Cursul 13

Mulţimi Julia

Fie $f : \mathbb{C} \to \mathbb{C}$ o funcţie complexă şi fie $f^n = f \circ f \circ \cdots \circ f$ iterata de ordin n a lui f. Peste tot în continuare vom presupune că f este dezvoltabilă în serie de puteri în tot planul (cum sunt funcţiile polinomiale, funcţiile $\exp(z)$, $\sin(z)$, $\cos(z)$, ş.a.) sau că este câtul a două astfel de funcţii (în particular f poate fi o funcţie raţională).

Prin *orbita* (sau *traiectoria*) unui punct $\zeta \in \mathbb{C}$, înţelegem şirul

$$z_0 = \zeta, z_1 = f(\zeta), z_2 = f^2(\zeta), \ldots, z_n = f^n(\zeta), \ldots$$

sau, altfel spus, şirul recurent

$$\begin{cases} z_{n+1} = f(z_n), \ n = 0, 1, 2, \ldots \\ z_0 = \zeta. \end{cases}$$

Orbita unui punct fix este formată numai din acel punct. In general, dacă ζ este un *punct periodic de ordin p*, adică un punct fix al iteratei f^p, p fiind cel mai mic număr natural cu această proprietate, atunci orbita sa este formată numai din p puncte distincte, deoarece $z_p = f^p(\zeta) = \zeta = z_0$. Mai mult, toate cele p puncte ale orbitei, $z_0 = \zeta$, $\ldots$, $z_{p-1} = f^{p-1}(\zeta)$ sunt şi ele periodice de ordin p.

Punctul periodic ζ de ordin p este *atractiv* dacă $|(f^p)'(\zeta)| < 1$, *repulsiv* dacă $|(f^p)'(\zeta)| > 1$ şi *neutru* în rest. Este uşor de văzut că în acest caz avem

$$(f^p)'(z_0) = (f^p)'(z_1) = \cdots = (f^p)'(z_{p-1}) = f'(z_0)f'(z_1)\cdots f'(z_{p-1}),$$

unde $z_0 = \zeta$, $\ldots$, $z_{p-1} = f^{p-1}(\zeta)$ sunt punctele orbitei lui ζ, şi prin urmare punctele unei orbite periodice sunt în acelaşi timp toate atractive sau toate repulsive sau toate de tip neutru.

Dacă ζ este un punct periodic de ordin p atractiv atunci există o vecinătate a sa cu proprietatea că pentru orice $\widetilde{\zeta}$ din acea vecinătate subşirul $z_{pk} = f^{pk}(\widetilde{\zeta})$ este convergent la ζ pentru $k \to \infty$. Mai mult, în acest caz toate punctele orbitei lui ζ sunt atractive (formează o *orbită periodică atractivă*) şi pentru orice astfel de orbită există o mulţime deschisă $B \subset \mathbb{C}$, *bazinul ei de atracţie*, astfel încât pentru orice $\widetilde{\zeta} \in B$ şirul $(f^n(\widetilde{\zeta}))_n$ este format din subşirurile $(f^{pk}(\widetilde{\zeta}))_k$, $(f^{pk+1}(\widetilde{\zeta}))_k$, $\ldots$, $(f^{pk+p-1}(\widetilde{\zeta}))_k$ convergente fiecare la câte unul din punctele orbitei periodice.

Punctele $\zeta \in \mathbb{C}$ au fost clasificate în funcţie de *predictibilitatea* orbitelor lor. Punctele cu un comportament previzibil formează *mulţimea Fatou* asociată lui f, notată F_f, iar celelalte formează *mulţimea Julia*, J_f. Definiţiile exacte presupun

cunoştinţe avansate de teoria funcţiilor complexe care depăşesc interesul nostru, aşa că vom preciza doar semnificaţia lor. Punctul ζ este un punct *Fatou* dacă toate traiectoriile care pornesc dintr-o vecinătate a sa au acelaşi comportament: de exemplu toate tind la acelaşi punct fix atractiv (finit sau nu), sau toate tind la aceeaşi orbită periodică atractivă.

Printr-o teoremă clasică dată Paul Montel în 1927, dacă ζ este un punct Julia, atunci pentru orice vecinătate U a sa, oricât de mică, reuniunea traiectoriilor care pleacă din punctele lui U umple tot planul, cu excepţia eventuală a cel mult două puncte. Impredictibilitatea orbitei lui $\zeta \in J_f$ este evidentă: o eroare oricât de mică în alegerea datei iniţiale $\widetilde{\zeta}$ în vecinătatea lui ζ poate conduce la o orbită $(f^n(\widetilde{\zeta}))$ care se poate îndepărta oricât de mult de cea a lui ζ.

Este clar că orice punct fix repulsiv este punct Julia. Mai mult, în cazul funcţiilor considerate de noi, are loc următoarea caracterizare: mulţimea Julia J_f este închiderea mulţimii punctelor periodice repulsive ale lui f.

Să exemplificăm pentru funcţia $f(z) = z^2$. Ecuaţia $f(z) = z$ are numai două soluţii $z_0 = 0$ şi $z_1 = 1$. Deoarece $f'(z) = 2z$, obţinem imediat că $z_0 = 0$ este un atractor (adică un punct fix atractiv) iar $z_1 = 1$ este un repulsor (punct fix repulsiv). Iteratele de ordin n sunt de forma

$$f^n(z) = z^{2^n},$$

prin urmare orbita oricărui $\zeta \in \mathbb{C}$ este de forma (ζ^{2^n}). Dacă $|\zeta| < 1$ atunci $\zeta^{2^n} \to 0$ pentru $n \to \infty$ iar dacă $|\zeta| > 1$ avem $\zeta^{2^n} \to \infty$. Să observăm că şi punctul de la infinit este punct fix, $f(\infty) = \infty$. In final obţinem că mulţimea Fatou a lui f este formată din două componente conexe

$$F_f = B_0 \cup B_\infty,$$

mulţimea $B_0 = \{z \in \mathbb{C} \text{ cu } |z| < 1\}$ fiind bazinul de atracţie al lui $z_0 = 0$ iar $B_\infty = \{z \in \mathbb{C} \text{ cu } |z| > 1\}$ bazinul de atracţie al punctului de la infinit. Punctele care au rămas, cele de pe cercul unitate, formează mulţimea Julia

$$J_f = \{z \in \mathbb{C} \text{ cu } |z| = 1\}.$$

Să observăm că avem şi egalitatea

$$J_f = \partial B_\infty,$$

care este adevărată în general pentru orice funcţie f polinomială.

Punctele periodice de ordin $p \geq 2$ sunt soluţiile nenule ale ecuaţiei $f^p(z) = z$, adică $z^{2^p} = z$, care sunt de forma

$$z_{k,p} = \cos\frac{2k\pi}{2^p - 1} + i \sin\frac{2k\pi}{2^p - 1}, \quad k = 0, 1, \ldots, 2^p - 2.$$

Toate acestea se află pe cercul unitate şi sunt puncte periodice repulsive, deoarece $|f'(z_{k,p})| = 2|z_{k,p}| = 2 > 1$. Este uşor de văzut că se verifică egalitatea

$$J_f = \overline{\bigcup_{p=2}^{\infty} \{z_{k,p} \,|\, k = 0, 1, \ldots, 2^p - 2\}}.$$

1. Colorarea mulţimii Julia pe baza definiţiei. Următorul program Python colorează cu alb mulţimea Julia şi cu roşu mulţimea Fatou asociată unei funcţii complexe f, analizând comportarea şirului $(f^n(z_0))$ pentru două valori aleatoare ale datei initiale ζ în regiunea din $\mathbb{C}$ corespunzătoare pixelului colorat.

```
import ComplexPygame as C
import Color
import random

def JuliaRandom():
    c = 0.45 + 0.2j
    rhoMax = 1.0e20

    def f(z):
        # f(z)=(z*z*z+c)/(z*z*z-c)
        u = z * z * z
        if u == c:
            return rhoMax
        else:
            return (u + c) / (u - c)

    def zar():
        return random.uniform(-0.5, 0.5)

    C.setXminXmaxYminYmax(-2.7, 3.5, -3.1, 3.1)
    C.fillScreen()
    prec = 0.01
    nrIter = 100
    for coloana in C.screenColumns():
        for zeta in coloana:
            z1 = zeta + prec * complex(zar(), zar())
            z2 = zeta + prec * complex(zar(), zar())
            for k in range(nrIter):
                z1 = f(z1)
                z2 = f(z2)
                if C.rho(z1) + C.rho(z2) >= rhoMax:
                    break
```

```
            col = Color.White
            if C.rho(z1) >= rhoMax and C.rho(z2) >= rhoMax                    or C.rho
                col = Color.Red
            C.setPixel(zeta, col)
        if C.mustClose():
            return

if __name__ == '__main__':
    C.initPygame()
    C.run(JuliaRandom)
```

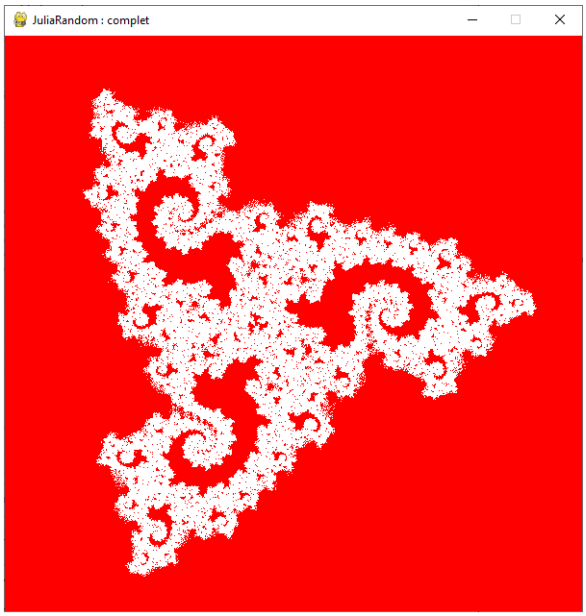

FIGURA 1. `JuliaRandom()`

2. Metoda locului final. Următoarea funcţie re-colorează mulţimea Julia din exemplul precedent prin *metoda locului final*: au aceeaşi culoare pixelii ζ care după un număr prestabilit de iteraţii (suficient de mare), ajung în acelaşi loc.]

```
def JuliaGreen():
    c = complex(0.45, 0.2)
    rhoMax = 1.0e20

    def f(z):
        # f(z)=(z*z*z+c)/(z*z*z-c)
        u = z * z * z
```

```
        if u == c:
            return rhoMax
        else:
            return (u + c) / (u - c)

C.setXminXmaxYminYmax(-2.7, 3.5, -3.1, 3.1)
nrIter = 100
for coloana in C.screenColumns():
    for zeta in coloana:
        z = zeta
        for k in range(nrIter):
            z = f(z)
            if C.rho(z) >= rhoMax: break
        C.setPixel(zeta, Color.Index(sum(C.getHK(z))))
    if C.mustClose():
        return
```

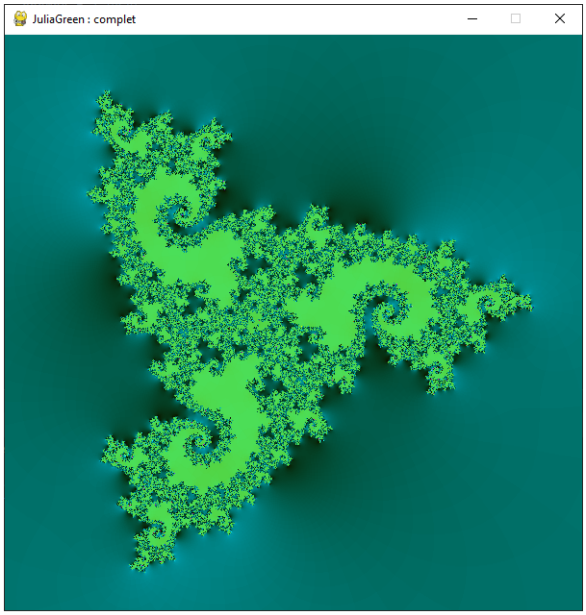

FIGURA 2. `JuliaGreen()`

Imaginea din Figura 3 a fost obţinută tot prin metoda locului final, dar pentru funcţia $f_c(z) = z^2 + c$, cu $c = -0.21 - 0.70i$. Sunt puse în evidenţă componentele conexe ale mulţimii Fatou, componenta albă corespunde punctului de la infinit, care în acest caz este un punct fix atractiv, iar componentele colorate corespund unei orbite periodice atractive.

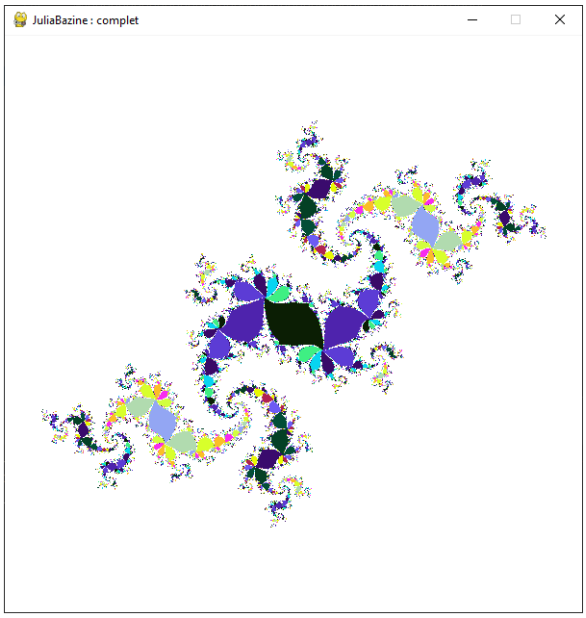

FIGURA 3. Componente Fatou

3. Metoda iteraţiilor retrograde. Fie $f : \mathbb{C} \to \mathbb{C}$ o funcţie raţională şi J_f mulţimea Julia asociată ei. Dacă z_0 este un punct oarecare din J_f (de exemplu un punct fix repulsiv) atunci J_f coincide cu închiderea mulţimii predecesorilor lui z_0, adică

$$J_f = \overline{\bigcup_n f^{-n}(z_0)},$$

unde $f^{-n}(z_0) = \{z \in \mathbb{C} \,|\, f^n(z) = z_0\}$. Această proprietate stă la baza *metodei iteraţiilor retrograde*, metodă folosită de următorul program pentru trasarea mulţimii Julia asociate funcţiei polinomiale de gradul doi $f(z) = z^2 + c$, cu $c = -0.4538 + 0.55946\,i$.

```
def JuliaRetro():  # f(z) = z * z + c
    c = -0.4538 + 0.55946j

    def Sqrt(z):
        a = z.real
        b = z.imag
        if b == 0.0:
            if a >= 0.0:
                return complex(math.sqrt(a), 0.0)
            else:
                return complex(0.0, math.sqrt(-a))
        else:
```

```
        w = math.sqrt(a * a + b * b)
        if b >= 0.0:
            return complex(math.sqrt((w + a) / 2.0),
                           math.sqrt((w - a) / 2.0))
        else:
            return complex(-math.sqrt((w + a) / 2.0),
                           math.sqrt((w - a) / 2.0))

# def Sqrt(z):
#    return C.fromRhoTheta(math.sqrt(C.rho(z), C.theta(z)/2)
#
# def Sqrt(z):
#     return pow(z, 0.5)

def trans_total(li):
    rez = []
    for z in li:
        rez.append(Sqrt(z - c))
        rez.append(-Sqrt(z - c))
    return rez

def trans_random(li):
    rez = []
    for z in li:
        C.setPixel(z, Color.Black)
        if random.random() < 0.5:
            rez.append(Sqrt(z - c))
        else:
            rez.append(-Sqrt(z - c))
    return rez

C.setXminXmaxYminYmax(-1.5, 1.5, -1.5, 1.5)
delta = Sqrt(1 - 4 * c)
fig = [(1 + delta) / 2, (1 - delta) / 2]
for k in range(15):
    fig = trans_total(fig)
for k in range(1000):
    fig = trans_random(fig)
    if k % 10 == 0 and C.mustClose():
        return
```

4. Mulţimi Julia pentru polinoame. În cazul polinoamelor, punctul de la infinit este un punct fix atractor, iar frontiera bazinului său de atracţie coincide cu mulţimea Julia asociată polinomului. Complementara bazinului de atracţie al punctului de la infinit este numită *mulţimea Julia plină* asociată polinomului

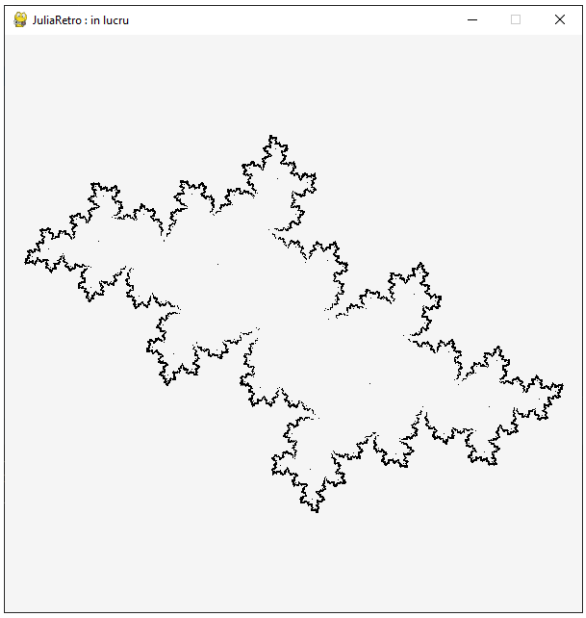

FIGURA 4. `JuliaRetro()`

f, şi este formată din acele puncte z_0 pentru care şirul $(f^{\circ n}(z_0))$ este mărginit. Frontiera acesteia coincide şi ea, evident, cu mulţimea Julia propriu-zisă a lui f.

Următoarea funcţie utilizează modul de colorare ETA (*escape time algorithm*) pentru a pune în evidenţă mulţimea Julia plină ataşată funcţiei f din exemplul precedent.

```
def JuliaPlina():
    c = complex(-0.4538, 0.55946)

    def f(z):
        return z * z + c

    C.setXminXmaxYminYmax(-1.5, 1.5, -1.5, 1.5)
    nrIter = 1001
    for coloana in C.screenColumns():
        for zeta in coloana:
            z = zeta
            for k in range(nrIter):
                if C.rho(z) > 4: break
                z = f(z)
            C.setPixel(zeta, Color.Index(775 + k))
        if C.mustClose():
            return
```

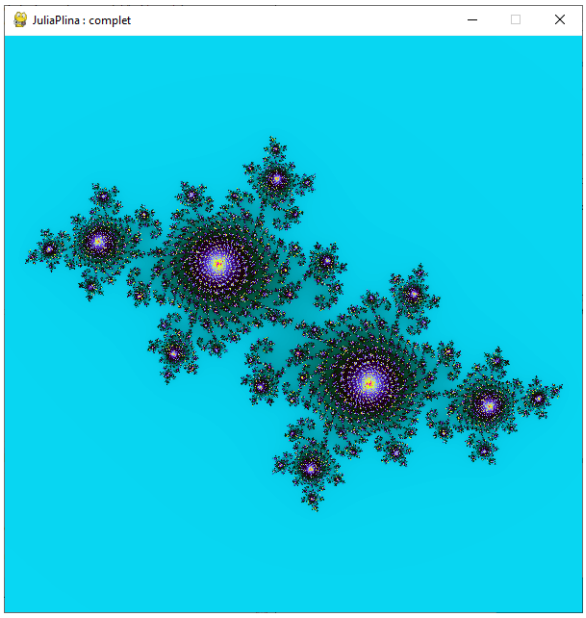

FIGURA 5. `JuliaPlina()`

5. Exemplu final. Următorul program Python ilustrează mulţimea Julia a funcţiei f asociate prin metoda lui Newton polinomului generalizat

$$p(z) = (z^3 - 1)^{\omega},$$

pentru $\omega = 0.54 + 0.5i$.

```
import ComplexPygame as C
import Color
import math

def JuliaNewton():
    eps0 = C.fromRhoTheta(1.0, 0.0 * math.pi / 3.0)
    eps1 = C.fromRhoTheta(1.0, 2.0 * math.pi / 3.0)
    eps2 = C.fromRhoTheta(1.0, 4.0 * math.pi / 3.0)
    omega = 0.54 + 0.5j

    def f(z):
        if z in [eps0, eps1, eps2]:
            return z
        w0 = omega / (z - eps0)
        w1 = omega / (z - eps1)
        w2 = omega / (z - eps2)
        return z - 1 / (w0 + w1 + w2)
```

```
    C.setXminXmaxYminYmax(-1000.5, 1000.5, -1000.5, 1000.5)
    prec = 0.001
    nrIter = 10000
    for coloana in C.screenColumns():
        for zeta in coloana:
            z = zeta
            for k in range(nrIter):
                z = f(z)
                if C.rho(z - eps0) < prec \
                        or C.rho(z - eps1) < prec \
                        or C.rho(z - eps2) < prec:
                    break
            C.setPixel(zeta, Color.Index(635 + 5 * k // 2))
        if C.mustClose():
            return

if __name__ == '__main__':
    C.initPygame()
    C.run(JuliaNewton)
```

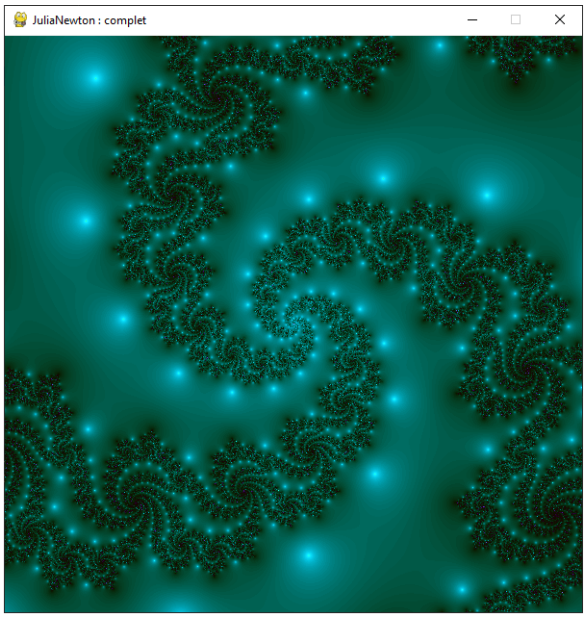

FIGURA 6. JuliaNewton()