Cursul 1

Corpul numerelor complexe

1. Introducere

Dezvoltarea istorică a gândirii matematice a urmărit îndeaproape evoluţia ideii de număr. Această evoluţie, exprimată succint prin şirul de incluziuni

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C},$$

nu a fost nicidecum liniară ci foarte sinuoasă, cu etape suprapuse sau chiar în ordine inversă.

Primele au apărut *numerele naturale* $1, 2, 3, \ldots$, urmate de *fracţii* şi *rapoarte*. In lumea antică deja apăruseră semne că numai acestea nu sunt suficiente; de exemplu, incomensurabilitatea diagonalei pătratului cu latura sa era cunoscută din vremea lui Pitagora.

Următoarea achiziţie a fost numărul *zero* care, plecând din India, a ajuns în Europa în jurul anului 1000, odată cu scrierea poziţională a numerelor, prin intermediul cărţilor lui Muhammad al-Khwarizmi, persanul de la numele căruia provine termenul de algoritm.

Numerele *negative* şi *imaginare* apar în acelaşi timp, în Italia secolului al XVI-lea, în parte tot sub influenţa surselor islamice şi bizantine. Aceste numere, cu denumiri care sugerează o acceptare doar formală a existenţei lor, s-au dovedit utile în rezolvarea ecuaţiilor algebrice. Astfel, unitatea negativă, -1, a fost definită ca fiind soluţia ecuaţiei $x + 1 = 0$, iar unitatea imaginară, $\sqrt{-1}$, ca soluţie a ecuaţiei $x^2 + 1 = 0$, chiar dacă printre numerele deja cunoscute nu exista niciunul care să verifice vreo una dintre aceste două ecuaţii.

Calculul formal cu *numere complexe*, adică cu expresii de forma $a + b\sqrt{-1}$ s-a dezvoltat prin analogie cu cel al numerelor iraţionale pătratice de forma $a + b\sqrt{2}$. În ciuda numeroaselor rezultate obţinute cu astfel de numere în secolele XVII-XVIII, matematicienii au avut reţineri cu privire la utilizarea lor. Îndoiala asupra "existenţei" acestor numere a dispărut odată cu apariţia unei interpretări geometrice foarte simple a lor, şi anume reprezentarea lor ca puncte în plan. Folosirea sistematică şi popularizarea reprezentării geometrice a numerelor complexe o datorăm lucrărilor lui Carl Friedrich Gauss (1777-1855).

Denumirile nefericite, care mai provoacă şi astăzi confuzii, de "numere reale" şi "numere imaginare" au fost introduse de René Descartes (1596 – 1650) pentru clasificarea rădăcinilor polinoamelor. Inţelegerea deplină, atât a numerelor reale cât şi a celor complexe, a avut loc concomitent, în secolul al XIX-lea, odată cu fundamentarea analizei matematice începută de Augustin-Louis Cauchy (1789 – 1857).

Mulţimea numerelor complexe $\mathbb{C}$, apărută ca instrument de lucru în rezolvarea ecuaţiilor polinomiale, s-a dovedit a fi chiar *cadrul natural* al acestui domeniu, dacă luăm în consideraţie, de exemplu, *Teorema fundamentală a algebrei*, enunţată în 1608 şi demonstrată două secole mai târziu, în 1806, care afirmă că orice polinom cu coeficienţi reali sau complecşi are măcar o rădăcină în $\mathbb{C}$. Dezvoltarea teoriei numerelor complexe a continuat cu trecerea de la funcţii polinomiale la funcţii oarecare de o variabilă complexă, cu studiul derivabilităţii şi integrabilităţii acestora, ajungându-se în final la stabilirea unui capitol fundamental în matematică, capitol numit *Analiză complexă*, prin analogie cu *Analiza matematică* a funcţiilor de variabilă reală.

Teoria funcţiilor de o variabilă complexă a pătruns, prin intermediul ecuaţiilor diferenţiale mai ales, în numeroase domenii din ştiinţă şi tehnică. De exemplu, în electrotehnică toate mărimile "sinusoidale" care caracterizează curentul electric sunt definite prin funcţii cu valori complexe. Sub presiunea numeroaselor lor utilizări, numerele complexe au ajuns şi în limbajele de programare, în *C++* sub formă de clase în biblioteca standard iar în *C#* şi Python chiar ca tip predefinit al limbajului.

In cursul pe care îl vom parcurge în acest semestru vom prezenta elementele de bază ale analizei complexe, cu accent pe aplicaţiile acestora în principal în grafica pe calculator, dar nu numai. In acest scop vom utiliza limbajul de programare Python în care numerele complexe sunt gata încorporate:

```
z = complex(2, 3)
print(f"z={z} are tipul {type(z)}")
# z=(2+3j) are tipul <class 'complex'>
```

In aplicaţiile grafice pe care le avem în vedere vom folosi numerele complexe sub această formă predefinită de limbaj, dar în acest curs introductiv, ca exerciţiu de programare în Python, vom prezenta structura algebrică a numerelor complexe prin implementarea unei clase proprii de lucru cu aceste numere, clasa `Complex` din fişierul `MyComplex.py` ataşat.

2. Forma algebrică a unui număr complex

Mulţimea numerelor complexe $\mathbb{C}$ se defineşte ca fiind mulţimea perechilor de numere reale

$$\mathbb{C} = \mathbb{R} \times \mathbb{R} = \{(x, y) \mid x, y \in \mathbb{R}\}$$

dotată cu două operaţii binare, adunarea:

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2), \quad \forall\, (x_1, y_1), (x_2, y_2) \in \mathbb{C},$$

şi înmulţirea

$$(x_1, y_1)(x_2, y_2) = (x_1x_2 - y_1y_2, x_1y_2 + x_2y_1), \quad \forall\, (x_1, y_1), (x_2, y_2) \in \mathbb{C}.$$

Teoremă. $(\mathbb{C}, +, \cdot)$ *este un corp comutativ. Prin corespondenţa* $x \to (x, 0)$, $\mathbb{R}$ *devine un subcorp al lui* $\mathbb{C}$.

Demonstraţie. Este evident că adunarea şi înmulţirea sunt operaţii interne. Asociativitatea şi comutativitatea adunării sunt imediate, $(0, 0)$ este element neutru iar opusul $-z$ al lui $z = (x, y)$ este $(-x, -y)$. Asociativitatea înmulţirii se verifică prin calcul, comutativitatea este evidentă, iar $(1, 0)$ este elementul unitate. Pentru un $z = (a, b) \neq (0, 0)$ găsirea inversului $z^{-1} = (x, y)$ revine la rezolvarea sistemului liniar

$$\begin{cases} ax - by = 1 \\ bx + ay = 0, \end{cases}$$

care, deoarece $\Delta = a^2 + b^2 \neq 0$, are soluţia unică $z^{-1} = \left(\frac{a}{a^2+b^2}, \frac{-b}{a^2+b^2}\right)$. Distributivitatea înmuţirii faţă de adunare se verifică uşor prin calcul, şi astfel am arătat că $(\mathbb{C}, +, \cdot)$ este un corp comutativ.

In sfârşit, este clar că aplicaţia $x \to (x, 0)$, definită pe $\mathbb{R}$ cu valori în $\mathbb{C}$ este injectivă şi are proprietăţile

$$(x_1, 0) + (x_2, 0) = (x_1 + x_2, 0)$$

$$(x_1, 0)(x_2, 0) = (x_1x_2, 0),$$

de unde rezultă imediat că imaginea sa $\{(x, 0) \mid x \in \mathbb{R}\} \subset \mathbb{C}$, este un subcorp al lui $\mathbb{C}$ izomorf cu corpul numerelor reale $\mathbb{R}$.

Observaţie. Fiind dat un număr complex $z = (x, y)$, convenim să numim numerele reale x şi y ca fiind *partea reală* şi, respectiv, *partea imaginară* a lui z, şi vom utiliza notaţiile $x = \operatorname{Re} z$, $y = \operatorname{Im} z$. Aplicaţiile $z \to \operatorname{Re} z$ şi $z \to \operatorname{Im} z$, definite pe $\mathbb{C}$ cu valori în $\mathbb{R}$ păstrează operaţia de adunare, adică

$$\operatorname{Re}(z_1 + z_2) = \operatorname{Re} z_1 + \operatorname{Re} z_2$$

$$\operatorname{Im}(z_1 + z_2) = \operatorname{Im} z_1 + \operatorname{Im} z_2,$$

dar nu şi operaţia de înmulţire.

A doua parte a teoremei de mai sus afirmă că mulţimea numerelor reale $\mathbb{R}$ poate fi identificată, prin izomorfismul de corpuri dat de *aplicaţia de scufundare* $x \in \mathbb{R} \to (x, 0) \in \mathbb{C}$, cu mulţimea numerelor complexe cu partea imaginară nulă

$$\mathcal{R} = \{z \,|\operatorname{Re} z = 0\} = \{(x, 0) \mid x \in \mathbb{R}\} \subset \mathbb{C}.$$

Aceasta înseamnă că, în loc de

$$\mathbb{R} \cong \mathcal{R} \subset \mathbb{C}$$

vom scrie

$$\mathbb{R} \subset \mathbb{C},$$

iar numerele complexe de forma $(x, 0)$ vor fi notate cu x atunci când nu există pericol de confuzie.

Mai mult, se verifică imediat că orice număr complex $z = (x, y)$ poate fi descompus sub forma

$$z = (x, y) = (x, 0) + (0, y) = (x, 0) + (0, 1)(y, 0),$$

şi, prin urmare, cu notaţia

$$i = (0, 1),$$

obţinem aşa numita *formă algebrică* a lui z:

$$z = x + iy,$$

unde, subliniem, x, y şi i sunt numere complexe. Din definiţia înmulţirii avem că

$$(0, 1)(0, 1) = (-1, 0) = -(1, 0),$$

relaţie care, cu convenţiile făcute, se scrie

$$i^2 = -1.$$

Aplicând regulile de calcul dintr-un corp comutativ, deducem mai departe că

$$(x_1 + iy_1)(x_2 + iy_2) = x_1x_2 + ix_1y_2 + iy_1x_2 + i^2y_1y_2$$

$$= x_1x_2 - y_1y_2 + i(x_1y_2 + y_1x_2),$$

justificând astfel de ce înmulţirea numerelor complexe a fost definită atât de complicat.

Să urmărim cum au fost implementate cele de mai sus în clasa `Complex` definită în modulul `MyComplex.py`.

Egalitatea $\mathbb{C} = \mathbb{R}\times\mathbb{R}$ se reflectă direct în structura unui obiect de tip `Complex`, care este format numai din două variabile instanţă, `x` şi `y`, iniţializate de constructorul obiectului:

```
class Complex:

    def __init__(self, x=0, y=0):
        assert isinstance(x, (int, float))
        assert isinstance(y, (int, float))
        self.x = x
        self.y = y
```

Aici, pentru depanare, se verifică dacă la apelare parametrii `x` şi `y` sunt de tip numeric, întreg sau flotant.

Definiţiile operaţiilor cu numere complexe apar, exact sub forma dată, la supraîncărcarea operatorilor corespunzători:

```
def __add__(self, other):
    # aduna self + other; z+w se traduce z.__add__(w)
    if isinstance(other, (int, float)):
        return Complex(self.x + other, self.y)
    if isinstance(other, Complex):
        return Complex(self.x + other.x, self.y + other.y)
    raise TypeError(f"Operatie nedefinita: {type(self)} + {type(other)}")


def __mul__(self, other):
    # returneaza self*other; z*w se traduce z.__mul__(w)
    if isinstance(other, (int, float)):
        return Complex(self.x * other, self.y * other)
    if isinstance(other, Complex):
        return Complex(self.x * other.x - self.y * other.y,
                       self.x * other.y + self.y * other.x)
    raise TypeError(f"Operatie nedefinita: {type(self)} * {type(other)}")
```

Definiţie. *Conjugatul* lui $z = x + iy$ este numărul complex $\bar{z} = a - ib$.

Propoziţie. Aplicaţia $z \to \bar{z}$ are următoarele proprietăţi imediate:

(i) $\overline{z_1 + z_2} = \overline{z_1} + \overline{z_2}$

(ii) $\overline{z_1 z_2} = \overline{z_1}\,\overline{z_2}$

(iii) $\overline{\bar{z}} = z$

(iv) $\bar{z} = z \Leftrightarrow z \in \mathbb{R}$

(v) $\operatorname{Re} z = \frac{z+\bar{z}}{2}$

(vi) $\operatorname{Im} z = \frac{z-\bar{z}}{2i}$

Verificăm de exemplu (ii)

$$\overline{z_1}\,\overline{z_2} = (x_1 - iy_1)(x_2 - iy_2) = (x_1x_2 - y_1y_2) - i(x_1y_2 + x_2y_1) = \overline{z_1 z_2}.$$

Observaţie. În corpul comutativ $(\mathbb{C}, +, \cdot)$ prin *scădere* se înţelege

$$z_1 - z_2 = z_1 + (-z_2),$$

iar prin *împărţire*

$$\frac{z_1}{z_2} = z_1 \cdot z_2^{-1}.$$

Este clar că din (i) şi (ii) urmează şi

(vii) $\overline{z_1 - z_2} = \overline{z_1} - \overline{z_2}$

$(viii)$ $\overline{\left(\frac{z_1}{z_2}\right)} = \frac{\overline{z_1}}{\overline{z_2}}$

(ix) $\overline{z^n} = \overline{z}^n$

Definiţie. *Modulul* lui $z = x + iy$ este numărul real $|z| = \sqrt{x^2 + y^2}$.
Propoziţie. Aplicaţia $z \to |z|$ are următoarele proprietăţi:

(i) $|z|^2 = z \cdot \bar{z}$

(ii) $|z| = |\bar{z}|$

(iii) $|z| = 0 \Leftrightarrow z = 0$

(iv) $|z_1 z_2| = |z_1||z_2|$

(v) $|z_1 + z_2| \leq |z_1| + |z_2|$

Verificăm de exemplu *inegalitatea triunghiulară* (v):

$$|z_1 + z_2|^2 = (z_1 + z_2)\overline{(z_1 + z_2)} = (z_1 + z_2)(\overline{z}_1 + \overline{z}_2) =$$

$$= z_1\overline{z}_1 + z_1\overline{z}_2 + z_2\overline{z}_1 + z_2\overline{z}_2 = |z_1|^2 + (z_1\overline{z}_2 + \overline{z_1\overline{z}_2}) + |z_2|^2$$

$$= |z_1|^2 + 2\,\mathrm{Re}\, z_1\overline{z}_2 + |z_2|^2 \leq |z_1|^2 + 2|z_1\overline{z}_2| + |z_2|^2$$

$$= |z_1|^2 + 2|z_1||z_2| + |z_2|^2 = (|z_1| + |z_2|)^2.$$

3. Forma trigonometrică a unui număr complex

Identificând $\mathbb{R}$ cu o dreaptă pe care se consideră o "axă a numerelor" Ox, urmează că $\mathbb{C} = \mathbb{R} \times \mathbb{R}$ se identifică cu un plan raportat la un sistem de coordonate cartezian xOy, asociind fiecărui punct M de coordonate (x, y) numărul complex $z = x + iy$, numit *afixul* lui M. Vom trece acum la coordonate polare.

Propoziţie. Fie $z \in \mathbb{C}$, $z \neq 0$. Există şi este unic $\theta \in (-\pi, \pi]$, numit *argumentul* numărului complex z şi notat $\arg z$, astfel încât

$$\cos\theta = \frac{\mathrm{Re}\, z}{|z|} \text{ şi } \sin\theta = \frac{\mathrm{Im}\, z}{|z|}.$$

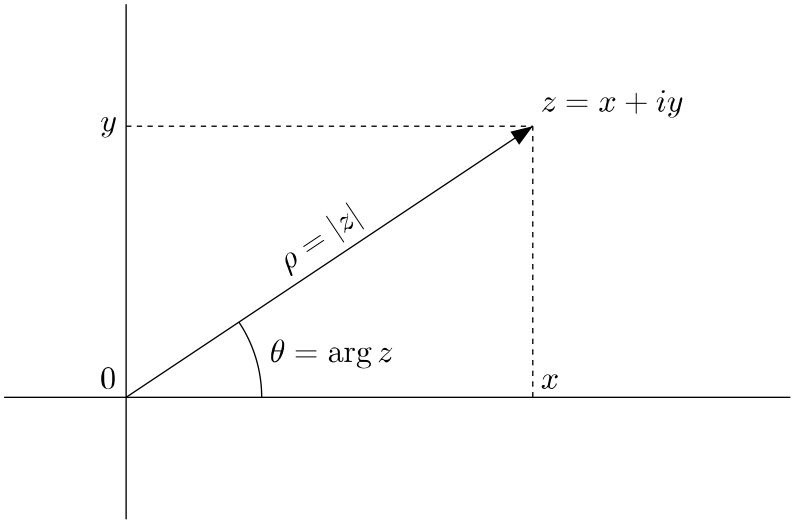
Demonstraţie. Fie $z = x + iy$, cu $|z| \neq 0$. Din $|z| = \sqrt{x^2 + y^2} \geq \sqrt{x^2} = |x|$ rezultă că $\frac{x}{|z|} \in [\,-1, 1]$, deci există

$$\theta_0 = \arccos\frac{x}{|z|} \in [\,0, \pi\,].$$

Avem $\sin^2\theta_0 = 1 - \cos^2\theta_0 = \left(\frac{y}{|z|}\right)^2$. Dacă $y \geq 0$ definim $\theta = \theta_0$, altfel $\theta = -\theta_0$. Unicitatea lui $\theta = \arg z$ decurge din echivalenţa

$$\begin{cases} \cos\theta = \cos\varphi \\ \sin\theta = \sin\varphi \end{cases} \Leftrightarrow \exists k \in \mathbb{Z} \text{ a. î. } \theta - \varphi = 2k\pi,$$

prin urmare, $\theta, \varphi \in (-\pi, \pi\,]$ implică $\theta = \varphi$.



Observaţie. Fie punctul $M(x, y)$ cu afixul $z = x + iy$. Raza $\rho = |z| \geq 0$ şi unghiul $\theta = \arg z \in (-\pi, \pi\,]$ reprezintă chiar coordonatele polare ale lui M. Legătura dintre cele două tipuri de coordonate este dată de formulele

$$\begin{cases} x = \rho\cos\theta \\ y = \rho\sin\theta \end{cases} \Leftrightarrow \begin{cases} \rho = \sqrt{x^2 + y^2} \\ \theta = \pm\arccos\frac{x}{\sqrt{x^2+y^2}} \end{cases}$$

unde semnul din faţa lui arccos este dat de semnul lui y.

Este uşor de văzut că argumentul unui număr complex z cu $\operatorname{Re} z \neq 0$ se poate afla şi cu formula

$$\theta = \operatorname{arctg}\frac{y}{x} + k\pi,$$

unde k are una din valorile 0, 1 sau -1, în funcţie de cadranul în care cade z.

Cu aceste notaţii obţinem imediat *forma trigonometrică* a lui z;

$$z = x + iy = \rho\cos\theta + i\rho\sin\theta = \rho(\cos\theta + i\sin\theta).$$

Să observăm că modulul $\rho = |z| = \sqrt{x^2 + y^2}$ şi argumentul $\theta = \arg z$ al unui număr complex $z = x + iy$ au fost implementate în structura `Complex` prin proprietăţile

```
    @property
    def rho(self):
        # furnizeaza modulul rho = |x+yi| pentru
        # z = x+yi = rho(cos(theta) + i sin(theta))
        return math.hypot(self.x, self.y)

    @property
    def theta(self):
        # furnizeaza argumentul theta pentru
        # z = x+yi = rho(cos(theta) + i sin(theta))
        return math.atan2(self.x, self.y)
```

Observaţie. Conform implemetării de mai sus, argumentul $\theta = \arg z$ returnat de proprietatea `theta` este în intervalul $(-\pi, \pi]$, conform cerinţelor. Acest fapt este utilizat, de exemplu, în următorul test care decide dacă punctul z se află sau nu în semiplanul stâng determinat de dreapta ab parcursă de la a la b:

```
def zEsteLaStangaLuiAB(z, a, b):
    return ((z - a) / (b - a)).theta > 0

#Exemplu:

z = Complex(0, 1)
a = Complex(0, 0)
b = Complex(1, 1)

print(zEsteLaStangaLuiAB(z, a, b))
# True
```

4. Fişierul `MyComplex.py`

Prezentăm în continuare modulul în care este definită clasa `Complex`

```
import math
class Complex:

    def __init__(self, x=0, y=0):
        assert isinstance(x, (int, float))
        assert isinstance(y, (int, float))
        self.x = x
        self.y = y

    @property
    def rho(self):
        # furnizeaza modulul rho = |x+yi| pentru
        # z = x+yi = rho(cos(theta) + i sin(theta))
        return math.hypot(self.x, self.y)

    @property
    def theta(self):
        # furnizeaza argumentul theta pentru
        # z = x+yi = rho(cos(theta) + i sin(theta))
        return math.atan2(self.x, self.y)

    @property
    def conj(self):
        # furnizeaza conjugatul
        return Complex(self.x, -self.y)

    def __repr__(self):
        # reprezinta pe z ca string: "Complex(x,y)"
        return f"Complex({self.x}, {self.y})"

    def __str__(self):
        # reprezinta pe z ca string: "x+yi"
        return f"({self.x}{self.y:+}i)"

    def __neg__(self):
        # expresia -z este tradusa z.__neg__()
        return Complex(-self.x, -self.y)

    def __pos__(self):
        # expresia +z este tradusa z.__pos__()
        return Complex(self.x, self.y)
```

```
def __add__(self, other):
    # aduna self + other; z+w se traduce z.__add__(w)
    if isinstance(other, (int, float)):
        return Complex(self.x + other, self.y)
    if isinstance(other, Complex):
        return Complex(self.x + other.x, self.y + other.y)
    raise TypeError(f"Operatie nedefinita: {type(self)} + {type(other)}")

def __radd__(self, other):
    # aduna other + self;  5+z se traduce z.__radd__(5)
    if isinstance(other, (int, float)):
        return Complex(self.x + other, self.y)
    raise TypeError(f"Operatie nedefinita: {type(other)} + {type(self)}")

def __sub__(self, other):
    # scade self - other; z-w se traduce z.__sub__(w)
    if isinstance(other, (int, float)):
        return Complex(self.x - other, self.y)
    if isinstance(other, Complex):
        return Complex(self.x - other.x, self.y - other.y)
    raise TypeError(f"Operatie nedefinita: {type(self)} - {type(other)}")

def __rsub__(self, other):
    # scade other - self; 5-z se traduce z.__rsub__(5)
    if isinstance(other, (int, float)):
        return Complex(other - self.x, -self.y)
    raise TypeError(f"Operatie nedefinita: {type(other)} - {type(self)}")

def __mul__(self, other):
    # returneaza self*other; z*w se traduce z.__mul__(w)
    if isinstance(other, (int, float)):
        return Complex(self.x * other, self.y * other)
    if isinstance(other, Complex):
        return Complex(self.x * other.x - self.y * other.y,
                       self.x * other.y + self.y * other.x)
    raise TypeError(f"Operatie nedefinita: {type(self)} * {type(other)}")

def __rmul__(self, other):
    # returneaza other*self;  5*z se traduce z.__rmul__(5)
    if isinstance(other, (int, float)):
        return Complex(self.x * other, self.y * other)
    raise TypeError(f"Operatie nedefinita: {type(other)} * {type(self)}")

def __truediv__(self, other):
    # returneaza self/other; z/w se traduce z.__truediv__(w)
```

```
        if isinstance(other, (int, float)):
            if other != 0:
                return Complex(self.x / other, self.y / other)
            else:
                raise ZeroDivisionError(f"Impartire la zero: {type(self)} / 0")
        if isinstance(other, Complex):
            ro2 = other.rho ** 2
            if ro2 != 0.0:
                return (self * other.conj) / ro2
            else:
                raise ZeroDivisionError(f"Impartire la zero: {type(self)} / 0")
        raise TypeError(f"Operatie nedefinita: {type(self)} / {type(other)}")

    def __rtruediv__(self, other):
        # returneaza other/self;  5/z devine z.__rtruediv__(5)
        if isinstance(other, (int, float)):
            if self != 0:
                return (self.conj * other) / (self.rho ** 2)
            else:
                raise ZeroDivisionError(f"Impartire la zero: {type(other)} / 0")

    def __eq__(self, other):
        # testeaza daca self == other
        try:
            delta = self - other
        except TypeError:
            raise TypeError(f"Operatie nedefinita: {type(self)} == {type(other)}")
        else:
            return delta.rho < 1.0e-20

    def __ne__(self, other):
        # testeaza daca self != other
        try:
            delta = self - other
        except TypeError:
            raise TypeError(f"Operatie nedefinita: {type(self)} != {type(other)}")
        else:
            return delta.rho >= 1.0e-20

    def __pow__(self, n):
        # calculeaza z**n cu formula lui Moivre
        if isinstance(n, (int, float)):
            rlan = self.rho ** n
            ntheta = self.theta * n
            xx = round(rlan * math.cos(ntheta), 10)
```

```
            yy = round(rlan * math.sin(ntheta), 10)
            return Complex(xx, yy)
        raise TypeError(f"Operatie nedefinita: {type(self)}**{type(n)}")

    @classmethod
    def fromRhoTheta(cls, rho, theta):
        return Complex(rho * math.cos(theta), rho * math.sin(theta))
```