium ilorazowego), że jest zbieżny dla każdego $z \in \mathbb{C}$. Absolutnie kluczową własnością jest

$$e^{z+w} = e^z \cdot e^w.$$

Ponadto:

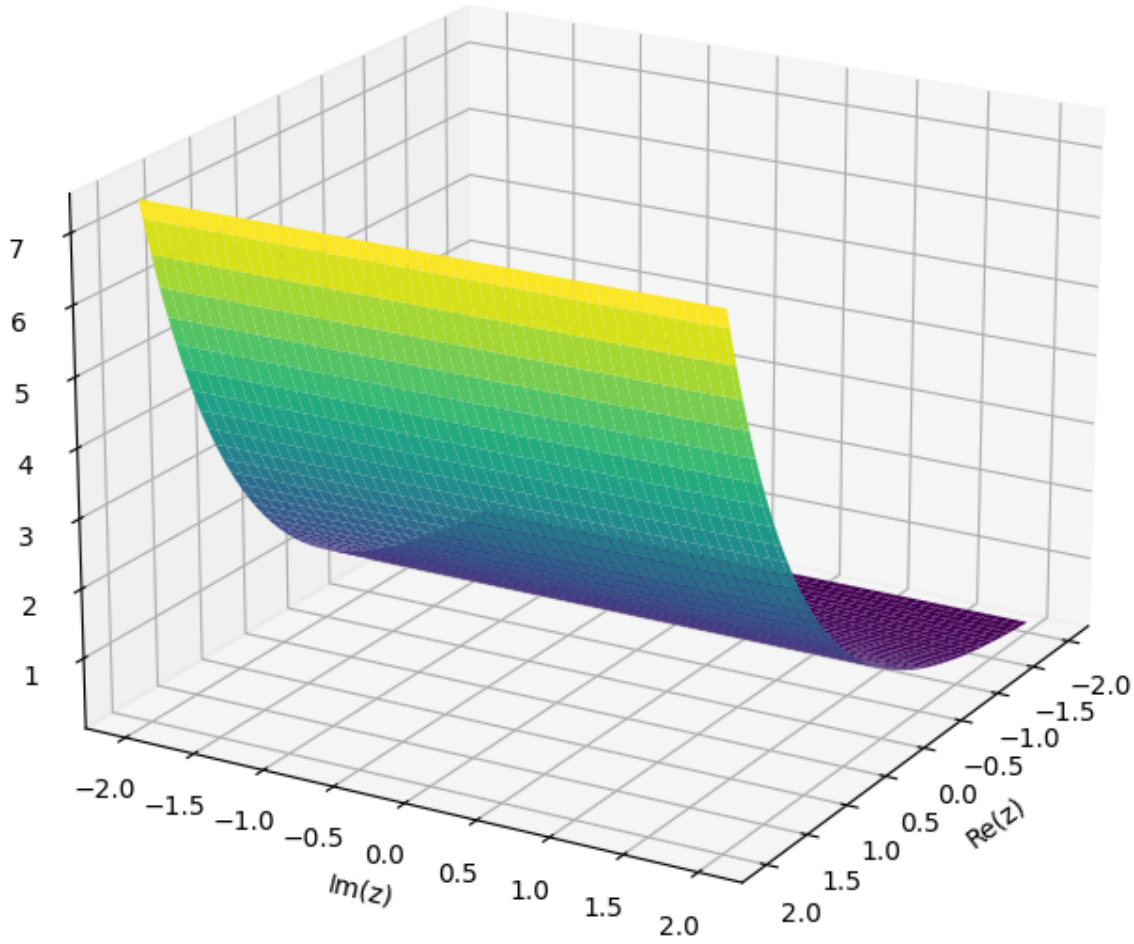
$$e^{z+w} = e^z e^w.$$

In [28]:
```
import numpy as np
import matplotlib.pyplot as plt

# Tworzenie siatki punktów w dziedzinie zespolonej
x = np.linspace(-2, 2, 400)
y = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x, y)
Z = X + 1j*Y

# Obliczanie wartości funkcji e^z
F = np.exp(Z)

# Wykres
plt.figure(figsize=(8, 8))
plt.imshow(np.abs(F), extent=(-2, 2, -2, 2), cmap='viridis', origin='lower')
plt.colorbar(label='|e^z|')
plt.title('Wykres funkcji e^z w dziedzinie zespolonej')
plt.xlabel('Re(z)')
plt.ylabel('Im(z)')
plt.grid()
plt.show()
```



In [29]:

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Tworzenie siatki punktów w dziedzinie zespolonej
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)
Z = X + 1j*Y

# Obliczanie wartości funkcji e^z
F = np.exp(Z)

# Wykres 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, np.abs(F), cmap='viridis')

# Ustawienia wykresu
ax.set_title('Wykres funkcji e^z w dziedzinie zespolonej')
ax.set_xlabel('Re(z)')
ax.set_ylabel('Im(z)')
ax.set_zlabel('|e^z|')
ax.view_init(elev=20, azim=30)

plt.show()
```



Przypomnijmy mnożenia szeregów metodą Cauchy'ego:

$$\begin{aligned} e^z e^w &= \sum_{n=0}^{\infty} \left(1 \cdot \frac{w^n}{n!} + \frac{z}{1!}\frac{w^{n-1}}{(n-1)!} + \frac{z^2}{2!}\frac{w^{n-2}}{(n-2)!} + \cdots + \frac{z^n}{n!} \cdot 1 \right) = \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \left(w^n + \binom{n}{1} z w^{n-1} + \binom{n}{2} z^2 w^{n-2} + \cdots + \binom{n}{n} z^n \right) = \\ &= \sum_{n=0}^{\infty} (z+w)^n = \\ &= e^{z+w}. \end{aligned}$$

Zdefiniujmy też $\sin z$ i $\cos z$:

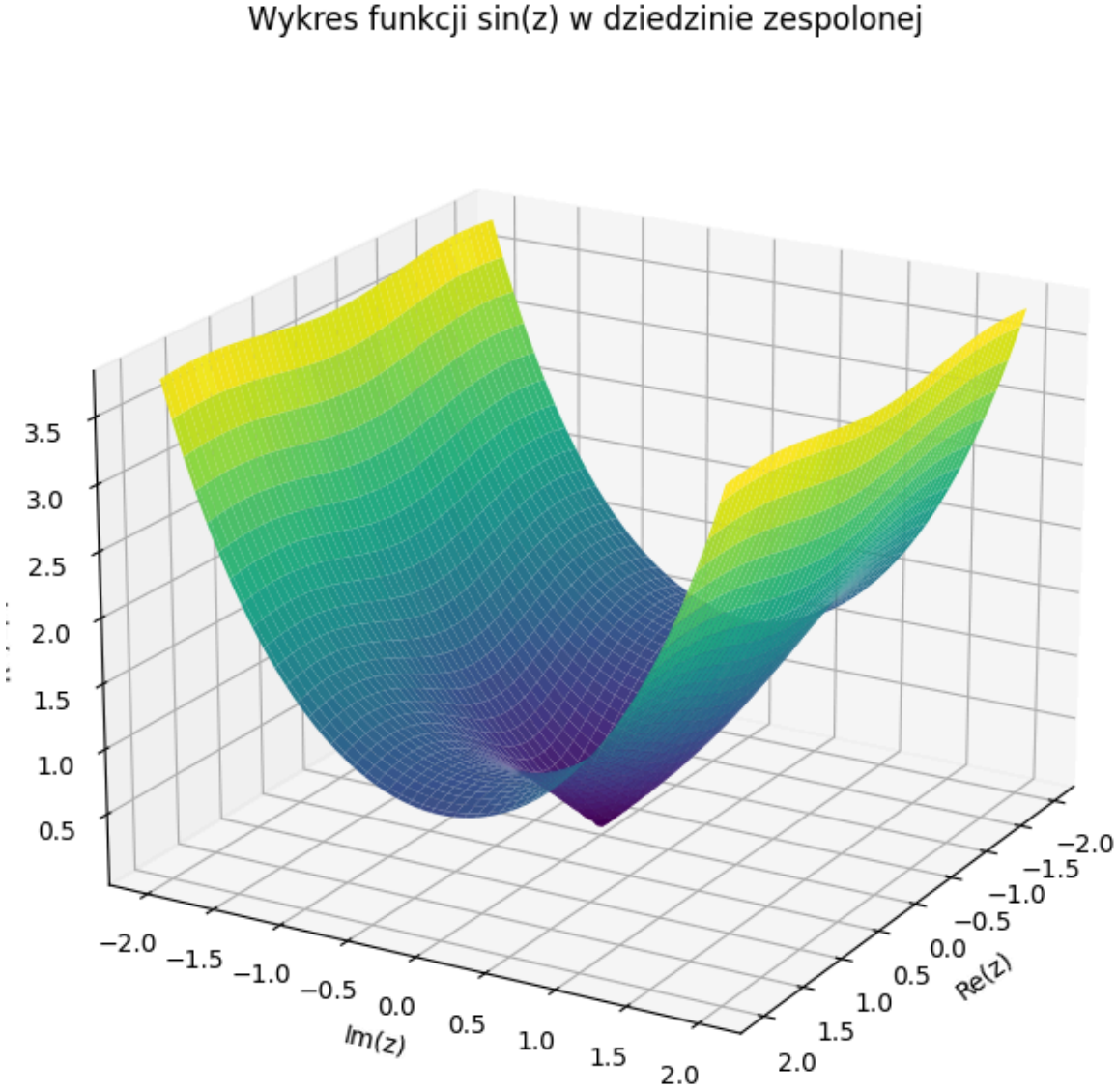
$$\begin{aligned} \sin z &= \frac{e^{iz} - e^{-iz}}{2i} = z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!} + \cdots \\ \cos z &= \frac{e^{iz} + e^{-iz}}{2} = 1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \frac{z^6}{6!} + \cdots. \end{aligned}$$

Funkcje te mają następujace własności:

$$\frac{d}{dz}\sin z = \frac{ie^{iz} + ie^{-iz}}{2i} = \cos z$$
$$\frac{d}{dz}\cos z = \frac{ie^{iz} - ie^{-iz}}{2} = -\sin z$$
$$\sin^2 z + \cos^2 z = \frac{e^{2iz} + 2 + e^{-2iz}}{4} + \frac{e^{2iz} - 2 + e^{-2iz}}{-4} = 1.$$

Czyli gdy $z = x \in \mathbb{R}$ to $|\cos x|$ i $|\sin x|$ są ograniczone przez 1, ale to nie jest prawdą dla wszystkich z!

In [31]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Tworzenie siatki punktów w dziedzinie zespolonej
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)
Z = X + 1j*Y

# Obliczanie wartości funkcji sinus zespolonego
F = np.sin(Z)

# Wykres 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, np.abs(F), cmap='viridis')

# Ustawienia wykresu
ax.set_title('Wykres funkcji sin(z) w dziedzinie zespolonej')
ax.set_xlabel('Re(z)')
ax.set_ylabel('Im(z)')
ax.set_zlabel('|sin(z)|')
ax.view_init(elev=20, azim=30)

plt.show()
```



Wygląda inaczej niż w przypadku rzeczywistym? To proszę zrobić przekrój dla $\Im(z) = 0$... Ale własności sa nam znane:

$$\cos(z+w) = \cos z \cos w - \sin z \sin w$$
$$\sin(z+w) = \sin z \cos w + \cos z \sin w$$

I tu damy uzasadnienie, bo to pozwoli docenić szeregi potęgowe:

$$\cos z \cos w - \sin z \sin w = \frac{(e^{iz}+e^{-iz})(e^{iw}+e^{-iw})}{4} + \frac{(e^{iz}-e^{-iz})(e^{iw}-e^{-iw})}{4} = \\ = \frac{e^{i(z+w)}+e^{-i(z+w)}}{2} = \cos(z+w).$$

Różniczkując obustronnie uzyskamy:

$$-\sin z \cos w - \cos z \sin w = -\sin(z+w).$$

Czyli

$$\sin(z+w) = \sin z \cos w + \cos z \sin w.$$

> To może dziwne, ale za pomocą tego szeregu możemy **zdefiniować** co to jest liczba π...

Niech x oznacza **najmniejszą** liczbę, dla której $\cos x = 0$. Wtedy $\pi = 2x$.

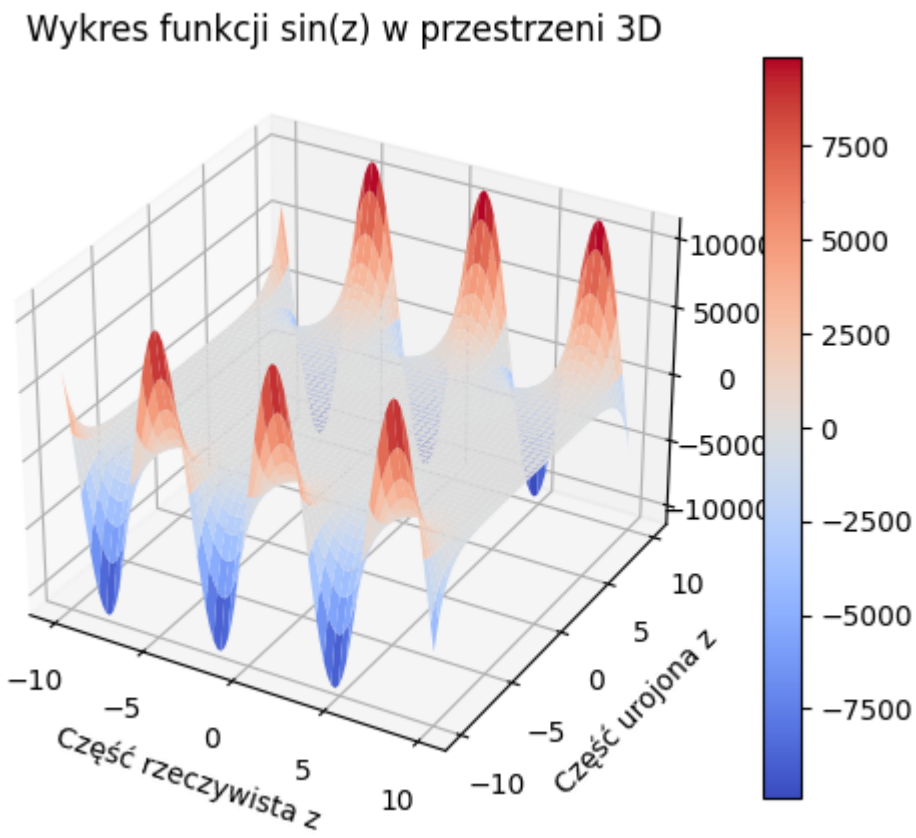
A teraz uzasadnimy, że to poprawne: $\sin^2 z + \cos^2 z = 1$, czyli $\sin \frac{\pi}{2} = \pm 1$. A ponieważ $\cos x > 0$ na $[0, \frac{\pi}{2}]$ (rozwinąć w szereg!), to $\sin \frac{\pi}{2} \geq 0$ z twierdzenia o wartości średniej. Czyli $\sin \frac{\pi}{2} = 1$.

Czyli także:

$$\begin{aligned}
\cos\left(z + \frac{\pi}{2}\right) &= -\sin z \\
\sin\left(z + \frac{\pi}{2}\right) &= \cos z \\
\cos(z + \pi) &= -\cos z \\
\sin(z + \pi) &= -\sin z \\
\cos(z + 2\pi) &= \cos z \\
\sin(z + 2\pi) &= \sin z.
\end{aligned}$$

To raz jeszcze wykres (zmienimy dziedzinę):

In [34]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Wygeneruj zestaw wartości x i y
x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

# Oblicz wartości sin(z)
Z = np.sin(X + 1j * Y)

# Stwórz wykres funkcji sin(z) w przestrzeni 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, np.real(Z), cmap='coolwarm')

# Dodaj etykiety osi
ax.set_xlabel('Część rzeczywista z')
ax.set_ylabel('Część urojona z')
ax.set_zlabel('Część rzeczywista sin(z)')

# Dodaj tytuł wykresu
ax.set_title('Wykres funkcji sin(z) w przestrzeni 3D')

# Dodaj pasek kolorów
fig.colorbar(surf)

# Wyświetl wykres
plt.show()
```



Przypadek szczególny zespolonych szeregów potęgowych - szeregi Fouriera.

Szeregi Fouriera są zespolonymi szeregami potęgowymi w sensie, że są reprezentacją funkcji okresowej za pomocą sumy nieskończonej wyrażeń postaci:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega x} = \sum_{n=-\infty}^{\infty} c_n \left(e^{i\omega x}\right)^n = \sum_{n=-\infty}^{\infty} c_n z^n,$$

gdzie $z = e^{i\omega x} = \cos \omega x + i \cdot \sin \omega x$. Zapamiętajmy pojawienie się w tym wzorze funkcji trygonometrycznych...

Każdy składnik $e^{in\omega x}$ w tej sumie jest funkcją zespoloną, a współczynniki c_n mogą być również zespolone. Dlatego szereg Fouriera jest zespolonym szeregiem potęgowym, ponieważ składa się z potęg zmiennej niezależnej x pomnożonych przez funkcje zespolone $e^{in\omega x}$.

> To jednak bardzo ciekawy przypadek i poświęcimy mu osobny wykład!

Zobaczmy też coś z informatyki kwantowej: QFT ("Quantum Fourier Transform"):

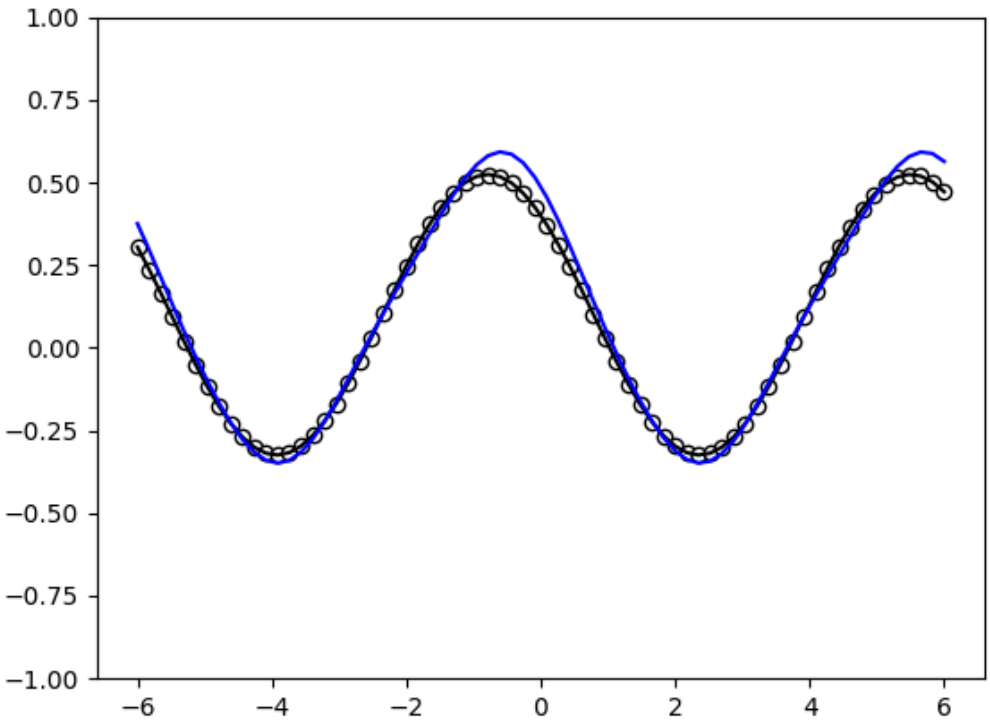
W informatyce kwantowej, kwantowy szereg Fouriera może być wyrażony jako kombinacja operacji kwantowych na rejestrze kwantowym. Dla n-bitowego rejestru kwantowego, kwantowy szereg Fouriera może być zrealizowany za pomocą tzw. kwantowej bramki Fouriera.

Wzór na kwantową bramkę Fouriera dla n-bitowego rejestru kwantowego jest dany przez:

$$QFT = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}} |y\rangle \langle x| .$$

gdzie $|x\rangle$ i $|y\rangle$ to stany kwantowe rejestru, a $N = 2^n$ to liczba stanów w rejestrze. Tu np.obliczamy pierwsze $2 * K + 1$ współczynników Fouriera funkcji $2 * \pi$ okresowej:

```
def fourier_coefficients(f, K):

n_coeffs = 2 * K + 1

t = np.linspace(0, 2 * np.pi, n_coeffs, endpoint=False)

y = np.fft.rfft(f(t)) / t.size

return y
```



Kwantowa bramka Fouriera jest używana do przeprowadzenia kwantowej transformacji Fouriera na reprezentacji stanu kwantowego rejestru. Kwantowa transformata Fouriera (QFT) jest kluczowym narzędziem dla algorytmów kwantowych, takich jak algorytm Shora do rozkładania dużych liczb na czynniki pierwsze i oszacowania fazy kwantowej do znajdowania wartości własnych operatorów.

Zastosowanie zespolonych szeregów potęgowych.

A teraz możemy docenić jak przydatne jest różniczkowanie szeregów potęgowych - zróżniczkujmy szereg ("wyraz po wyrazie"):

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots$$

otrzymując

$$0 + 1 + z + \frac{z^2}{2!} + \cdots = e^z.$$

Czyli

$$(e^z)' = e^z.$$

Ćwiczenie.

Podobnie proszę zbadać $\sin z$ i $\cos z$. To chyba nawet prostsze niż z definicji - wystarczą pochodne wielomianu (potęgi), a tego można "nauczyć" nawet komputer...

Wracamy do tematu: obliczamy **sumy** szeregów!

Różniczkowanie (i całkowanie) szeregów potęgowych umożliwi nam obliczanie sum pewnych szeregów. Do tej pory było niewiele takich szeregów, których sumy potrafiliśmy wyznaczyć dokładnie.

Podamy prosty przykład jak to wykorzystać.

Niech $f(x) = \sum_{n=0}^{\infty} x^n$. Promień zbieżności to oczywiście $x = 1$. Zróżniczkujmy go:

$$f'(x) = \left(\sum_{n=0}^{\infty} x^n\right)' = \sum_{n=1}^{\infty} n \cdot x^{n-1}.$$

Ale $f(x)$ to przecież suma szegu geometrycznego o ilorazie x, czyli $f(x) = \frac{1}{1-x}$ (można też wykorzystać tę postać funkcji i rozwinąć ją w szereg Taylora!). Policzmy pochodną:

$$\left(\frac{1}{1-x}\right)' = \frac{1}{(1-x)^2}.$$

Czyli dla $|x| < 1$ zachodzie równość, a więc:

$$\sum_{n=1}^{\infty} n \cdot x^{n-1} = \frac{1}{(1-x)^2}.$$

Wstawiając do tego wzoru liczby x takie, że $|x| < 1$ mamy gotowe sumy szeregów.

Np. dla $x = \frac{1}{2}$:

$$\sum_{n=1}^{\infty} \frac{n}{2^n} = \frac{1}{(1-\frac{1}{2})^2} = 4.$$

Proszę wstawić kilka innych liczb oraz - na ćwiczeniach - zróżniczkować ten szereg $f'(x)$ ponownie i zbadać jakie szeregi teraz da się obliczyć...

Funkcje hiperboliczne.

Definiujemy funkcje hiperboliczne:

$$\cosh z = \frac{e^z + e^{-z}}{2} = 1 + \frac{z^2}{2!} + \frac{z^4}{4!} + \frac{z^6}{6!} + \cdots$$
$$\sinh z = \frac{e^z - e^{-z}}{2} = z + \frac{z^3}{3!} + \frac{z^5}{5!} + \frac{z^7}{7!} + \cdots$$

I ponownie różniczkując szeregi potęgowe:

$$\frac{d}{dz}\cosh z = \sinh z$$
$$\frac{d}{dz}\sinh z = \cosh z$$

A z definicji:

$$\cosh iz = \cos z$$
$$\sinh iz = i \sin z.$$

Również łatwo sprawdzić, że:

$$\cosh^2 z - \sinh^2 z = 1.$$

In [37]:
```
from sympy import *

x = symbols('x')
f = exp(x)/x**2

# Obliczenie granicy funkcji f dla x dążącego do nieskończoności
lim = limit(f, x, oo)

# Sprawdzenie zbieżności funkcji f dla x dążącego do nieskończoności
if lim.is_finite:
    print("Funkcja jest zbieżna dla x dążącego do nieskończoności")
else:
    print("Funkcja jest rozbieżna dla x dążącego do nieskończoności")
```

```
Funkcja jest rozbieżna dla x dążącego do nieskończoności
```

W tym przykładzie funkcja $\frac{\exp(x)}{x^2}$ jest podana jako wejście.

Funkcja *convergence* z biblioteki SymPy sprawdza zbieżność szeregu potęgowego tej funkcji (w nieskończoności (oo)). Funkcja zwraca True, jeśli szereg jest zbieżny, lub False, jeśli jest rozbieżny.

Warto przeanalizować **JAK** ta biblioteka działa (dokumentacja!) ==> warunek Cauchy'ego, kryteria Cauchy'ego i d'Alemberta (tw. Cauchy'ego-Hadamarda), porównawcze i całkowe!

Szeregi potęgowe - analiza i przetwarzanie sygnałów cyfrowych.

Możemy wykorzystać szereg potęgowy do przybliżenia funkcji filtracji w dziedzinie częstotliwości.

Przykładowy problem:: zaimplementować filtr dolnoprzepustowy w dziedzinie częstotliwości za pomocą szeregu potęgowego i sprawdzić jego działanie na sygnale.

Algorytm:

1. Definiowanie funkcji filtra: Użyjemy zespolonych szeregów potęgowych do przybliżenia funkcji filtra dolnoprzepustowego w dziedzinie częstotliwości.

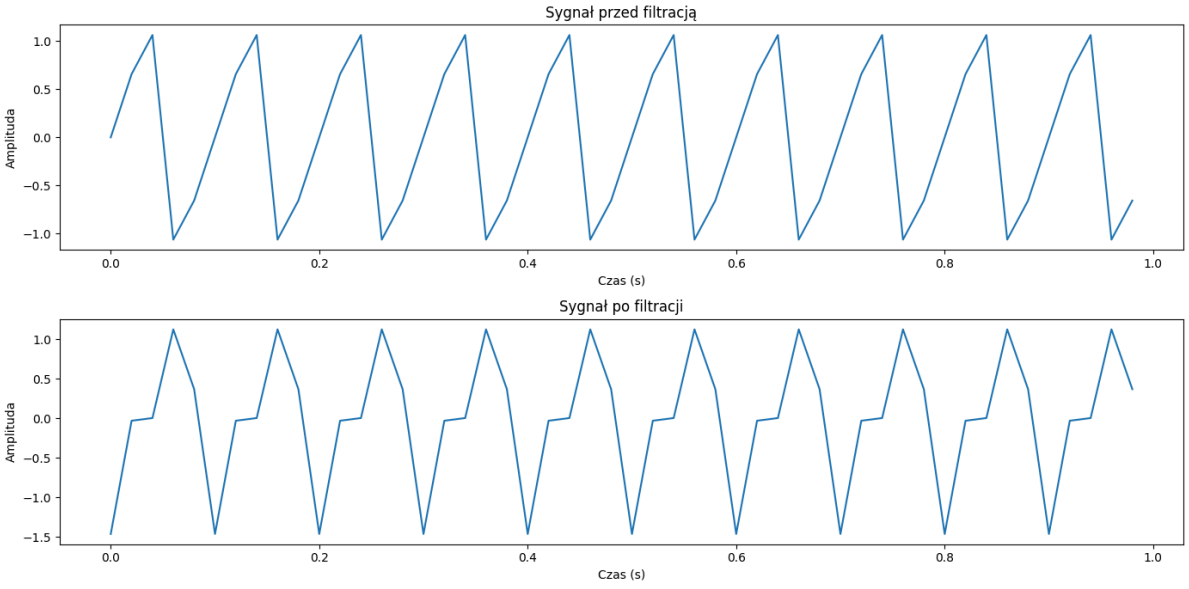
2. Aplikacja filtra do sygnału: Przefiltrujemy sygnał wejściowy, wykorzystując zaprojektowany filtr.

W poniższym kodzie funkcja *low_pass_filter_frequency_response* generuje odpowiedź częstotliwościową filtra dolnoprzepustowego przy użyciu zespolonego szeregu Taylora. Używamy szeregu potęgowego do przybliżenia funkcji

$$\text{odpowiedź} = \frac{1}{1 + \sum_{n=0}^{N-1} \frac{\left(j \cdot \frac{\text{częstotliwości}}{\text{częstotliwość odcięcia}}\right)^n}{n!}} \quad .$$

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

def low_pass_filter_frequency_response(cutoff_frequency, num_terms, frequencies):
    """
    Funkcja generująca odpowiedź częstotliwościową filtra dolnoprzepustowego
    przy użyciu szeregu potęgowego.

    :param cutoff_frequency: Częstotliwość odcięcia filtra.
    :param num_terms: Liczba wyrazów szeregu Taylora.
    :param frequencies: Wektor częstotliwości.
    :return: Odpowiedź częstotliwościowa filtra.
    """
    response = np.zeros_like(frequencies, dtype=complex)
    for n in range(num_terms):
        term = (1j * frequencies / cutoff_frequency) ** n / np.math.factorial(n)
        response += term
    return 1 / response

def apply_filter(signal, filter_response, sampling_rate):
    """
    Zastosowanie filtra do sygnału w dziedzinie częstotliwości.

    :param signal: Sygnał wejściowy.
    :param filter_response: Odpowiedź filtra w dziedzinie częstotliwości.
    :param sampling_rate: Częstotliwość próbkowania sygnału.
    :return: Sygnał wyjściowy po filtracji.
    """
    # Przekształcamy sygnał do dziedziny częstotliwości
    signal_fft = np.fft.fft(signal)

    # Częstotliwości odpowiadające każdemu pasmu w sygnale
    frequencies = np.fft.fftfreq(len(signal), 1 / sampling_rate)

    # Zastosowanie filtra
    filtered_fft = signal_fft * filter_response

    # Powrót do dziedziny czasowej
    filtered_signal = np.fft.ifft(filtered_fft)
    return np.real(filtered_signal)

# Parametry filtra
cutoff_frequency = 5  # Częstotliwość odcięcia filtra (Hz)
```

```
num_terms = 10  # Liczba wyrazów szeregu Taylora

# Parametry sygnału
sampling_rate = 50  # Częstotliwość próbkowania (Hz)
duration = 1  # Czas trwania sygnału (s)
t = np.linspace(0, duration, int(sampling_rate * duration), endpoint=False)
signal = np.sin(2 * np.pi * 10 * t) + 0.5 * np.sin(2 * np.pi * 30 * t)

# Wektor częstotliwości
frequencies = np.fft.fftfreq(len(signal), 1 / sampling_rate)

# Generowanie odpowiedzi częstotliwościowej filtra
filter_response = low_pass_filter_frequency_response(cutoff_frequency, num_terms, f

# Filtracja sygnału
filtered_signal = apply_filter(signal, filter_response, sampling_rate)

# Wizualizacja wyników
plt.figure(figsize=(14, 7))

plt.subplot(2, 1, 1)
plt.title("Sygnał przed filtracją")
plt.plot(t, signal)
plt.xlabel("Czas (s)")
plt.ylabel("Amplituda")

plt.subplot(2, 1, 2)
plt.title("Sygnał po filtracji")
plt.plot(t, filtered_signal)
plt.xlabel("Czas (s)")
plt.ylabel("Amplituda")

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



Zastosowanie funkcji zespolonych w informatyce: algorytm generowania fraktali Mandelbrota.

Jednym z klasycznych przykładów zastosowania funkcji zespolonych w informatyce jest generowanie fraktali Mandelbrota. Fraktal Mandelbrota jest zbiorem punktów na

płaszczyźnie zespolonej, który ilustruje złożoną strukturę geometryczną wynikającą z iteracyjnych przekształceń funkcji zespolonych.

> Zbiór Mandelbrota jest definiowany w oparciu o iterację funkcji kwadratowej $f_c(z) = z^2 + c$, gdzie z i c są liczbami zespolonymi. Dla każdego punktu c na płaszczyźnie zespolonej, iterujemy z począwszy od $z = 0$:

$$z_{n+1} = z_n^2 + c.$$

Punkt c należy do zbioru Mandelbrota, jeśli ciąg (z_n) pozostaje ograniczony. W praktyce, sprawdzamy to poprzez iterowanie funkcji dla określonej liczby kroków N i sprawdzanie, czy moduł $|z_n|$ przekracza pewną wartość progową (zwykle 2). To oczywiście nie jest ścisły dowód - bo tego na komputerze nie da się sprawdzić...

> **Algorytm generowania fraktala Mandelbrota.**

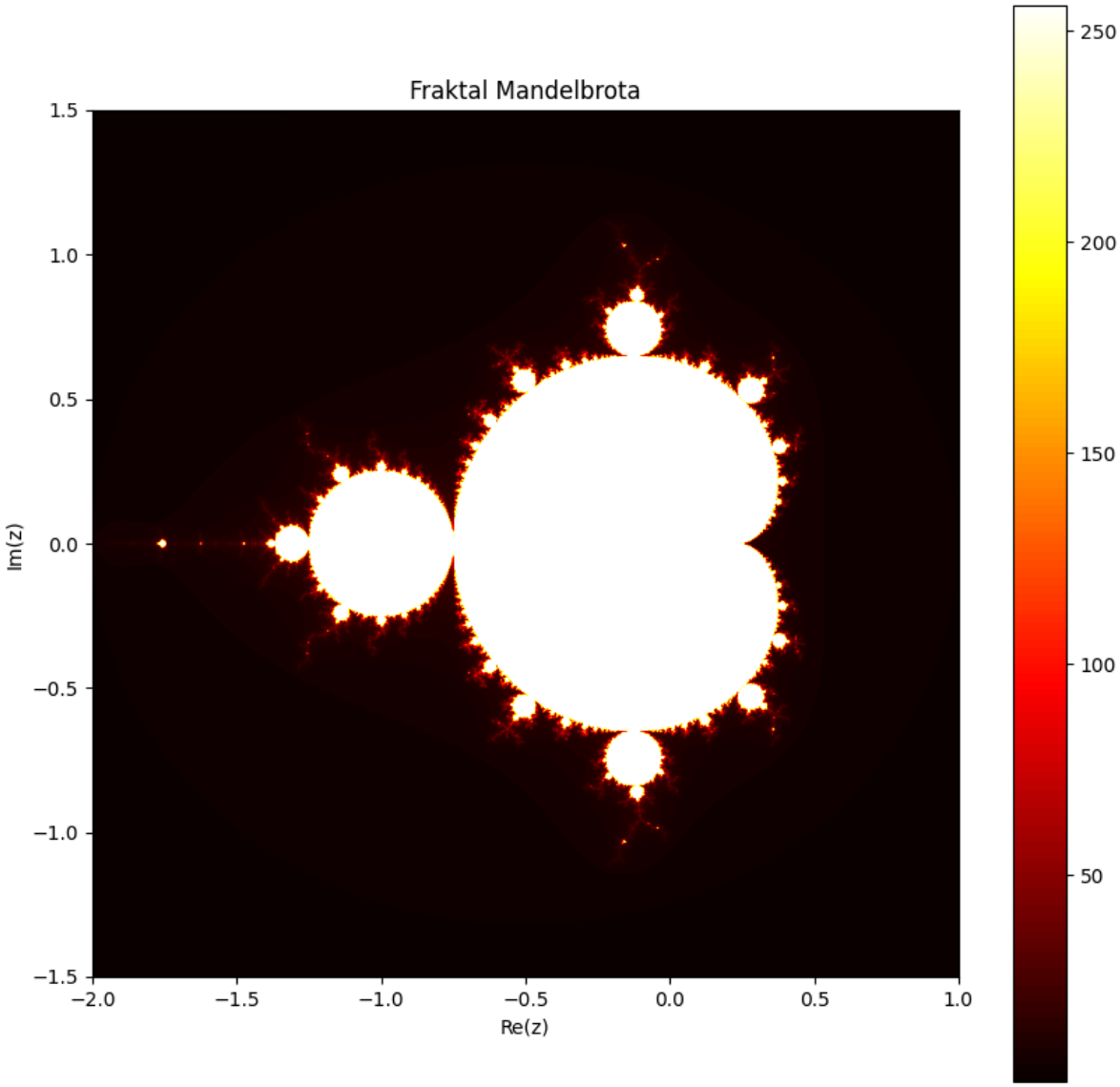
1. **Parametry algorytmu**: **max_iter**: maksymalna liczba iteracji dla sprawdzenia ograniczoności ciągu. **threshold**: wartość progowa dla $|z_n|$. **resolution**: rozdzielczość obrazu (szerokość i wysokość).

2. **Iteracja**: dla każdego punktu c na płaszczyźnie zespolonej, począwszy od $z = 0$, iterujemy funkcję $f_c(z)$. Sprawdzamy, czy $|z_n|$ przekracza wartość progową. Jeśli tak, przerywamy iterację i zapisujemy liczbę wykonanych iteracji.

3. **Wizualizacja**: Przypisujemy kolor każdemu punktowi w zależności od liczby iteracji potrzebnej do osiągnięcia wartości progowej.

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

def mandelbrot(c, max_iter):
    z = 0
    for n in range(max_iter):
        if abs(z) > 2:
            return n
        z = z**2 + c
    return max_iter

def generate_mandelbrot_set(xmin, xmax, ymin, ymax, width, height, max_iter):
    r1 = np.linspace(xmin, xmax, width)
    r2 = np.linspace(ymin, ymax, height)
    mandelbrot_set = np.zeros((width, height))
    for i in range(width):
        for j in range(height):
            mandelbrot_set[i, j] = mandelbrot(r1[i] + 1j * r2[j], max_iter)
    return mandelbrot_set

# Parametry fraktala
xmin, xmax, ymin, ymax = -2.0, 1.0, -1.5, 1.5
width, height = 800, 800
max_iter = 256

# Generowanie fraktala
mandelbrot_set = generate_mandelbrot_set(xmin, xmax, ymin, ymax, width, height, max
```

```
# Wizualizacja
plt.figure(figsize=(10, 10))
plt.imshow(mandelbrot_set.T, extent=[xmin, xmax, ymin, ymax], cmap='hot', interpola
plt.colorbar()
plt.title('Fraktal Mandelbrota')
plt.xlabel('Re(z)')
plt.ylabel('Im(z)')
plt.show()
```



> Powyższy przyklad to raczej jednak ciekawostka. O (chyba) najczęściej spotykanym zastosowaniu funkcji zespolonych i szeregów zespolonych w informatyce, czyli o **transformacie Fouriera** i jej dyskretnej wersji powiemy wkrótce szczegółowo. Większość operacji wykonamy opierając się o postacie zespolone pewnych funkcji...

Zadania.

> Zadanie 1.

Zbadać zbieżność ciągu funkcyjnego (f_n) o n-tym wyrazie:

(a) $f_n(x) = \dfrac{2}{1+nx^2}$,

(b) $f_n(x) = \dfrac{x}{1+n^2x^2}$,

(c) $f(x) = \dfrac{2nx}{2+nx^2}$,

(d) $f_n(x) = 2n \sin \dfrac{x}{n}$.

Zadanie 2.

Wykazać jednostajną zbieżność ciągu funkcyjnego (f_n) o n-tym wyrazie:

(a) $f_n(x) = x + \dfrac{1}{n}\sin nx$,

(b) $f_n(x) = \dfrac{1}{n}\operatorname{arctg} x^n$.

Zadanie 3.

Wyznaczyć zbiór tych wartości x, dla których podany szereg funkcyjny jest zbieżny:

(a) $\sum_{n=1}^{\infty} e^{-n^2x}$,

(b) $\sum_{n=1}^{\infty} \dfrac{x^n}{n+\sqrt{n}}$,

(c) $\sum_{n=1}^{\infty} nxe^{-nx}$,

(d) $\sum_{n=1}^{\infty} x^n \operatorname{tg} \dfrac{x}{2^n}$,

(e) $\sum_{n=1}^{\infty} \dfrac{x^n}{1+x^{2n}}$.

Zadanie 4.

Zbadać zbieżność szeregów funkcyjnych:

(a) $\sum_{n=1}^{\infty} \dfrac{\sin 3nx}{2^n}$,

(b) $\sum_{n=1}^{\infty} \dfrac{\cos nx}{2n\sqrt{n}}$,

(c) $\sum_{n=1}^{\infty} \dfrac{2x}{4+n^4x^2}$,

(d) $\sum_{n=1}^{\infty} \dfrac{(-1)^{n+1}}{n^x}$ dla $x \in \mathbb{R}_+ \cup \{0\}$.

Zadanie 5.

Wykazać jednostajną zbieżność szeregów funkcyjnych:

(a) $\sum_{n=1}^{\infty} \dfrac{\sin^2 nx}{n\sqrt{n}}$,

(b) $\sum_{n=1}^{\infty} \dfrac{\sin 2^n \pi x}{2^n}$,

(c) $\sum_{n=1}^{\infty} \dfrac{2 + 3\cos 2nx}{\sqrt{n} + \sqrt{n^3}}$,

(d) $\sum_{n=1}^{\infty} (-1)^n \dfrac{e^{-nx^2}}{n^3 + n}$.