# Aula 24: 13/NOV/2018

## Aula passada

**Método científico**. Está presente em tudo que fizemos.

- **observar** algum aspecto da natureza
- **hipotetizar** um modelo consistente com as observações
- **prever** eventos usando as hipóteses
- **verificar** as predições fazendo mais observações
- **validar** repetindo até que hipoteses e obeservações estajam de acordo

Mais modelos computacionais e simulação em pesquisa cientifica: laboratório de ideias com área do círculo

## Hoje

- classe nativa `complex`
- `complex` em sequências iteradas
- `complex` em fractais: conjunto de Mandelbrot

## Fractais

Um **fractal** é um conjunto que exibe um padrão repetido em cada escala.

Enquanto equação, fractais são usualmente não diferenciáveis em todos os pontos.

### Funções iteradas

Seja $\mathtt{f}(\mathtt{z}) = \mathtt{z}^2 - \mathtt{z}_0$ onde $\mathtt{z}_0$ é uma constante complexa.

Iterações: $\mathtt{z}_0$, $\mathtt{z}_1 = \mathtt{f}(\mathtt{z}_0)$, $\mathtt{z}_2 = \mathtt{f}(\mathtt{z}_1)$, etc.

Após várias iterações, o valor de $|\mathtt{z}_\mathtt{i}|$ pode continuar pequeno ou pode explodir, dependendo do valor inicial $\mathtt{z}_0$

Exemplo: $\mathtt{f}(\mathtt{z}) = \mathtt{z}^2 - 0.75$

$\mathtt{z}_0 = 1$

$\mathtt{z}_1 = 0.25$

$\mathtt{z}_2 = -0.6875$

$\mathtt{z}_3 = -0.2773$

$\mathtt{z}_4 = -0.6731$

## Sequência iterada

Considere a sequência $z_0, z_1, z_2, \ldots, z_t, \ldots$ onde $z_{t+1} = z_t^2 + z_0$.

Por exemplo, para $z_0 = 1 + i$ temos que

| $t$ | $z_t$ |
|---|---|
| 0 | $1 + i$ |
| 1 | $(1 + i)^2 + 1 + i = 1 + 3i$ |
| 2 | $(1 + 3i)^2 + 1 + i = -7 + 6i$ |

Usa a classe nativa `complex`.

Exemplo de uso

```
% python seq_iterada.py 3 1 1
0 : (1+1j)
1 : (1+3j)
2 : (-7+7j)
```

```
import sys
def main(argv = None):
    argv = sys.argv
    n  =  int(argv[1])
    re = float(argv[2])
    im = float(argv[3])
    z0 = complex(re, im)
    z = z0
    for t in range(n):
        print(str(t) + " : " + str(z))
        z = z*z + z0
```

## Conjunto de Mandelbrot

Definido algoritmicamente.

Um ponto $z_0$ está no conjunto de se a sequência $|z_0|, |z_1|, |z_2|, \ldots$ é limitada, ou seja, existe uma constante $c$ tal que $|z_t| < c$ para todo $t$.

Exemplos:

| $z_0$ | está no conjunto? |
|---|---|
| $0 + 0i$ | sim |
| $2 + 0i$ | não |
| $1 + i$ | não |
| $-0.5 + 0i$ | sim |
| $.10 - 0.64i$ | sim |

**Fato**. $z_0$ está no conjunto de Mandelbrot se e somente se $|z_t| < 2$ para todo $t$.

Usaremos esse fato no programas mais adiante.

### Problema

Fazer um programa que dado número real `size` e um ponto `(xc,yc)` desenha o conjunto de Mandelplot contido no quadrado de largura `size` e centro `(xc, yc)`.

Ideia: representar o conjunto através de uma imagem onde associamos a cada número complexo $z_0 = x_0 + y_0 i$ ao ponto $(x_0, y_0)$. Cada ponto no conjunto pode ser pintado como uma certa cor `MAND`

Como podemos fazer isso?

Lembrar dos EP02, EP03, EP04.

Podemos utilizar uma matriz `img` de dimensão `n` $\times$ `n` como uma representação discreta do conjunto de Madelbrot. A cada *pixel* `[i][j]` da matriz associamos a um ponto no plano $\texttt{z}_0 = \texttt{x}_0 + \texttt{y}_0\texttt{i}$ do plano complexo da seguinte:

```
x0 = xc - size/2 + i*size/n
y0 = xc - size/2 + i*size/n
```

Agora associamos ao pixel `[i][j]` a *cor* correpondente ao menor inteiro `t` tal que $|\texttt{z}_\texttt{t}| > 2$.

Se a sequência iterada com início em `z_0` não deixar o círculo de raio 2 em `range(MAX)` iterações declaramos que o ponto $\texttt{x}_0 + \texttt{y0i}$ está no conjunto e associamos a *cor`MAX` ao pixel correspondente

### Função básica

A função básica é a seguinte:

```
#-----------------------------------------------------
def mand(z0, maxi):
    '''(complex, int) -> int

    Retorna o menor inteiro t < maxi tal que a sequência iterada
    começando em z0 deixa o interior do círculo de raio 2 ao chegar em z_t.
    Se a sequência não deixar o interior do círculo de
    raio 2 em maxi iterações a função retorna maxi.
    '''
    z = z0
    for t in range(maxi):
        if abs(z) > 2: return t  # norma de complexos
        z = z*z + z0       # multiplicação de complexos
    return maxi
```

### Cliente

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
from matplotlib import pyplot as plt
from config import *
def main(argv=None):
```

```
    xc   = float(argv[1])
    yc   = float(argv[2])
    size = float(argv[3])

    color_map = COLOR_MAP
    if argc == 5: color_map = argv[4]

    n  = N    # create n-by-n image
    maxi = MAX  # maximum number of iterations
    img = [[0 for col in range(n)] for lin in range(n)]
    for lin in range(n):
        for col in range(n):
            x0 = xc - size/2 + size*col/n
            y0 = yc + size/2 - size*lin/n
            z0 = complex(x0, y0)
            cor = maxi-mand(z0, maxi)
            img[lin][col] = cor

    # crie e mostre a imagem
    fig = plt.figure(figsize=(10,10))
    pcolor = plt.pcolor(img, cmap = color_map)
    fig.canvas.draw()
    plt.show()
```

## Animação

Arquivo `Mandelbrot_sequence_new.gif` foi copiado da Wikipedia:

Zooming into the Mandelbrot set

Simpsons contributor at English Wikipedia - Transferred from en.wikipedia to Commons by Franklin.vp using CommonsHelper.

```
Used Zom-B's library with my own code and a golden gradient (similar to the
default gradient used in Ultra Fractal). Each scene is 6x supersampled to
remove sharp edges. Took... a while to render Links to Java source code: Zom-B
version project directory containing DoubleDouble class, adjustments made by
Simpsons Contributor to keep max iteration and anti-aliasing factor at more
conservative values for faster rendering. New golden gradient added. Includes
animated gif encoder. Contents 1 Zom-B version 2 Licensing 3 Src code 4
Original upload log Zom-B version Mandelbrot zoom with center at
(-0.743643887037158704752191506114774, 0.131825904205311970493132056385139)
and magnification 1 .. 3.18 × 1031 created using my own Java program, using:
Double-double precision (self-written library), Adaptive maxiter depending on
the inverse square root of the magnification Adaptive per-pixel antialiasing
strength depending on the maximum iteration of nearby pixels (15x AA max),
(during antialiasing phase, maxiter is quadrupled), Iteration smoothing, New
warm gradient which also gives clearer details, applied to the base-2 log of
the smoothed iteration number, Modified periodicity checking algorithm from
Fractint, for significant speedup, Main cardioid and period-2 bulb checking
```

```
for another speedup, Multi-threaded calculation 136 hours calculation time on
two PC's (6 cores combined)

    Public Domain
    File:Mandelbrot sequence new.gif
    Created: 27 January 2010
```

```
from pilutil import mostre_animacao
from pilutil import salve_animacao
def main(argv=None):
    xc    = float(argv[1]);
    yc    = float(argv[2]);
    size = float(argv[3]);
    color_map = COLOR_MAP
    n     = N     # create N-by-N image
    maxi = MAX  # maximum number of iterations
    video = [None]*FRAMES
    for k in range(int(FRAMES)):
        img = [[0 for j in range(n)] for i in range(n)]
        for lin in range(n):
            for col in range(n):
                x0 = xc - size/2 + size*col/n
                y0 = yc + size/2 - size*lin/n
                z0 = complex(x0, y0)
                cor = maxi-mand(z0, maxi)
                img[lin][col] = cor
        video[k] = img
        size /= ZOOM

    # crie e mostre animação
    mostre_animacao(video, color_map)
    salve_animacao(video, color_map)
```

## Complex

Uma implementação de números complexos em Python

```
class Complex(object):
    def __init__(self, real, imag=0.0):
        self.real = real
        self.imag = imag

    def __str__(self):
        if self.imag == 0: return "(" + str(self.real) + "+ 0j)"
        if self.real == 0: return str(self.imag) + "j"
        if self.imag <  0: return "(" + str(self.real) + "+" + str(-self.imag) + "j)"
        return "(" + str(self.real) + "+" + str(self.imag) + "j)"

    def __add__(self, other):
        return Complex(self.real + other.real,
                       self.imag + other.imag)

    def __sub__(self, other):
        return Complex(self.real - other.real,
                       self.imag - other.imag)

    def __mul__(self, other):
        return Complex(self.real*other.real - self.imag*other.imag,
                       self.imag*other.real + self.real*other.imag)

    def __div__(self, other):
        sr, si, or, oi = self.real, self.imag, \
                         other.real, other.imag # short forms
        r = float(or**2 + oi**2)
        return Complex((sr*or+si*oi)/r, (si*or-sr*oi)/r)

    def __abs__(self):
        return sqrt(self.real**2 + self.imag**2)

    def __neg__(self):   # defines -c (c is Complex)
        return Complex(-self.real, -self.imag)

    def __eq__(self, other):
        return self.real == other.real and self.imag == other.imag

    def __ne__(self, other):
        return not self.__eq__(other)

    def __str__(self):
        return '(%g, %g)' % (self.real, self.imag)

    def __repr__(self):
        return 'Complex' + str(self)
```

```
    def __pow__(self, power):
        raise NotImplementedError\
              ('self**power is not yet impl. for Complex')
```

# Apêndice

```
The sequence generated by f(z) = z^2 + c beginning with 0 begins
     0,c,c^2 + c, ...
Call these terms z0, z1, z2, ...

In fact this sequence will be unbounded if any of its terms
has magnitude larger than 2.

The main idea is that if the magnitude |z| is bigger than both
2 and the magnitude |c|, then |f(z)|/|z| > |z| - 1 > 1, so that
by induction the sequence of magnitudes will grow geometrically.

To establish the inequality, note that
  |f(z)|/|z| = |z^2 + c|/|z|
             >= (|z|^2-|c|)/|z|          [ triangle inequality ]
              = |z| - (|c|/|z|)
              > |z| - 1                  [ |z| > |c| ]
              > 1                        [ |z| > 2 ]

If  |c| <= 2 and  |zn| > 2 then certainly
z = zn satisfies these hypotheses, so the sequence is
unbounded.

If |c| > 2 then |c^2+c| >= |c|^2 - |c| = |c|(|c|-1) > |c| > 2
so that z=z2 satisfies the hypotheses of the argument above.

mike hurley   mgh3@po.cwru.edu
```