Ciągi i szeregi funkcyjne.

Definicja.

Dany jest ciąg funkcji rzeczywistych (f_n) określonych na wspólnej dziedzinie $D \subset \mathbb{R}$, czyli $f_n : D \to \mathbb{R}$.

> > Uwaga: wartości funkcji f_n nie muszą być rzeczywiste, mogą być w dowolnej, wspólnej przestrzeni Y - na ogół metrycznej.

> Ciągiem funkcyjnym nazywamy funkcję o dziedzinie $\mathbb{N}$ i wartościach w tym zbiorze $\{f_n : n \in \mathbb{N}\}$.

> Szeregiem funkcyjnym utworzonym z ciągu f_n jest wyrażenie

$$S = \sum_{n=1}^{\infty} f_n = \lim_{N\to\infty} \sum_{n=1}^{N} f_n \text{ albo} \quad S(\cdot) = \sum_{n=1}^{\infty} f_n(\cdot).$$

> Ciąg sum częściowych takiego szeregu, to ciąg funkcji

$$S_N = \sum_{k=1}^{N} f_k \text{ albo} \quad S_N(\cdot) = \sum_{k=1}^{N} f_k(\cdot).$$

> (Kropka w nawiasie ma za zadanie przypominać, że f_n, S_n itp. nie są liczbami tylko funkcjami. Nie chcemy też pisać $f_N(x), S_N(x)$ itp., gdyż to by oznaczało wartość funkcji f w jednym punkcie, a więc liczbę.)

Ciąg [gdy szereg $\Rightarrow$ gdy położyć S_n w miejsce f_n] liczbowy (f_n) może na rozważanym podzbiorze $E \subset D$ (w szczególności na całej dziedzinie D, lub na swoim obszarze zbieżności) być

> a) zbieżny punktowo, to znaczy zbieżny jest ciąg $(f_n(x))$ w każdym punkcie $x \in E$,
>
> b) zbieżny punktowo bezwzględnie, to znaczy być zbieżnym bezwzględnie w każdym punkcie $x \in E$: $(|f_n(x)|)$ jest zbieżny,
>
> c) zbieżny jednostajnie, to znaczy $a_n = \sup_{x\in E} f_n$ jeat zbieżny, czyli spełniać warunek

$$\forall_{\epsilon>0}\exists_{n_0}\forall_{n\geq n_0}\forall_{x\in E}\ |f(x) - f_n(x)| < \epsilon,$$

> d) zbieżny jednostajnie bezwzględnie, kiedy ciąg $b_n = \sup_{x\in E} |f_n|$ jest zbieżny.

Proste motywacje.

A gdzie to napotkamy w informatyce? Jest wiele takich sytuacji, choć nie zawsze padnie wówczas słowo **ciąg**. Klasyczne przypadki omówimy na wykładzie i będa to:

1. **ciągi funkcji aproksymujacych** daną funkcję f (mając f szukamy g_n dla której odległość of f (wyjaśnimy jak to rozumieć)

jest np. mniejsza niż $\frac{1}{n}$: i tak dla każdego n powstaje ciąg,

2. **wielomiany będące sumami częściowymi szeregów Taylora** i przybliżające wartości funkcji rozwijanej w taki szereg

(tu mamy i ciąg i szereg funkcyjny), lub wielomiany aproksymacyjne i ich zastosowania w metodzie najmniejszych kwadratów, która jest stosowana w analizie numerycznej i statystyce. Metoda ta polega na znalezieniu wielomianu $P_n(x)$, który najlepiej "pasuje" do zadanej funkcji f na danym przedziale $[a, b]$ poprzez minimalizację średniego kwadratu różnicy pomiędzy $f(x)$ a $P_n(x)$ na tym przedziale.

3. pojawi sie podobna idea **przybliżania funkcji ciągiem wielomianów trygonometrycznych i szeregiem (Fouriera)**,

np. ciąg funkcji trygonometrycznych: $f_n(x) = \sin(nx)$. Ciąg ten składa się z funkcji o różnych okresach (częstotliwościach fal). W informatyce taki ciąg funkcji może być wykorzystany w dziedzinie przetwarzania sygnałów do analizy i przetwarzania fal dźwiękowych lub obrazów.

4. ciąg funkcji gamma: $f_n(x) = \gamma(n + x)$. Ciąg ten składa się z funkcji γ dla różnych wartości argumentów.

Funkcja γ jest zdefiniowana dla wszystkich liczb zespolonych, z wyjątkiem ujemnych liczb całkowitych i zer. W praktyce jednak często korzysta się z wartości funkcji γ dla dodatnich liczb całkowitych i połówek. Ciąg funkcji γ ma szerokie zastosowanie w matematyce i fizyce, gdzie jest wykorzystywany do rozwiązywania różnych problemów, takich jak równania różniczkowe, całki i wiele innych. W informatyce ciąg funkcji γ jest stosowany w wielu algorytmach numerycznych, takich jak **metoda Gaussa-Laguerre'a** do całkowania numerycznego (o czym powiemy później po wprowadzeniu układów ortogonalnych), czy algorytm Brenta do znajdowania pierwiastków funkcji.

5. **QuickSort jako ciąg funkcji**: jest to algorytmem sortowania oparty na strategii "dziel i zwyciężaj". Można go rozumieć jako ciąg funkcji, które operują na zbiorze danych poprzez podział go na mniejsze fragmenty, porównywanie elementów i ich przemieszczanie w odpowiednie miejsca. Każde wywołanie rekurencyjne QuickSorta można traktować jako kolejną funkcję w ciągu, która operuje na coraz mniejszych podzbiorach danych, aż do momentu, gdy zbiór stanie się posortowany. Podobnie **MergeSort** również jest algorytmem sortowania, który korzysta z strategii "dziel i zwyciężaj".

Proces sortowania w MergeSort można interpretować jako ciąg kroków, w których poszczególne podzbiory danych są sortowane, a następnie scalane w coraz większe i posortowane podzbiory, aż do uzyskania całkowicie posortowanego zbioru danych.

6. **łańcuchy Markowa.** W wielu modelach badanych środkami informatyki będą wykorzystane procesy, a zwłaszcza łańcuchy Markowa (na tym wykładzie to tylko pojęcie matematyczne, dużo więcej o nim - na innych przedmiotach, np. rachunek prawdopodobieństwa, procesy stochastyczne czy teoria grafów - wszystkie z zastosowaniami w informatyce). Tymczasem łańcuch Markowa z punktu widzenia analizy to ciąg funkcyjny, a funkcjami są zmienne losowe (czyli: funkcje). Są m.in. doskonałym narzędziem do tworzenia sztucznej inteligencji, losowanie obiektów - aby wygenerować losowy obiekt zgodnie z pewnym rozkładem konstruujemy łańcuch Markowa dla którego rozkład ten jest rozkładem stacjonarnym, po czym wykonujemy odpowiednio długie symulacje tego łańcucha (metodą Monte Carlo z wykorzystaniem łańcuchów Markowa, ang. Markov Chain Monte Carlo). Dale np. modelowanie - można skutecznie modelować wiele naturalnych procesów i struktur (np. modelując język naturalny można zbudować algorytm kompresji tekstu lub do generowania losowych tekstów). Oczywiście w fizyce i biologii obliczeniowej. No i wreszcie algorytm "PageRank" (algorytm szeregowania stron PageRank). Algorytm ten bazuje na łańcuchu Markowa, który jest modelem procesu poruszania się użytkownika po zbiorze wszystkich (znanych systemowi) stron WWW. I może jeszcze: generowanie haseł za pomocą łańcuchów Markowa

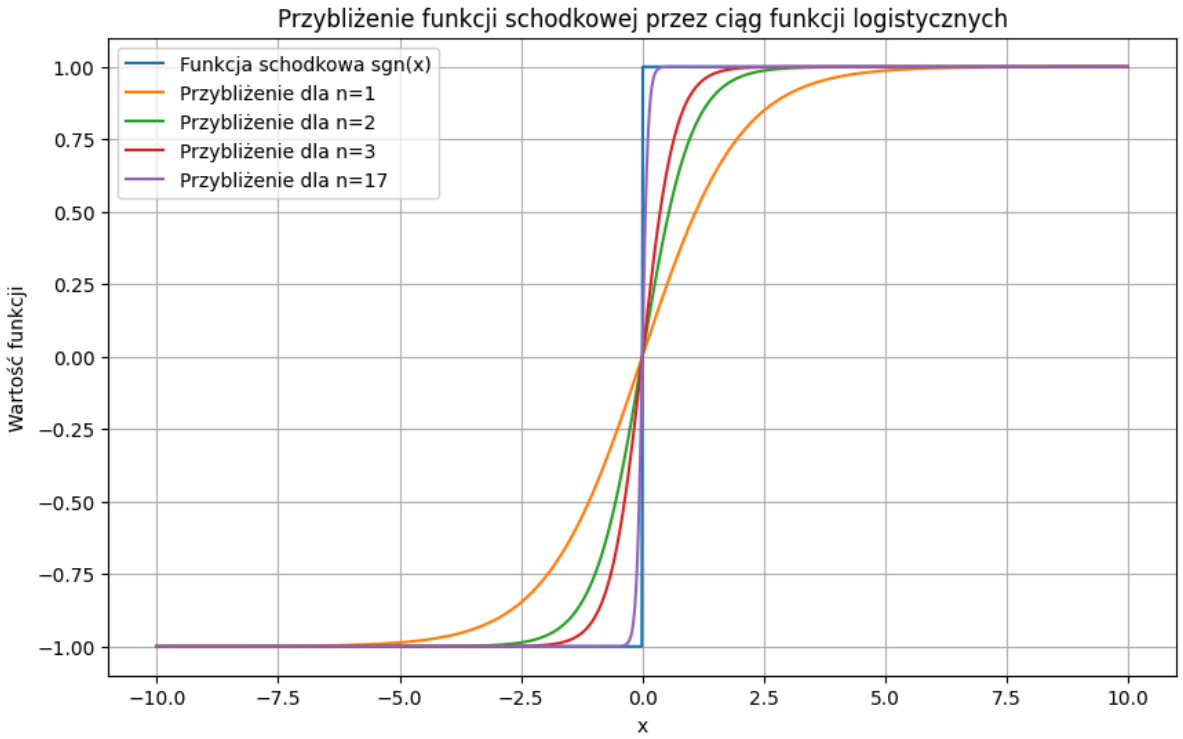
Uwaga.

> Pamiętajmy też, że podobnie jak dla ciągów liczbowych: zapewnimy sobie zbieżność danego ciągu lub szeregu funkcyjnego, a następnie wykorzystamy wyrazy (ciąg) lub sumy częściowe (szereg) do **przybliżania** granicy (cokolwiek by to na razie znaczyło - ściślej o tym powiemy później). To także działa w dwóch kierunkach: mając dana funkcję chcemy ją przybliżyć innymi ("lepszymi" obliczeniowo), albo na odwrót - mając procedurę generującą pewien ciąg lub szereg funkcyjny chcemy znaleźć jej (jego) granicę!

Więcej informacji - na kolejnych wykładach...

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt

# Funkcja logistyczna zmodyfikowana dla x < 0
def sigmoid(x, n):
    return 1 / (1 + np.exp(-n * x))

# Przybliżenie funkcji schodkowej przez ciąg funkcji logistycznych
def approximate_sgn(x, n=1):
    return 2 * sigmoid(x, n) - 1

# Zakres danych wejściowych
x_values = np.linspace(-10, 10, 1000)

# Funkcja schodkowa
sgn_values = np.sign(x_values)

# Przybliżenie funkcji schodkowej przez ciąg funkcji logistycznych dla różnych n
n_values = [1, 2, 3, 17]
approx_sgn_values = [approximate_sgn(x_values, n) for n in n_values]
```

```python
# Rysowanie wykresów
plt.figure(figsize=(10, 6))
plt.plot(x_values, sgn_values, label='Funkcja schodkowa sgn(x)')
for i, n in enumerate(n_values):
    plt.plot(x_values, approx_sgn_values[i], label=f'Przybliżenie dla n={n}')
plt.xlabel('x')
plt.ylabel('Wartość funkcji')
plt.title('Przybliżenie funkcji schodkowej przez ciąg funkcji logistycznych')
plt.legend()
plt.grid(True)
plt.show()
```



Ćwiczenie.

Zauważmy, że funkcje przybliżające są gładsze (lepsze?) od przybliżanej. Ale te pojęcia (*przybliżanie*, *lepsze*) są na razie intuicyjne. W naukach ścisłych to niedpouszczalne. Zajmiemy się w dalszej części wykładu (i kolejnych) sprecyzowaniem ich. A na razie:

> 1. Rozważyć inne podobne ciągi, np. oparte o $f_1(x) = arctg(x)$.
>
> 2. Przybliżyć funkcje ReLu w uczeniu maszynowym: $g(x) = \max\{0, x\}$.

Warto dodać, że takie funkcje, ale okresowe, przybliżymy też na kolejnym wykładzie inna klasą funkcji - wielomianami trygonometrycznymi...

Zbieżność.

Powiemy, że szereg jest zbieżny w punkcie $x \in D$ jeśli zbieżny jest szereg liczbowy $\sum_{n=1}^{\infty} f_n(x)$ (inaczej, zbieżny jest ciąg sum częściowych $S_n(x)$).

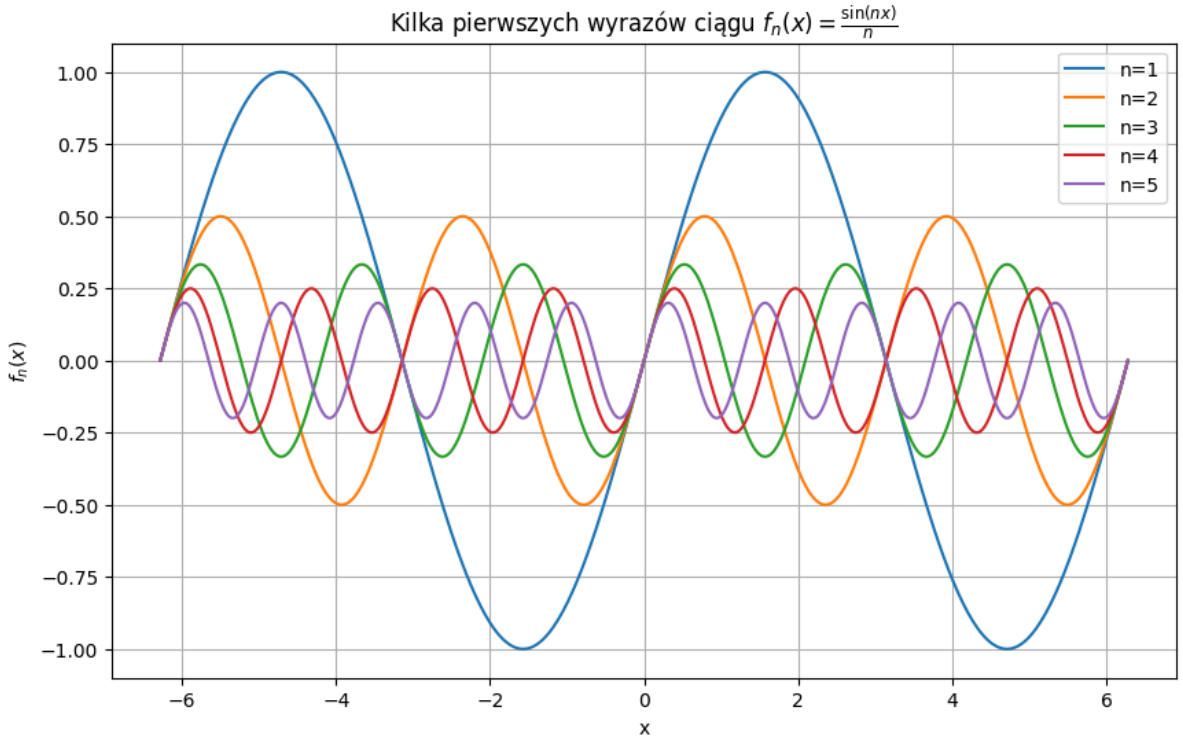
Formalnie, $S(\cdot)$ jest funkcją określoną na podzbiorze D (być może pustym), składających się z takich punktów x, dla których szereg liczbowy $\sum_{n=1}^{\infty} f_n(x)$ jest zbieżny.

W takim punkcie, $S(x)$ przyjmuje wartość równą sumie tego szeregu. Powyższy podzbiór dziedziny D nazywamy *obszarem zbieżności* szeregu funkcyjnego.

Ciąg funkcyjny zbieżny:

Ciąg $f_n(x) = \frac{\sin nx}{n}$, $f(x) \equiv 0, x \in \mathbb{R}$.

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

def fn(x, n):
    return np.sin(n*x) / n

# Zakres wartości x
x_values = np.linspace(-2*np.pi, 2*np.pi, 400)

# Wybierz kilka pierwszych wartości n
n_values = [1, 2, 3, 4, 5]

# Tworzenie wykresu dla każdej wartości n
plt.figure(figsize=(10, 6))
for n in n_values:
    plt.plot(x_values, fn(x_values, n), label=f'n={n}')

plt.title('Kilka pierwszych wyrazów ciągu $f_n(x) = \\frac{\\sin(nx)}{n}$')
plt.xlabel('x')
plt.ylabel('$f_n(x)$')
plt.legend()
plt.grid(True)
plt.show()
```



To tylko zbieżność punktowa? **Proszę to sprawdzić na ćwiczeniach...**

Przykład.

Szereg geometryczny $\sum_{n=0}^{\infty} x^n$ jest **formalnie** określony na całej prostej $\mathbb{R}$.

Ale obszarem zbieżności jest przedział $(-1, 1)$. Szereg jest w tym obszarze zbieżny bezwzględnie, ale nie jednostajnie. Faktycznie, ustalmy $\epsilon = 1$ i przypuśćmy, że znaleźliśmy n_0 jak w definicji jednostajnej zbieżności. Wtedy

$$|S(x) - S_{n_0}(x)| = \left| \sum_{k=n_0}^{\infty} x^k \right| = \left| \frac{x^{n_0}}{1-x} \right|.$$

Rozwiązując nierówność

$$\left| \frac{x^{n_0}}{1-x} \right| < \epsilon$$

otrzymujemy łatwo, że $|x|$ musi być mniejsze od $\eta = \sqrt[n_0]{\epsilon}$, a to jest liczba ostro mniejsza od 1. Zatem warunek nie jest spełniony dla $x \in (-1, -\eta) \cup (\eta, 1)$.

Granica jednostajnie zbieżnych ciągów funkcyjnych.

Geometrycznie warunek zbieżności jednostajnej oznacza, że w dowolnym pasie o brzegach $y = f(x) \pm \varepsilon$ leżą prawie wszystkie krzywe $y = f(x)$.

W związku z tym ciąg np. ciąg $f_n(x) = x^n$ w przedziale $0 \leq x < 1$ nie jest zbieżny jednostajnie do funkcji f, bo w pobliżu punktu $x = 1$ krzywe $y = x^n$ nie leżą dowolnie blisko linii prostej $y = 0$.

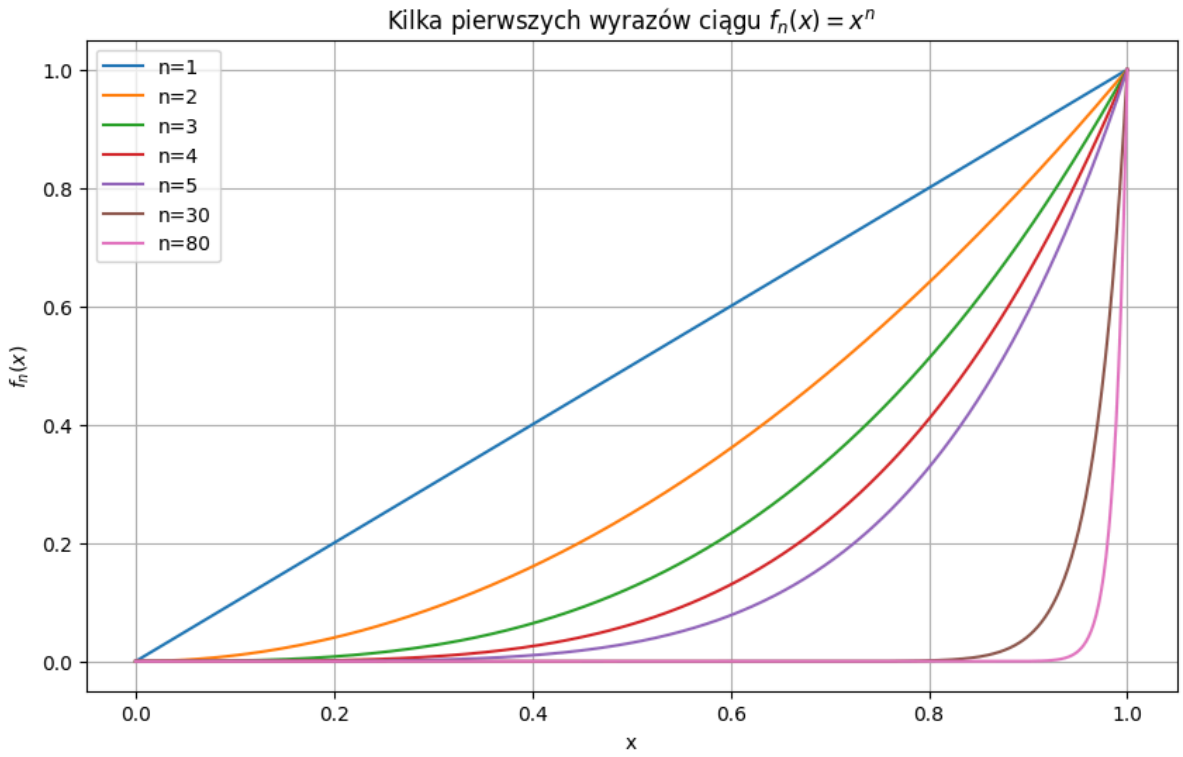
Ciągłość granicy.

Powstaje pytanie: czy funkcja graniczna ciągu (lub suma szeregu) funkcji ciągłych jest funkcją ciągłą? Odpowiedź może być przecząca, bo np. ciąg funkcji $f_n(x) = x^n$, $n = 1, 2, \ldots$ w przedziale $[0, 1]$. Jest to ciąg geometryczny, więc zbieżny do zera dla $0 \leq x < 1$ oraz do 1 dla $x = 1$. Oznaczając więc $f(x) = 0$ dla $0 \leq x < 1$, $f(1) = 1$, mamy $\lim_{n\to\infty} f_n(x) = f(x)$ dla każdego $x \in [0, 1]$.

Można więc na tyle wzmocnić definicję zbieżności ciągu funkcyjnego, aby z ciągłości wyrazów tego ciągu wynikała ciągłość funkcji granicznej. Podajemy najpierw zwykłą definicję zbieżności ciągu funkcji (f_n) w każdym punkcie osobno.

In [6]:
```python
import numpy as np
import matplotlib.pyplot as plt

def fn(x, n):
    return x**n

# Zakres wartości x
x_values = np.linspace(0, 1, 800)

# Wybierz kilka pierwszych wartości n
n_values = [1, 2, 3, 4, 5, 30, 80]

# Tworzenie wykresu dla każdej wartości n
plt.figure(figsize=(10, 6))
for n in n_values:
```

```
    plt.plot(x_values, fn(x_values, n), label=f'n={n}')

plt.title('Kilka pierwszych wyrazów ciągu $f_n(x) = x^n$')
plt.xlabel('x')
plt.ylabel('$f_n(x)$')
plt.legend()
plt.grid(True)
plt.show()
```



Zbieżność punktowa, ale niejednostajna.

Ciąg

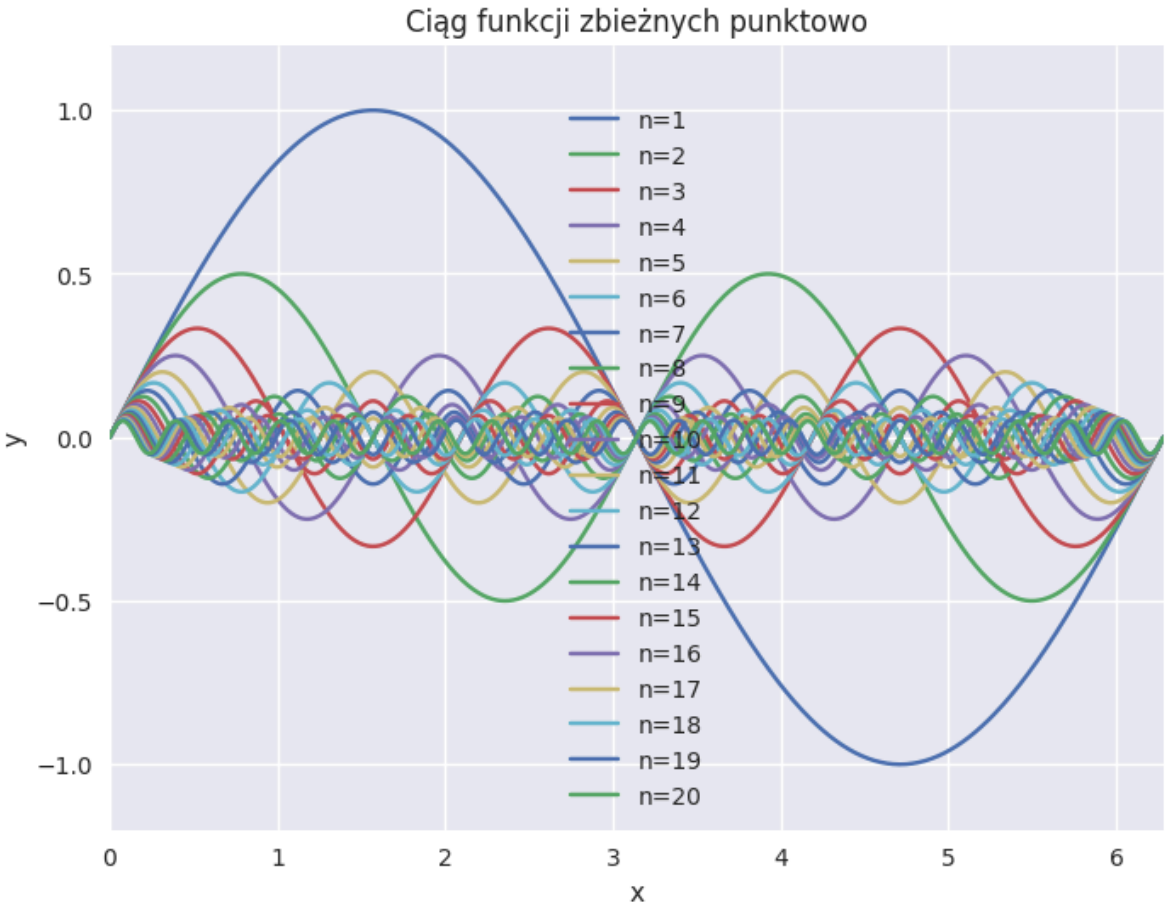
$$f_n(x) = x^n, \quad x \in [0,1]$$

$f(x) = 0, x \in [0,1), f(1) = 1.$

To tylko zbieżność punktowa! Zauważmy, że f nie jest ciągła, a wszystkie funkcje f_n - tak.

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

# Ustawienia wykresu
#plt.style.use('seaborn')
fig, ax = plt.subplots(figsize=(8, 6))
ax.set(xlim=(0, 2*np.pi), ylim=(-1.2, 1.2),
       xlabel='x', ylabel='y',
       title='Ciąg funkcji zbieżnych punktowo')

# Funkcja ciągu
def fn(x, n):
    return np.sin(n*x)/n

# Wartości x dla których rysowany jest wykres
x = np.linspace(0, 2*np.pi, 500)
```

```
# Rysowanie ciągu funkcji
for n in range(1, 21):
    y = fn(x, n)
    ax.plot(x, y, label=f'n={n}')

# Legenda i wyświetlenie wykresu
ax.legend()
plt.show()
```



> **Uwaga:** w informatyce będą też rozważane funkcje f_n, które nie będą "przybliżały" tych wyjściowych funkcji f nawet w sensie zbieżnosci punktowej. Celem może być *dokładne* uzyskanie $f(x_i) = f_n(x_i)$ dla pewnego zbioru punktów x_i z dziedziny, ale ważniejsze będzie wybranie tych funkcji z pewnej klasy (np. wielomianów stopnia 3) niż uzyskanie zbieżności punktowej w całejk dziedzinie. To zadania interpolacji - o której też oczywiście powiemy, a o tej uwadze warto pamiętać przy omawianiu efektu Rungego...

Kryterium zbieżności jednostajnej.

Na początek przedstawimy twierdzenie, które w wielu przypadkach ułatwia badanie jednostajnej zbieżności.

Twierdzenie. (o jednostajnej zbieżności ciągu funkcyjnego).

> Niech (f_n) będzie ciągiem funkcji określonych w niepustym zbiorze A i o wartościach rzeczywistych lub zespolonych,

oraz niech f będzie funkcją określoną w A o wartościach rzeczywistych lub zespolonych. Ciąg (f_n) jest jednostajnie zbieżnym w A do funkcji f wtedy i tylko wtedy , gdy $\sup_{x\in A} \mid f_n(x) - f(x) \mid \longrightarrow 0$, przy $n \to \infty$.

Warunek (kryterium) Cauchy'ego.

Jak zazwyczaj - bez zbadania zbieżności punktowej oraz - co gorsza - znalezieniu sum punktowych (co na razie jest BARDZO trudne) mamy problem z wyznaczenien $f(x)$. To oznacza, że sprawdzanie zbieżności jednostajnej zarówno z definicji, jak i kryterium zbieżności jest trudne. Rozwiązaniem jest procedura znana z ciągów liczbowych - warunek Cauchy'ego, gdzie znajomość $f(x)$ nie jest konieczna.

Twierdzenie.

> Ciąg funkcji $\{f_n(x)\}$ jest jednostajnie zbieżny w zbiorze A wtedy i tylko wtedy gdy dla każdej liczby $\varepsilon > 0$ istnieje wskaźnik $N(\varepsilon)$ taki, że dla wszystkich x zbioru A mamy
>
> $$|f_n(x) - f_m(x)| \leq \varepsilon \quad \text{dla} \quad n, m \geq N(\varepsilon).$$

Skupimy się jednak na **szeregach funkcyjnych**, bo w informatyce są one niezbędne, a nasz czas jest ograniczony...

Jednak nasz szereg geometryczny spełnia w swoim obszarze zbieżności inny ważny warunek, tzw. zbieżności niemal jednostajnej:

> Definicja. Szereg funkcyjny jest na rozważanym zbiorze E zbieżny niemal jednostajnie, jeśli dla dowolnego właściwego przedziału domkniętego $I = [a, b]$, $I \subset E$ jest on zbieżny jednostajnie na I.

Powyższy warunek ma sens tylko przy rozważaniu zbiorów E, które nie są przedziałami właściwymi domkniętymi (dla przedziałów właściwych domkniętych niemal jednostajna zbieżność jest zbieżnością jednostajną).

Sprawdzenie, że szereg geometryczny jest niemal jednostajnie zbieżny na $(0, 1)$ --- na ćwiczenia.

Twierdzenie Arzeli-Ascoli'ego.

> > Uwaga: w matematyce jednym z podstawowych wyników dotyczących zastosowań zbieżności ciągów jest twierdzenia Arzeli-Ascoli'ego mówiące, że:

Twierdzenie Arzeli-Ascoli'ego Jeżeli (f_n) jest ciągiem funkcji rzeczywistych określonych na przedziale zwartym, który jest wspólnie ograniczony i jednakowo ciągły (tzn. rodzina $\{f_n\colon n \in \mathbb{N}\}$ jest jednakowo ciągła), to zawiera on podciąg zbieżny jednostajnie.

Założenie jednakowej ciągłości jest istotne – istnieje ciąg ograniczonych funkcji ciągłych $f_n\colon [0,1] \to \mathbb{R}$, który nie ma podciągu zbieżnego jednostajnie. Np. niech

$$f_n(x) = \frac{x^2}{x^2 + (1-nx)^2}$$

dla $0 \leqslant x \leqslant 1$ oraz $n \in \mathbb{N}$. Licznik i mianownik wyrażenia f_n są nieujemne, czyli $|f_n(x)| \leqslant 1$ (są więc wspólnie ograniczone na $[0,1]$). Poza tym

$$\lim_{n\to\infty} f_n(x) = 0$$

dla każdego $x \in [0,1]$, ale jednak $f_n\left(\frac{1}{n}\right) = 1$ dla $n = 1, 2, 3 \ldots$, więc żaden podciąg ciągu (f_n) nie jest zbieżny jednostajnie.

> **Przykład zastosowania w informatyce.**

Rozważmy problem z dziedziny przetwarzania obrazów. Załóżmy, że chcemy aproksymować pewną funkcję obrazu f, która jest funkcją ciągłą, przy użyciu szeregu funkcji f_n generowanych przez pewien algorytm kompresji obrazu. Chcemy udowodnić, że f_n zbiega do f w sensie jednostajnym, co oznacza, że kompresja obrazu będzie **dokładna dla wystarczająco dużych n.**

1. Jednostajna ograniczoność: możemy założyć, że wszystkie obrazy są normalizowane do wartości pikseli w przedziale $[0,1]$, co implikuje, że funkcje f_n są jednostajnie ograniczone przez 1.

2. Jednostajna ciągłość: załóżmy, że proces kompresji zachowuje pewien poziom regularności (gładkości) obrazu, co oznacza, że różnice między wartościami pikseli w bliskich sobie punktach są ograniczone. To oznacza, że f_n jest jednostajnie ciągła.

Zgodnie z lematem Arzeli-Ascoliego, jeśli mamy zbiór funkcji $\{f_n\}$, który jest jednostajnie ograniczony i jednostajnie ciągły, to istnieje podciąg f_{n_k}, który jest zbieżny jednostajnie do pewnej funkcji ciągłej. To oznacza, że możemy mieć pewność co do zbieżności naszych aproksymacji obrazu do funkcji oryginalnej w sensie jednostajnym, co jest kluczowe dla **stabilności i jakości procesu kompresji**.

W praktyce twierdzenie te może być stosowane do dowodu poprawności algorytmów kompresji i odtwarzania obrazów, zapewniając jednocześnie, że proces ten jest numerycznie stabilny i zachowuje jakość danych wizualnych.

Ciągłość granicy.

Twierdzenie.

> Jeżeli ciąg $\{f_n(x)\}$ jest jednostajnie zbieżny w A i funkcje $f_n(x)$ są funkcjami ciągłymi w punkcie x_0

zbioru A, to funkcja graniczna $f(x) = \lim_{n\to\infty} f_n(x)$ jest ciągła w punkcie x_0.

___W n i o s e k.___

> Jeżeli szereg $\sum_{n=1}^{\infty} f_n(x)$ jest jednostajnie zbieżny w zbiorze A, funkcje f_n są

ciągłe w punkcie x_0 zbioru A, to suma $s(x) = \sum_{n=1}^{\infty} f_n(x)$ jest funkcją ciągłą w punkcie x_0.

Kryteria zbieżności jednostajnej.

Najpierw dla szeregów o wyrazach nieujemnych.

Kryterium porównawcze Weierstrassa.

> Jeśli szereg liczbowy utworzony z ciągu (a_n) o wyrazach nieujemnych jest zbieżny,

oraz dla każdego $x \in E$ zachodzi nierówność $|f_n(x)| < a_n$, to szereg funkcyjny $\sum_{n=1}^{\infty} f_n(\cdot)$ jest bezwzględnie jednostajnie zbieżny na E.

Kryterium Dirichleta.

> Jeśli ciąg sum częściowych $S_n(\cdot)$ utworzony z ciągu funkcyjnego (f_n) jest wspólnie ograniczony, to znaczy jeśli istnieje stała M taka, że dla każdego n i {\bf każdego} $x \in E$ mamy $|S_n(x)| < M$, oraz jeśli ciąg funkcyjny (g_n) zbiega do zera monotonicznie i jednostajnie na E, to szereg
>
> $$\sum_{n=1}^{\infty} f_n(\cdot) g_n(\cdot)$$
>
> jest zbieżny jednostajnie na E.

Uwaga: Dla szeregów, gdzie wartości funkcji f_n są w przestrzeniach metrycznych mamy też inne interesujące typy zbieżności, ale ich omówienie zdecydowanie wykracza poza zakres naszego wykładu...

Kryterium Abela.

> Niech $(a_n(x))$ oraz $(b_n(x))$ będą ciągami funkcyjnymi określonymi w zbiorze A. Załóżmy, że
>
> (1) ciąg $(a_n(x))$ jest monotoniczny dla każdego $x \in A, \setminus a_n(x)) \geq 0$ w A oraz $a_n(x)$ jest ciągiem wspólnie ograniczonym w A ,
>
> (2) szereg $\sum_{n=1}^{\infty} b_n(x)$ jest jednostajnie zbieżny w A.
>
> Wtedy $\sum_{n=1}^{\infty} a_n(x) b_n(x)$ jest jednostajnie zbieżny w A.

Przykład 1.

Rozpatrzmy zbieżność szeregu $\sum_{n=1}^{\infty} \frac{1}{n^x}$ w $A = \{x : \ x \geq \delta > 1, \ x \in \mathbb{R}\}$.

Dla ciągu $f_n(x) = \frac{1}{n^x}$ zachodzi $\left|\frac{1}{n^x}\right| \leq \frac{1}{n^\delta}$.

Majoranta liczbowa ma postać $\sum_{n=1}^{\infty} \frac{1}{n^{\delta}}$, gdzie $\delta > 1$. Jest to szereg zbieżny jako uogólniony szereg harmoniczny rzędu większego od 1. Stąd na mocy kryterium Weierstrassa mamy, że szereg $\sum_{n=1}^{\infty} \frac{1}{n^{x}}$ jest zbieżny jednostajnie w zbiorze A. Jego sumę nazywamy **funkcją dzeta Riemanna** $\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{n^{x}}$.

> To tak "przy okazji": zastosowanie funkcji dzeta Riemanna (i całek!) w analizie asymptotycznej algorytmów. Przykładem takim jest **algorytm wyszukiwania w tablicy z haszowaniem z otwartym adresowaniem**. Haszowanie z otwartym adresowaniem to technika, w której wszystkie elementy są przechowywane bezpośrednio w tablicy haszującej. Gdy dochodzi do kolizji, algorytm próbuje znaleźć kolejny dostępny slot w tablicy za pomocą pewnej strategii próbkowania (np. liniowego, kwadratowego czy podwójnego haszowania).

Rozważmy więc analizę złożoności oczekiwanej liczby prób potrzebnych do wyszukania elementu w tablicy z haszowaniem z użyciem otwartego adresowania, gdy używamy liniowego próbkowania. Oczekiwana liczba prób T_{hit} potrzebnych do znalezienia elementu w tablicy, gdy element jest obecny, wynosi: $T_{\text{hit}} = \frac{1}{\alpha} \int_0^{\alpha} \frac{1}{1-x}\, dx$ (całka!!) Obliczamy tę całkę (:-)): $T_{\text{hit}} = \frac{1}{\alpha}(-\ln(1-x))|_0^{\alpha} = \frac{-\ln(1-\alpha)}{\alpha}$. Oczekiwana liczba prób T_{miss} potrzebnych do stwierdzenia, że element nie jest obecny w tablicy, wynosi więc $T_{\text{miss}} = \frac{1}{1-\alpha}$. Funkcja dzeta Riemanna odgrywa rolę w **analizie asymptotycznej** tej struktury danych poprzez powiązanie z szeregami harmonicznymi. Dla dużych m i n, w granicach, przybliżenia takie jak $\zeta(1+\alpha)$ dla małych α mogą być wykorzystywane do analizy oczekiwanych wartości takich jak T_{hit} czy T_{miss}, gdy α dąży do 1. *Więcj - na analizie algorytmów lub kryptografii...*

Przykład 2.

Rozpatrzmy tzw. szereg Fouriera (poświęcimy im wiele kolejnych wykładów...)

$$\sum_{n=1}^{\infty} \frac{\sin nx}{n}.$$

Przyjmując

$$a_n(x) = \frac{1}{n}, \quad b_n(x) = \sin nx$$

widzimy, że szereg $\sum_{n=1}^{\infty} \frac{\sin nx}{n}$ jest jednostajnie zbieżny w przedziale $[\delta, 2\pi - \delta]$ dla dowolnego $\delta > 0$.

Funkcje zespolone.

Wszystko, co zostało powiedziane o szeregach liczbowych i funkcyjnych stosuje się do szeregów zespolonych i funkcyjnych zespolonych. Jedyne różnice są następujące:

a) Szeregi o wyrazach nieujemnych są z definicji rzeczywiste, więc tu nie ma zmian (to dotyczy kryteriów porównawczego, całkowego, d'Alemberta, Cauchy'ego, Raabego i o zagęszczaniu. Symbol wartości bezwzględnej $|\cdot|$ trzeba interpretować jako moduł liczby zespolonej. Zbieżność bezwzględna dotyczy szeregu z nałożonymi na każdy wyraz modułami (a to już jest szereg rzeczywisty o wyrazach nieujemnych).

b) Twierdzenie Riemanna nie zawsze jest prawdziwe (nie zawsze można dostać, jako sumę szeregu przestawianego zespolonego, dowolnej liczby zespolonej, czy choćby rzeczywistej). Zbiór możliwych sum danego szeregu warunkowo zbieżnego jest na ogół bardzo trudny do opisania.

c) W kryterium Dirichleta ciąg (a_n) dopuszczamy zespolony, ale ciąg (b_n) pozostaje rzeczywisty.

d) W definicji szeregu funkcyjnego, f_n są funkcjami określonymi na $D \subset \mathbb{C}$ ($\mathbb{C}$ to zbiór liczb zespolonych) o wartościach zespolonych.

e) W definicji zbieżności jednostajnej $|\cdot|$ oznacza moduł.

f) W kryterium Dirichleta dla szeregów funkcyjnych jako (f_n) dopuszczamy ciąg funkcji zespolnych, ale (g_n) pozostaje ciągiem funkcji rzeczywistych.

g) Definicja zbieżności niemal jednostajnej na zbiorze $E \subset \mathbb{C}$ jest inna: na każdym zwartym podzbiorze (pojęcie z topologii; zwarte są na przykład koła domknięte) zbioru E mamy mieć zbieżność jednostajną.

h) W definicji szeregu potęgowego, zmienna x oraz współczynniki a i a_n są zespolone.

i) Wzory na promień zbieżności pozostają te same (oczywiście teraz występuje w nich moduł).

j) W twierdzeniu Cauchy'ego-Hadamarda szereg potęgowy jest zbieżny niemal jednostajnie na *kole otwartym* o środku w x_0 i promieniu r (czyli na $\{x : |x - x_0| < r\}$), nazywanym **kołem zbieżności**. Szereg jest rozbieżny poza kołem domkniętym, a na brzegu (czyli na okręgu) może być różnie (w niektorych punktach zbieżność, w innych rozbieżność).

Badanie zbieżności szeregów funkcyjnych na komputerze ma swoje ograniczenia. Oto kilka z nich:

1. Liczba wyrazów szeregu: aby obliczyć wartość nieskończonego szeregu funkcji, musimy obliczyć jego sumę przy użyciu skończonej liczby wyrazów. Jednakże, im większa liczba wyrazów, tym większe jest zapotrzebowanie na zasoby obliczeniowe i czas obliczeń. Może to prowadzić do problemów związanych z ograniczeniami pamięci lub czasu obliczeń.

2. Dostępność funkcji: niektóre szeregi funkcyjne mają skomplikowane wyrażenia, których nie można zaimplementować jako funkcji (np. w Pythonie). W takich przypadkach musimy poszukać innych sposobów na obliczenie wartości szeregu.

3. Właściwości numeryczne: wartości numeryczne zmiennych zmiennoprzecinkowych mają ograniczoną precyzję i dokładność. Dlatego, nawet jeśli obliczenia są przeprowadzane

poprawnie, to wynik może być niedokładny z powodu błędów numerycznych.
4. Ograniczenia biblioteki: biblioteki (takie jak SymPy w Pythonie) mogą mieć swoje ograniczenia w obsłudze niektórych typów funkcji lub szeregów funkcyjnych. W takich przypadkach konieczne może być napisanie własnej implementacji lub użycie innych narzędzi obliczeniowych.

5. Inne czynniki: istnieją również inne czynniki, takie jak zbieżność warunkowa, które mogą wpłynąć na wyniki obliczeń. Należy pamiętać, że każda metoda badania zbieżności szeregów funkcyjnych ma swoje ograniczenia, a wyniki powinny być interpretowane z ostrożnością i zawsze powinny być potwierdzone przez inne metody.

Badanie zbieżności szeregów funkcyjnych w Pythonie za pomocą bibliotek matematycznych}}, takich jak SymPy czy SciPy, jest ``w pełni'' możliwe. Jednak, jak w każdym narzędziu obliczeniowym, istnieją pewne ograniczenia i wyzwania, z którymi można się spotkać podczas badania zbieżności szeregów funkcyjnych.

Oto kilka potencjalnych ograniczeń w badaniu zbieżności szeregów funkcyjnych w **Pythonie**:

1. Skończona precyzja obliczeń: biblioteki matematyczne, takie jak SymPy czy SciPy, korzystają z precyzyjnych obliczeń numerycznych, ale w praktyce istnieje granica precyzji, którą można uzyskać. Dlatego niektóre szeregi funkcyjne mogą być trudne lub niemożliwe do obliczenia ze względu na niedostateczną precyzję.

2. Konieczność ustawiania granic: aby skutecznie badać zbieżność szeregów funkcyjnych, konieczne jest określenie dokładnych granic szeregów, takich jak granica wyrazów czy granica sumy częściowej. W niektórych przypadkach określenie tych granic może być trudne lub wymagać czasochłonnych obliczeń.

3. Ograniczenia pamięciowe: obliczanie szeregów funkcyjnych o dużej liczbie wyrazów może wymagać dużych zasobów pamięciowych. W przypadku braku wystarczającej ilości pamięci, obliczenia mogą ulec przerwaniu lub zwolnić.

4. Trudne do zbadania zbieżności: istnieją szeregi funkcyjne, dla których badanie zbieżności jest trudne lub niemożliwe, nawet za pomocą zaawansowanych narzędzi matematycznych. W takich przypadkach pożyteczne może być zastosowanie metod numerycznych do {\bf przybliżenia} wartości szeregu lub próby innych sposobów analizy szeregu funkcyjnego.

5. Wymagana znajomość matematyki: w celu skutecznego badania zbieżności szeregów funkcyjnych w Pythonie wymagana jest wiedza na temat matematyki i teorii szeregów. Wymagane jest również dobre zrozumienie bibliotek matematycznych, takich jak SymPy czy SciPy, aby móc wykorzystać ich możliwości w badaniu zbieżności szeregów funkcyjnych.

Przejścia graniczne - kontrprzykład.

Rozpocznijmy od kontrprzykładu. Jeśli

$$f(n,k) = \frac{nk}{n^2+k^2} \text{ dla } (n,k) \neq (0,0)$$

oraz $f(0,0) = 0$. Obliczmy granice:

$$\lim_{n\to\infty} f(n,k) = \lim_{n\to\infty} \frac{nk}{n^2+k^2} = \lim_{n\to\infty} \frac{k}{n+\frac{k^2}{n}} = 0,$$

ale to dla $n \neq k$ oraz analogicznie

$$\lim_{k\to\infty} f(n,k) = \lim_{k\to\infty} \frac{nk}{n^2+k^2} = \lim_{k\to\infty} \frac{n}{k+\frac{n^2}{k}} = 0$$

dla $k \neq n$. Czy wynika stąd, że to wszystko jedno w jakiej kolejności obliczmy granice?

Zobaczmy, kładąc w szczególności $n = k$ mamy

$$f(k,k) = \frac{k^2}{k^2+k^2},$$

czyli

$$\lim_{k\to\infty} f(k,k) = \frac{1}{2} \neq 0.$$

Pytanie: ___KIEDY?___

$$\lim_{k\to\infty}\lim_{n\to\infty} f(n,k) = \lim_{n\to\infty}\lim_{k\to\infty} f(n,k),$$

bo - jak widzimy nie zawsze tak jest, a to bardzo przydatna własność - o czym się przekonamy...

Twierdzenie o zamianie przejść granicznych.

Nie będziemy (na razie) poszukiwać optymalnych założeń, dla których poprzedni kontrprzykład nie mógłby zajść i oczekujemy, że:

$$\lim_{n\to\infty}\lim_{k\to\infty} f(n,k) = \lim_{k\to\infty}\lim_{n\to\infty} f(n,k).$$

Wynik pojawi się przy okazji omawiania funkcji wielu zmiennych, tu zaznaczmy tylko, że w świetle kontrprzykładu musimy uważać przy całkowaniu czy różniczkowaniu ciągów i szeregów funkcyjnych.

Pytanie: kiedy możemy zamienić kolejność przejść granicznych np. dla całkowania:

$$\lim_{n\to\infty}\lim_{k\to\infty} \sum_{i-1}^{k} f_n(\xi_i)\cdot\Delta x_i = \lim_{n\to\infty} \int_a^b f_n(x)\,dx$$

oraz

$$\lim_{k\to\infty}\lim_{n\to\infty} \sum_{i-1}^{k} f_n(\xi_i)\cdot\Delta x_i = \lim_{k\to\infty} \sum_{i-1}^{k} f(\xi_i)\cdot\Delta x_i = \int_a^b f(x)\,dx.$$

Musimy zbadać kiedy zamiana kolejności przejść granicznych jest możliwa. Dla pochodnej jest podobnie, z różnicą polegającą na ilorazie różnicowym w miejsce sumy Riemanna.

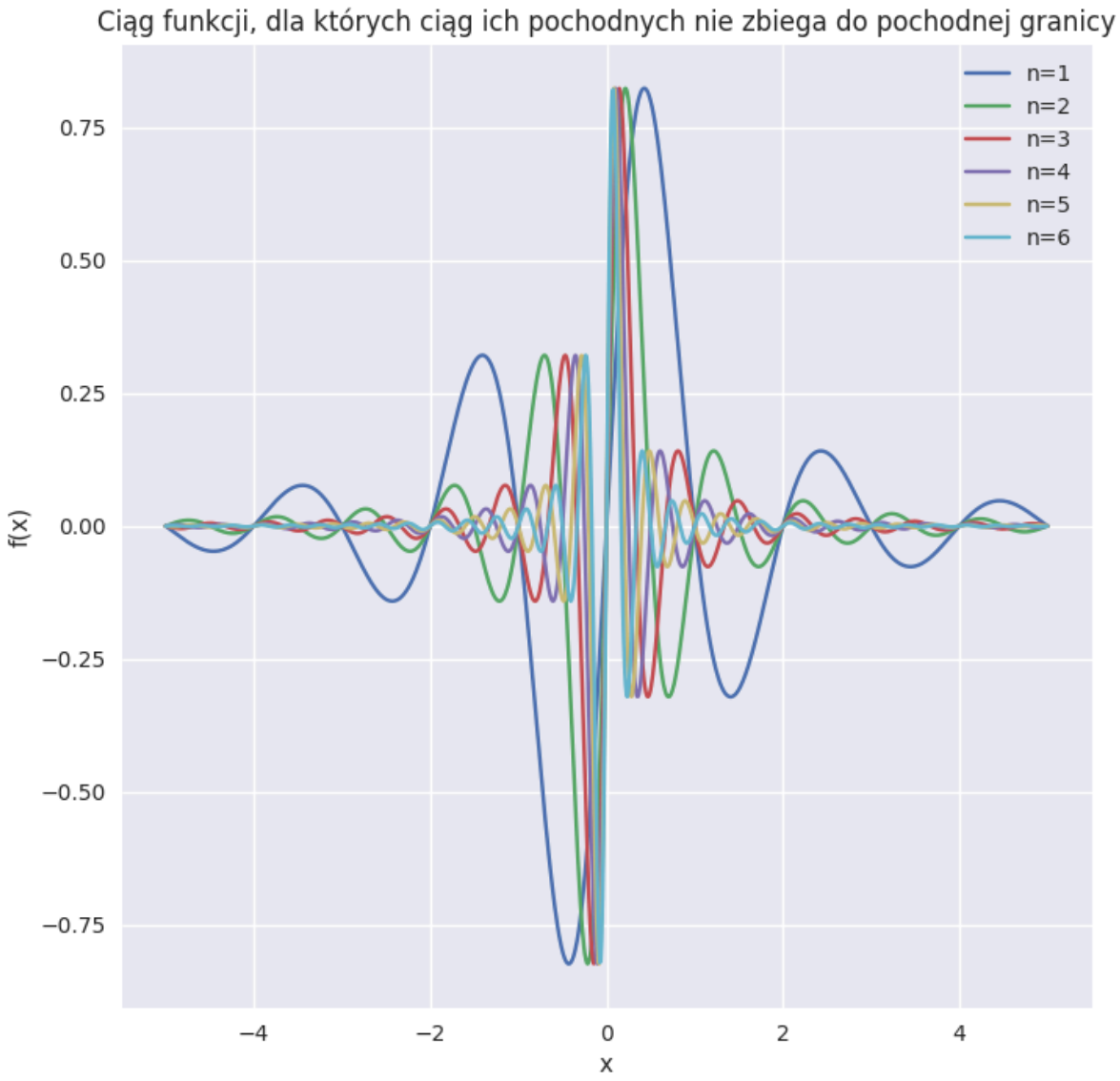
Różniczkowanie ciągów i szeregów funkcyjnych.

Zajmiemy się obecnie problemem, przy jakich założeniach z faktu, że ciąg funkcji różniczkowalnych $f_1, f_2, \ldots$ jest zbieżny do funkcji f wynika, że ciąg ich pochodnych jest zbieżny do pochodnej funkcji f.

Twierdzenie. (o różniczkowaniu ciągów funkcyjnych).

> Niech $(f_n(t))$ będzie ciągiem funkcyjnym określonym na przedziale (a, b), gdzie $0 < b - a < \infty$, takim, że:
>
> (1) ciąg $(f_n(x_0))$ jest zbieżny dla pewnego $x_0 \in (a, b)$,
>
> (2) f_n są różniczkowalne w (a, b),
>
> (3) ciąg pochodnych (f_n') jest jednostajnie zbieżny do funkcji g w (a, b).
>
> Wtedy
>
> (a) ciąg (f_n) jest jednostajnie zbieżny w (a, b),
>
> (b) funkcja $f(x) = \lim_{n\to\infty} f_n(x)$ jest różniczkowalna w (a, b),
>
> (c) $f'(x) = g(x)$ dla $x \in (a, b)$.

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

# Definicja ciągu funkcji
def f_n(x, n):
    return np.sin(np.pi * n * x) / (1 + n**2 * x**2)

# Definicja przedziału x
x = np.linspace(-5, 5, 1000)

# Rysowanie wykresu dla różnych wartości n
plt.figure(figsize=(8,8))
for n in range(1, 7):
    plt.plot(x, f_n(x, n), label=f'n={n}')
plt.legend()
plt.title('Ciąg funkcji, dla których ciąg ich pochodnych nie zbiega do pochodnej gr
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```



Wzory na ten ciąg funkcyjny:

$$f_n(x,n) = \frac{sin(\pi \cdot n \cdot x)}{1 + n^2 \cdot x^2}.$$

Każda funkcja w ciągu jest różniczkowalna, ale ciąg ich pochodnych nie zbiega do pochodnej granicy. Można to zaobserwować na wykresie, gdzie dla większych wartości n funkcje stają się wysoko oscylujące wokół osi $x = 0$, co sugeruje, że ich pochodne nie są zbieżne do pochodnej granicy. Ale dowód tego - samodzielnie (lub na ćwiczeniach!!).

Uwaga.

Powyższe twierdzenie jest również prawdziwe dla przedziału domkniętego $[a, b]$, przy czym różniczkowalność w przedziale $[a, b]$ oznacza różniczkowalność w (a, b), prawostronną różniczkowalność w punkcie a i lewostronną różniczkowalność w punkcie b.

Ćwiczenie: proszę sprawdzić zbieżności ciągów i ciągów ich pochodnych dla:

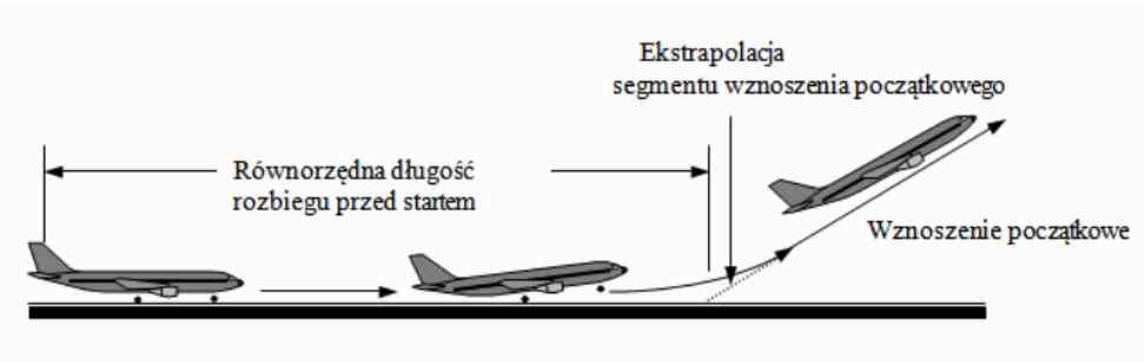
> (a) $f_n(x) = \frac{\sin n^2 x}{n}$,

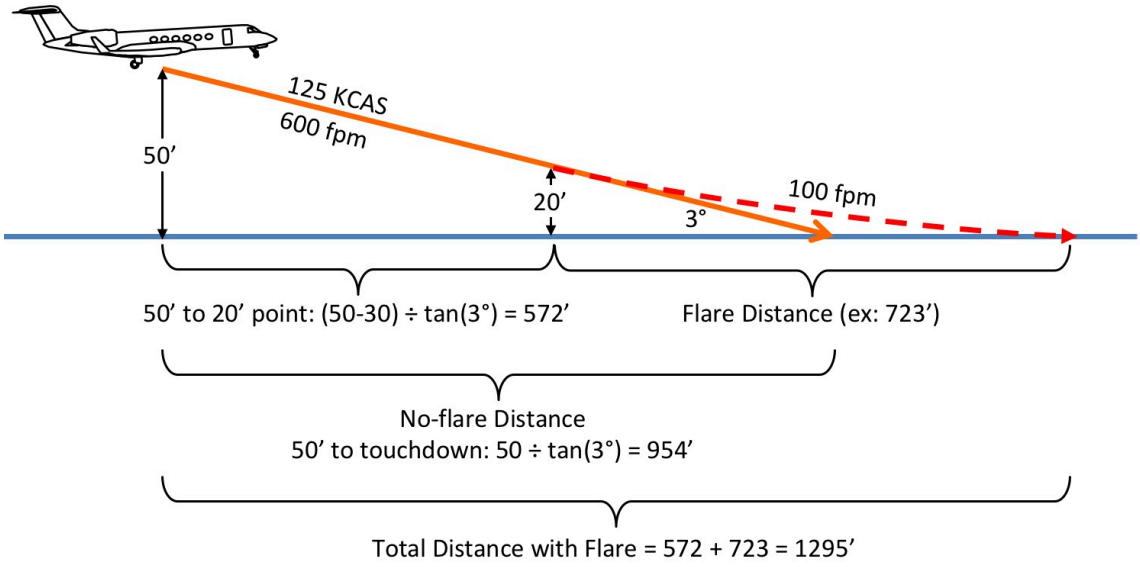
> (b) $f_n(x) = \frac{x}{1+n^2x^2}$.

Uwaga 2.

Takie własności jak omawiane ostatnio znajdą zastosowanie w interpolacji i ekstrapolacji: często będzie się oczekiwać znalezienia funkcji (z pewnej klasy), która ma zadane w ustalonych punktach (węzłach) wartości, ale - co ważne - ma również zadane w tych węzłach wartości pochodnej (interpolacja Hermite'a). Musimy kontrolować nie tylko ciągi funkcji, ale i ich pochodnych (czy całek, o czym za chwilę).

Trudno sobie wyobrazić, aby np. w zaplanowaniu punktu przyziemienia czy startu samolotu wyznaczyć jedynie jego położenie, a nie pochodną (kąt przyziemienia)...





Twierdzenie.

> Niech $(f_n(x))$ będzie ciągiem funkcyjnym określonym w przedziale (a,b), gdzie $0 < b-a < \infty$, takim że:
>
> (1^0) szereg $\sum_{n=0}^{\infty} f_n(x_0)$ jest zbieżny dla pewnego $x_0 \in (a,b)$,
>
> (2^0) f_n są różniczkowalne w (a,b),
>
> (3^0) szereg pochodnych $\sum_{n=0}^{\infty} f_n'(x)$ jest jednostajnie zbieżny do sumy $g(x)$ w (a,b).

> Wtedy
>
> (a) szereg $\sum_{n=0}^{\infty} f_n(x)$ jest jednostajnie zbieżny w (a,b)
>
> (b) funkcja $f(x)=\sum_{n=0}^{\infty} f_n(x)$ jest różniczkowalna w (a,b),
>
> (c) $f'(x)=g(x)$ dla $x\in(a,b)$, czyli $[\sum_{n=0}^{\infty} f_n(x)]'=\sum_{n=0}^{\infty} f_n'(x)$.

Całkowanie ciągów i szeregów funkcyjnych.

Teraz sformułujemy twierdzenie, które odpowiada nam na pytanie, kiedy można dokonać przejścia do granicy pod znakiem całki tj. kiedy
$\lim_{n\to\infty}\int_a^b f_n(x)\,dx=\int_a^b \lim_{n\to\infty} f_n(x)\,dx$, o ile funkcje f_n są ciągłe w $[a,b]$ i skończona granica $f(x)=\lim_{n\to\infty} f_n(x)$ istnieje dla każdego $x\in[a,b]$.

Twierdzenie. (o całkowaniu ciągów funkcyjnych).

> Jeżeli ciąg (f_n) funkcji ciągłych w $[a,b]$ jest zbieżny jednostajnie w $[a,b]$, to
>
> $$\lim_{n\to\infty}\int_a^b f_n(x)\,dx=\int_a^b \lim_{n\to\infty} f_n(x)\,dx.$$

Twierdzenie. (o całkowaniu szeregów funkcyjnych).

> Jeżeli szereg $\sum_{n=0}^{\infty} f_n(x)$ funkcji f_n ciągłych w $[a,b]$ jest jednostajnie zbieżny w $[a,b]$, to
>
> $$\sum_{n=0}^{\infty}\int_a^b f_n(x)\,dx=\int_a^b \sum_{n=0}^{\infty} f_n(x)\,dx.$$

Całkowanie ciągów i szeregów funkcyjnych.

Twierdzenie.

> Niech (f_n) będzie ciągiem funkcji ciągłych określonych na wspólnej dziedzinie D w postaci przedziału (właściwego lub nie), zbieżnym jednostajnie do funkcji granicznej f. Wtedy
>
> a) Funkcja f jest ciągła.
>
> b) Niech G_n będzie funkcją pierwotną dla f_n. Wtedy istnieją stałe C_n, takie że ciąg (F_n), gdzie $F_n=G_n+C_n$ zbiega niemal jednostajnie do funkcji F będącej pierwotną dla f.

Wniosek.

> Jeśli ciąg (f_n) zbiega niemal jednostajnie na przedziale D (właściwym lub nie, o dowolnych końcach), to granica f jest ciągła, funkcje pierwotne F_n (z odpowiednio dobranymi stałymi) są zbieżne niemal jednostajnie do funkcji F pierwotnej z f.

___Przykład.___

Bez zbieżności niemal jednostajnej twierdzenie zawodzi: granica nie musi być ciągła, a granica funkcji pierwotnych nie musi być pierwotną dla funkcji granicznej.

Niech $f_n = x^n$ na $[0,1]$. Te funkcje są zbieżne punktowo (ale nie jednostajnie) do funkcji

$$f(x) = \begin{cases} 0 \text{ dla } x \in [0,1) \\ 1 \text{ dla } x = 1 \end{cases}.$$

Jak widać, funkcja graniczna nie jest ciągła. Funkcje pierwotne F_n mają postać $F_n(x) = nx^{n-1}$. Te zbiegają punktowo do zera, ale tylko dla $x \in [0,1)$, a w $x = 1$ granica jest niewłaściwa ∞. Natomiast wzór $F(x) = \int_0^x f(t)\,dt$ produkuje funkcję równą zero na całym $[0,1]$. Tak więc równość dla funkcji pierwotnych nie zachodzi w $x = 1$.

Całkowanie szeregu funkcyjnego wyraz po wyrazie.

Wniosek.

> Jeśli szereg funkcyjny $\sum_{n=1}^{\infty} f_n$ zbiega niemal jednostajnie na przedziale D (właściwym lub nie,

o dowolnych końcach), to suma s jest ciągła, funkcje pierwotne $F_n(x) = \int_a^x f(t)\,dt$ tworzą szereg niemal jednostajnie zbieżny do funkcji $S(x) = \int_a^x s(t)\,dt$.

> Czyli
>
> $$\int_a^x \sum_{n=1}^{\infty} f_n(t)\,dt = \sum_{n=1}^{\infty} \int_a^x f_n(t)\,dt.$$
>
> > **Uwaga.** Koniecznie trzeba zauważyć, że podawane tu pojęcia i twierdzenia mają **natychmiastowe** zastosowania w informatyce, w tym m.in. w ramach całkowania numerycznego. Jeżeli funkcja ma rozwinięcie w szereg funkcyjny zbieżny jednostajnie na $[a,b]$ (np. w szereg potęgowy Taylora!)
> >
> > $$f = \sum_{n=1}^{\infty} f_n,$$
> >
> > to oczekiwalibyśmy
> >
> > $$\int_a^b f(x)\,dx = \sum_{n=1}^{\infty} \int_a^b f_n(x)\,dx \approx \sum_{n=1}^{N} \int_a^b f_n(x)\,dx.$$
> >
> > Teraz zauważmy, że jeżeli f_n są "łatwo całkowalne" np. to wielomiany (jak przy szeregu Taylora), to sumę po prawej stronie wyliczymy **dokładnie** i jedyne przybliżenie wynika z odrzucania reszty szeregu. Jeśli zaś to całki trudniejsze do obliczeń dokładnych, to zawsze możemy je obliczać numerycznie. I oczywiście dobieramy układ (f_n) tak, aby to

> > bylo łatwiejsze niz dla oryginalnej funkcji f. Mozna też wykorzystać ciągi funkcyjne zamiast szeregów.
> >
> > Takie metody całkowania bazują więc na aproksymacji funkcji podcałkowej (a nie interpolacji jak metody trapezów czy Simpsona). O metodach interpolacyjnych jeszcze powiemy szczgółowo na odrębnym wykładzie.

> > Jaka z tego korzyść? Po pierwsze - prostota metody, a po drugie w przypadku, gdy całki z f_n potrafimy obliczyć analitycznie (np. wielomiany), to mamy natychmiastowo **oszacowanie błędu** - np. z twierdzenia Leibnitz'a. Co więcej - ta metoda pozwala oszacować ile wyrazów trzeba zsumować w celu uzyskania żądanej precyzji!
> >
> > Oczywiście - możemy to też stosować do **całek niewłaściwych** - gdzie nie zadziałają metody oparte o sumy Riemanna!

> **Przykład.**

Zastosujmy metodę dla całki

$$I = \int_0^1 \frac{\sin x}{x} dx.$$

To właśnie całka niewłaściwa (ale jest zbieżna - polecam sprawdzić samodzielnie). Rozwińmy funkcję $\sin x$ w szereg Taylora (właściwie McLaurine'a - pozostawiam do samodzielnego obliczenia pochodne):

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

Następnie dzielimy ten szereg przez x:

$$\frac{\sin(x)}{x} = \frac{x - \frac{x^3}{6} + \frac{x^5}{120} - \cdots}{x} = 1 - \frac{x^2}{6} + \frac{x^4}{120} - \cdots$$

A teraz całkujemy każdą z funkcji f_n:

$$\int_0^1 1 \, dx, \quad \int_0^1 \frac{x^2}{6} dx, \quad \int_0^1 \frac{x^4}{120} dx, \ldots$$

Stąd (mamy zbieżność jednostajną! - istotność tego założenia pokażemy za chwilę na kontrprzykładzie, gdy nie będzie spełnione)

$$I = 1 - \frac{1}{3 \cdot 3!} + \frac{1}{5 \cdot 5!} - \ldots + (-1)^n \frac{1}{(2n+1) \cdot (2n+1)!} + \ldots$$

To teraz oszacujmy $I - S_N$. Z twierdzenia Leibnitz'a:

$$|I - S_N| < |S_{N+1}| = \frac{1}{(2n+3) \cdot (2n+3)!}.$$

Zauważmy, że już dla N=5 mamy błąd rzędu $\frac{1}{11 \cdot 11!}$, a więc $\approx 2 \cdot 10^{-9}$.

> > Zadanie: napisać program (dobry!) do szacowania ile składników sumujemy, aby uzyskać zadną precyzję ε_1.

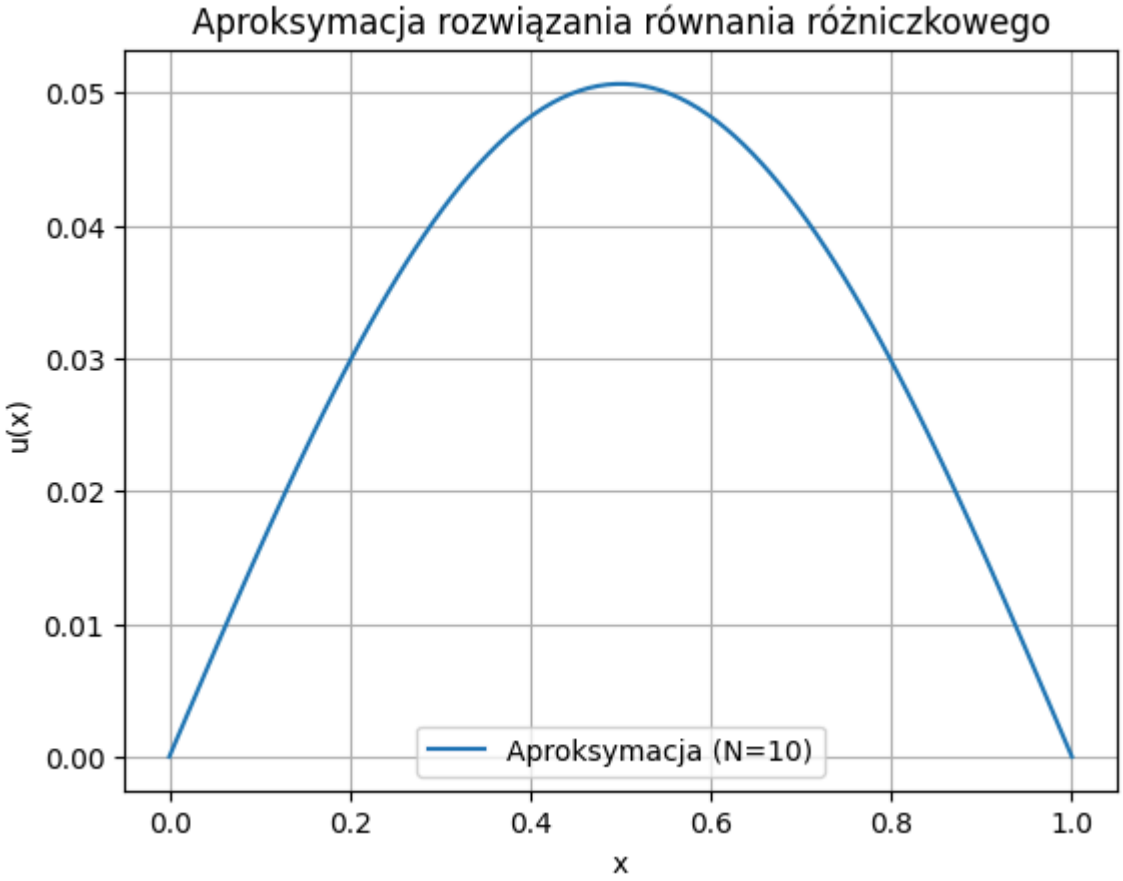
> W informatyce takie rozumowanie prowadzi do szacownia błędu nazywanego **błędem metody**, a w tym konkretnym przypadku to **bład obcięcia**. Pamiętajmy, że to bardzo istotny element informatyki - z reguły ma podstawy matematyczne, ale stanowi problem "czysto informatyczny".

> .....

> > **Uwaga:** o jeszcze bardziej istotnym zastosowaniu ciągów funkcyjnych oraz możliwości ich całkowania czy różniczkowania - np. w rozwiązywaniu numerycznym lub analitycznym *równań różniczkowych* na pewno ... nie zdążymy powiedzieć! Pod koniec naszych wykładów powiemy sporo o takich równaniach, w tym o przybliżaniu ich rozwiązań poprzez ciągi i szeregi funkcyjne, ale to temat na odrębny przedmiot. Proszę jednak zwrócić uwagę na ich stosowanie w tamtych algorytmach (np. metoda Galerkina).

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

# Definiujemy funkcję prawostronną
def f(x):
    return np.sin(np.pi * x)  # Możemy wybrać dowolną funkcję

# Funkcja do obliczenia współczynników c_n
def calculate_coefficients(N):
    c = np.zeros(N)
    for n in range(1, N+1):
        # Całkujemy prawą stronę równania, aby uzyskać c_n
        integral_value = np.trapz(f(x) * np.sin(n * np.pi * x), x)
        c[n-1] = integral_value / (n**2 * np.pi**2)
    return c

# Aproksymujemy rozwiązanie u(x)
def approximate_solution(x, c):
    u_N = np.zeros_like(x)
    for n, c_n in enumerate(c, start=1):
        u_N += c_n * np.sin(n * np.pi * x)
    return u_N

# Ustawienia
N = 10  # Liczba funkcji bazowych
x = np.linspace(0, 1, 100)  # Domena

# Obliczamy współczynniki
coefficients = calculate_coefficients(N)

# Aproksymujemy rozwiązanie
u_approx = approximate_solution(x, coefficients)

# Rysujemy wynik
```

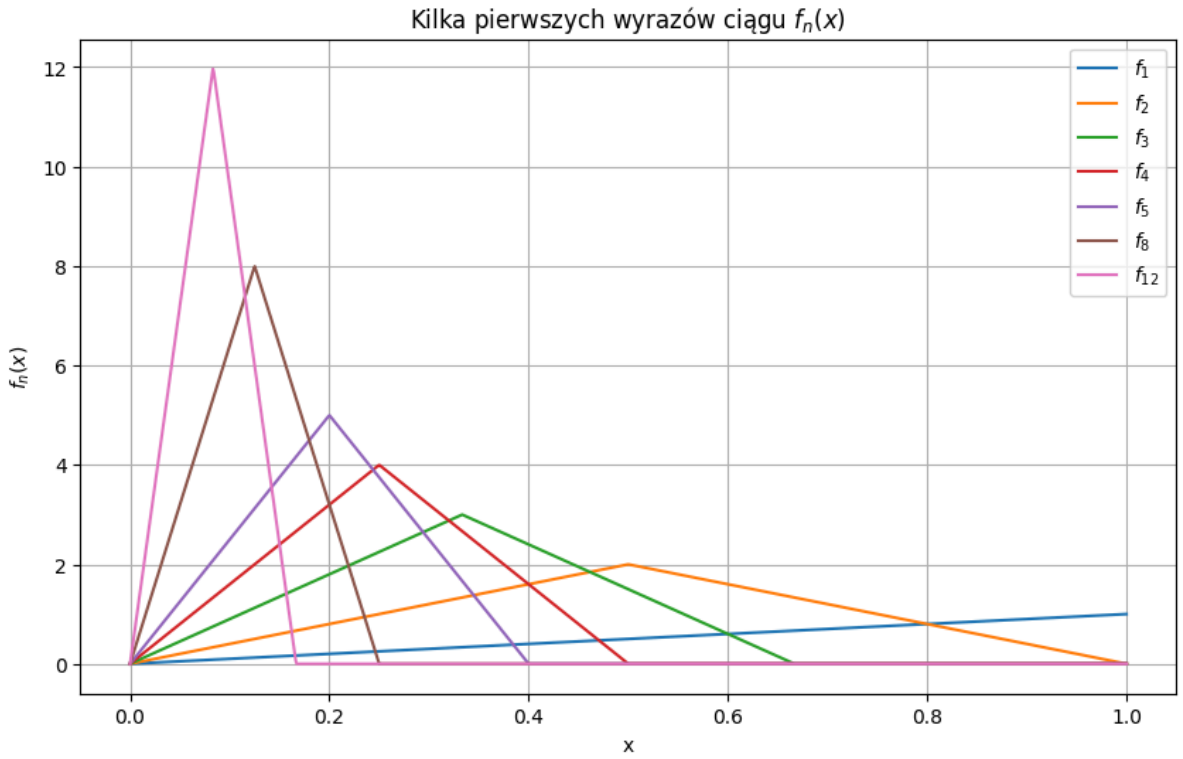
```
plt.plot(x, u_approx, label=f'Aproksymacja (N={N})')
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('Aproksymacja rozwiązania równania różniczkowego')
plt.legend()
plt.grid(True)
plt.show()
```



> A teraz ilustracja istotności twierdzenia o całkowaniu ciągów funkcyjnych:

In [12]:
```
import matplotlib.pyplot as plt
import numpy as np

# Definiujemy ciąg funkcji f_n
def f(n, x):
    if x <= 1/n:
        return n**2*x
    elif x <= 2/n:
        return 2*n-n**2*x
    else:
        return 0

# Tworzymy siatkę punktów na przedziale [0, 1]
X = np.linspace(0, 1, 1000)

plt.figure(figsize=(10, 6))
n_values = [1, 2, 3, 4, 5, 8, 12]
for n in n_values:
    Y = [f(n, x) for x in X]
    plt.plot(X, Y, label=f"$f_{{{n}}}$")
```

```
plt.title('Kilka pierwszych wyrazów ciągu $f_n(x)$')
plt.xlabel('x')
plt.ylabel('$f_n(x)$')
plt.legend()
plt.grid(True)
plt.show()
```



Ile wynoszą całki z funkcji f_n??

Proszę w kodzie odnaleźć wzór na ciąg funkcyjny, a także zastanowić się co jest granicą punktową (to widać na wykresie), czy jest zbieżny jednostajnie i czy jest zbieżny ciąg całek z f_n? Jeśli tak to czy do całki z f? SAMODZIELNIE!

Różniczkowanie ciągów i szeregów funkcyjnych.

Tym razem do "wejścia z pochodną pod granicę'' potrzebna jest (niemal) jednostajna zbieżność pochodnych.

Twierdzenie.

> Niech (f_n) będzie ciągiem funkcji ciągłych i różniczkowalnych określonych na wspólnej dziedzinie D w postaci przedziału (właściwego lub nie), takim że
>
> a) funkcje f_n zbiegają punktowo na D (do jakiejś funkcji f),
>
> b) funkcje pochodne (f_n') są ciągłe i zbiegają niemal jednostajnie do funkcji granicznej (ciągłej) g.
>
> Wtedy
>
> 1. ciąg (f_n) zbiega do f niemal jednostajnie (a zatem f jest ciągła),
> 2. funkcja graniczna f jest różniczkowalna na D i $f' \equiv g$.

___Przykład.___

Bez zbieżności niemal jednostajnej pochodnych twierdzenie zawodzi: granica pochodnych nie musi być pochodną dla funkcji granicznej. Można narysować przykład na $[0,1]$, w którym funkcje pochodne f_n' zbiegają (niejednostajnie) do funkcji Dirichleta, a funkcje f_n zbiegają do zera i to jednostajnie.

Wniosek. [Różniczkowanie szeregu funkcyjnego wyraz po wyrazie]

Jeśli szereg funkcyjny $s = \sum_{n=1}^{\infty} f_n$ zbiega punktowo na przedziale D (właściwym lub nie, o dowolnych końcach) oraz szereg utworzony z ciągłych funkcji pochodnych $\sum_{n=1}^{\infty} f_n'$ jest zbieżny niemal jednostajnie, to po pierwsze szereg $\sum_{n=1}^{\infty} f_n$ zbiega niemal jednostajnie, co za tym idzie jego suma s jest funkcją ciągłą, a po drugie suma szeregu pochodnych $\sum_{n=1}^{\infty} f_n'$ jest równa pochodnej funkcji s. Czyli

$$\left(\sum_{n=1}^{\infty} f_n\right)' = \sum_{n=1}^{\infty} f_n'.$$

O różniczkowaniu szeregów: obliczamy sumy szeregów.

Przykład. Niech

$$f(x) = \frac{1}{1-x^3}.$$

Z jednej strony obliczmy pochodną, a z drugiej rozwińmy w szereg potęgowy. *To zadanie samodzielne! Nie zapominajmy o promieniu zbieżności! Jak się ma koło zbieżności do dziedzeny funkcji f??*

Czyli otrzymujemy:

$$f'(x) = \frac{0 \cdot (1-x^3) - 1 \cdot (-3x^2)}{(1-x^3)^2} = \frac{3x^2}{(1-x^3)^2}$$

Dlatego pochodna funkcji $f(x)$ wynosi $f'(x) = \frac{3x^2}{(1-x^3)^2}$.

Poza tym (co każdy policzy - szereg potęgowy Taylora! w odpowiedniej dziedzinie)

$$\left(\frac{1}{1-x^3}\right)' = 3x^2 + 6x^5 + 9x^8 + \ldots = \sum_{n=1}^{\infty} 3n \cdot x^{3n-1}.$$

A więc w szczególności kładąc np. $x = \frac{1}{2}$ mamy:

$$f'\left(\frac{1}{2}\right) = \frac{3 \cdot \left(\frac{1}{2}\right)^2}{\left(1 - \left(\frac{1}{2}\right)^3\right)^2} = \frac{3 \cdot \left(\frac{1}{4}\right)}{\left(1 - \frac{1}{8}\right)^2} = \frac{3 \cdot \frac{1}{4}}{\left(\frac{7}{8}\right)^2} = \frac{3}{4} \cdot \frac{64}{49} = \frac{48}{49}.$$

Więc wartość pochodnej funkcji $f(x)$ w punkcie $x = \frac{1}{2}$ wynosi $\frac{48}{49}$.

$$\frac{48}{49} = 6 \cdot \sum_{n=1}^{\infty} \frac{n}{8^n}$$

$$\sum_{n=1}^{\infty} \frac{n}{8^n} = \frac{8}{49}.$$

Kolejne przykłady: na ćwiczeniach!

Kilka słów o aproksymacji.

W podsumowaniu wyraźnie podkreślmy co jest celem rozważań: daną funkcję zastępujemy pewnym ciągiem lub szeregiem funkcji, które są dla nas z pewnych względów korzystniejsze. Robimy to od razu na całej dziedzinie funkcji lub jej pewnych podprzedziałach (ale nie: punktach).

W takim przypadku mówimy o **aproksymacji funkcji** przez inne (np. wielomiany lub ich pewne klasy, wielomiany trygonometryczne itp.). Podawaliśmy przykład całkowania, gdzie korzystamy z wielomianów (szeregów potęgowych). Aby mówić o "przybliżaniu" f przez funkcje aproksymujące myślimy albo o odpowiednich metrykach, albo rodzajach zbieżności (np. punktowej). To bardzo użyteczne podejście i będziemy z tego często korzystać. Do tematu wrócimy przy okazji omawiania układów ortogonalnych funkcji - bo nadal dla nas otwarte jest pytanie: jakie funkcje powinny być zastosowane do aproksymacji? Nawet określenie "wielomiany" nie jest wystarczające, gdyż można (i warto) ograniczać się do pewnych ich klas (wielomiany Czebyszewa, Bernsteina itd.). Wyjaśnimy - dlaczego...

Nie pomylmy tego z sytuacją, którą także trzeba rozważać (i zrobimy to): gdy funkcja jest lub powinna być zadana dokładnie w pewnych punktach, a więc aproksymacja nie zadziała - nie kontrolujemy przecież wartości f_n w konkretnych punktach. Mamy więc inną możliwość: wybierać funkcje (na ogół wielomiany ustalonych rzędów), których wykresy przechodzą przez ustalone punkty: czyli **interpolacja**. Temu także poświęcimy odrębny wykład...

Twierdzenie Weierstrassa o aproksymacji.

> **Twierdzenie.**
>
> Dla każdej funkcji ciągłej określonej na przedziale domkniętym i przyjmującej wartości rzeczywiste istnieje **ciąg wielomianów** zbieżny jednostajnie do tej funkcji na tym przedziale.

Co ważne (dla informatyków), ten ciąg może składać się z tzw. **wielomianów Bersteina** (por. też krzywe Beziera!):

$$f_n(x) = \sum_{k=0}^{n} \binom{n}{k} f\left(\frac{k}{n}\right) x^k (1-x)^{n-k}.$$

Wielomiany bazowe Bernsteina służą do przedstawiania szeroko stosowanych w grafice komputerowej: krzywych Beziera, płatów Beziera i wywodzących się z nich innych rodzajów krzywych i powierzchni.

> Pamiętajmy też o tym na zajęciach z interpolacji (wkrótce)!!

Krzywe Beziera i wielomiany Bernsteina.

> Krzywym i płatom powierchniowym poświecimy nieco więcej uwagi na odrębnych zajęciach, głównie - oczywiście - w kontekście analizy matematycznej. Zagadnienia geometryczne dotyczace tych obiektów - dla zainteresowanych... Na razie - ich podsatwowe własności bazujące na funkcjach...

Krzywe Beziera i wielomiany Bernsteina są ze sobą ściśle powiązane, ponieważ krzywe Beziera są definiowane przy użyciu wielomianów Bernsteina jako funkcji bazowych.

Wielomiany Bernsteina są wykorzystywane do przybliżania dowolnej funkcji ciągłej f. Zapiszmy to następująco:

$$B_n(f;t) = \sum_{i=0}^{n} f\left(\frac{i}{n}\right) B_{i,n}(t),$$

gdzie: $B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$ to tzw. wielomianybazowe Bernsteina, $f\left(\frac{i}{n}\right)$ to wartości funkcji f w punktach $fracin.$

Krzywe Beziera mogą być postrzegane jako szczególny przypadek tego ogólnego przybliżenia funkcji ciągłej, gdzie wartościami funkcji f są punkty kontrolne krzywej. Zastępujemy wartości funkcji $f\left(\frac{i}{n}\right)$ przez punkty kontrolne $\mathbf{P}_i$.

Dlatego **krzywa Beziera stopnia** n jest wyrażona jako:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \mathbf{P}_i B_{i,n}(t),$$

gdzie: $\mathbf{P}_i$ to punkty kontrolne krzywej, a $B_{i,n}(t)$ to wielomiany bazowe Bernsteina.

Jak już wiemy wielomiany Bernsteina rzędu n są zdefiniowane jako

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}.$$

O tym warto pamiętać (ale najpierw omówimy sobie interpolację):

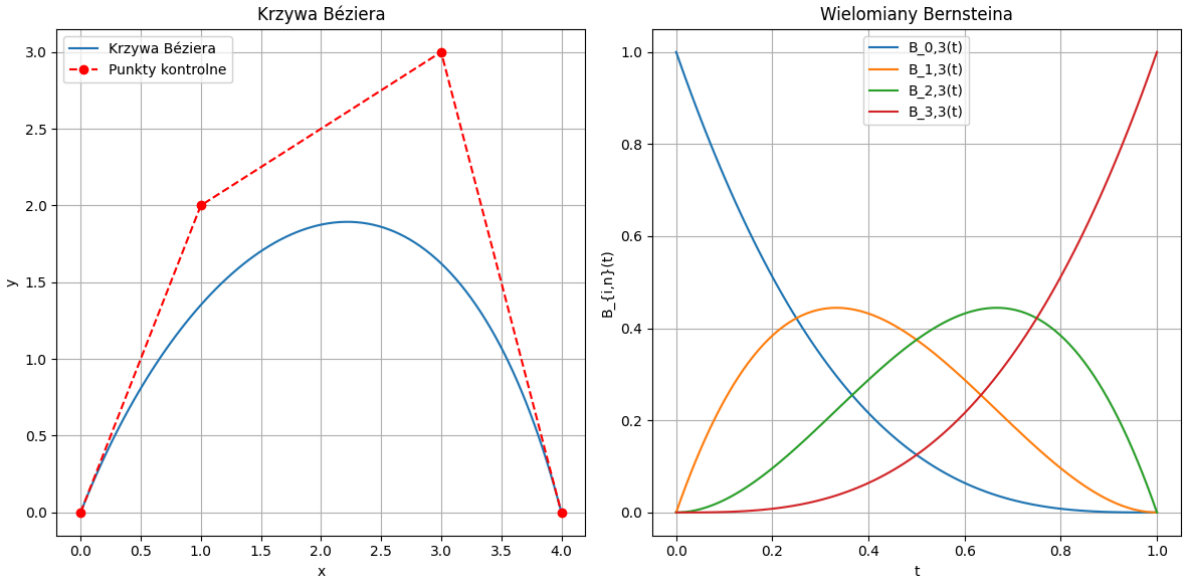
1. Wielomiany Bernsteina tworzą bazę funkcji dla krzywych Beziera. Każdy punkt na krzywej Beziera jest wyrażony jako kombinacja liniowa wielomianów Bernsteina i punktów kontrolnych.

2. Wielomiany Bernsteina mają właściwości interpolacyjne, które są wykorzystywane do określenia kształtu krzywej Beziera. Na przykład dla $t = 0$, krzywa Béziera zaczyna się w punkcie P_0, a dla $t = 1$ kończy się w punkcie P_n.

3. Wielomiany Bernsteina posiadają własności, które ułatwiają podział krzywych Beziera na mniejsze fragmenty, co jest przydatne w wielu algorytmach grafiki komputerowej, takich jak rekursywny algorytm de Casteljau. Dzięki tym właściwościom, krzywe Beziera są

szeroko stosowane w grafice komputerowej, projektowaniu CAD i innych dziedzinach związanych z modelowaniem geometrycznym.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

# Definicja wielomianów Bernsteina
def bernstein_poly(i, n, t):
    return comb(n, i) * (t ** i) * ((1 - t) ** (n - i))

# Definicja krzywej Béziera
def bezier_curve(control_points, num_points=100):
    n = len(control_points) - 1
    t = np.linspace(0, 1, num_points)
    curve = np.zeros((num_points, 2))
    for i in range(n + 1):
        bernstein = bernstein_poly(i, n, t)
        curve += np.outer(bernstein, control_points[i])
    return curve

# Punkty kontrolne dla krzywej Béziera
control_points = np.array([[0, 0], [1, 2], [3, 3], [4, 0]])

# Obliczanie krzywej Béziera
bezier_points = bezier_curve(control_points)

# Wykres krzywej Béziera
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(bezier_points[:, 0], bezier_points[:, 1], label='Krzywa Béziera')
plt.plot(control_points[:, 0], control_points[:, 1], 'ro--', label='Punkty kontroln
plt.title('Krzywa Béziera')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)

# Wykres wielomianów Bernsteina
plt.subplot(1, 2, 2)
t = np.linspace(0, 1, 100)
n = len(control_points) - 1
for i in range(n + 1):
    plt.plot(t, bernstein_poly(i, n, t), label=f'B_{i},{n}(t)')
plt.title('Wielomiany Bernsteina')
plt.xlabel('t')
plt.ylabel('B_{i,n}(t)')
plt.legend()
plt.grid(True)

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



Algorytm de Casteljau jest efektywną metodą obliczania punktów na krzywej Beziera. Jest również używany do podziału krzywej na mniejsze fragmenty. Algorytm ten działa na zasadzie rekurencyjnego mieszania punktów kontrolnych. Proszę do niego wrócić przy omawianiu interpolacji...

Na razie tylko kod do analizy jak działa algorytm de Casteljau i generuje krzywą Beziera na podstawie zadanych punktów kontrolnych.

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

# Algorytm de Casteljau
def de_casteljau(control_points, t):
    points = np.array(control_points)
    n = len(points)
    while n > 1:
        points = (1 - t) * points[:-1] + t * points[1:]
        n -= 1
    return points[0]

# Generowanie punktów na krzywej Béziera za pomocą algorytmu de Casteljau
def bezier_curve_de_casteljau(control_points, num_points=30):
    t_values = np.linspace(0, 1, num_points)
    curve = np.array([de_casteljau(control_points, t) for t in t_values])
    return curve

# Punkty kontrolne dla krzywej Béziera
control_points = np.array([[0, 0], [1, 2], [3, 1], [4, 0]])

# Obliczanie krzywej Béziera
bezier_points = bezier_curve_de_casteljau(control_points)

# Wykres krzywej Béziera
plt.figure(figsize=(12, 6))

plt.plot(bezier_points[:, 0], bezier_points[:, 1], label='Krzywa Béziera', color='b
plt.plot(control_points[:, 0], control_points[:, 1], 'ro--', label='Punkty kontroln
plt.title('Krzywa Béziera (algorytm de Casteljau)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

```
plt.grid(True)
plt.axis('equal')
plt.show()
```