28 Aula 28: 18/NOV/2019

28.1 Aulas passadas

- Aula 25: método científico
- Aula 26: exercícios: análise de algoritmos, 2SUM
- Aula 27: exercícios: monte carlo, exercício da prova de MAE0212

28.2 Hoje

- números complexos: conjunto de Mandelbrot
- computabilidade: problema da parada (*halting problem*)
- adaptative plot: deixa para lá

28.3 Arquivos

- `00-exemplos`: exemplos de execuções de `mandelbrot.py` e `mandelbrot-anim.py` com imagens legais;
- `seq_iterada.py`: dado número complexo `c` gera $\texttt{z}_0, \texttt{z}_1, \texttt{z}_2, \ldots$;
- `config.py`: contatates para os programa `mandelbrot.py` e `mandelbrot-anim.py`
- `mandelbrot.py`: gera imagens do conjunto de Mandelbrot (ver `testes.txt` para exemplos);
- `mandelbrot-anim.py`: gera video com imagens do conjunto de Mandelbrot (ver `testes.txt` para exemplos);
- `numpymagem.py`: arquivo do EP04, usado por `mandelbrot-anim.py`;
- `numpymutil.py`: arquivo do EP04 e EP05, usado por `mandelbrot-anim.py`;
- `videos`: diretório com alguns vídeos.
- `wikipedia_imgs`: diretorio com videos e gifs da wikipedia

28.4 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

Considere a função $\mathtt{f}(\mathtt{z}) = \mathtt{z}^2 - \mathtt{c}$ onde $\mathtt{c}$ é uma constante complexa.

Considere agora a seguinte sequência de iterações de $\mathtt{f}(\mathtt{z})$:

$\mathtt{z}_0 = 0$, $\mathtt{z}_1 = \mathtt{f}(\mathtt{z}_0)$, $\mathtt{z}_2 = \mathtt{f}(\mathtt{z}_1)$, $\mathtt{z}_3 = \mathtt{f}(\mathtt{z}_2)$, etc.

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

Exemplos

Para $\mathtt{c} = 1$ temos que $\mathtt{f}(\mathtt{z}) = \mathtt{z}^2 + 1$ e

t	z_t
0	0
1	1
2	2
3	5
4	26
5	677

Para $\mathtt{c} = -1$ temos que $\mathtt{f}(\mathtt{z}) = \mathtt{z}^2 - 1$ e

t	z_t
0	0
1	-1
2	0
3	-1
4	0
5	-1

Para $\mathtt{c} = -0.75$ temos que $\mathtt{f}(\mathtt{z}) = \mathtt{z}^2 + -0.75$ e

t	z_t
0	0
1	-0.75
2	-0.1875
3	$-0.7148\ldots$
4	$-0.2389\ldots$

t	z_t
5	$-0.6928\ldots$

28.5 Sequências iteradas

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

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

t	z_t
0	0
1	$1 + i$
2	$(1 + i)^2 + 1 + i = 1 + 3i$
3	$(1 + 3i)^2 + 1 + i = -7 + 7i$
4	$(-7 + 7i)^2 + 1 + i = 1 - 97i$
5	$(1 - 97i)^2 + 1 + i = -9407 - 193i$

Complex

Uso da classe nativa `complex`.

Exemplo de uso

```
    % python seq_iterada.py 3 1 1
    z0 : 0, abs(z0)= 0
    z1 : (1+1j), abs(z1)=1.41421
    z2 : (1+3j), abs(z2)=3.16228
    z3 : (-7+7j), abs(z3)=9.89949
```

```
def main():
    n  = int(...)
    re = float(...)
    im = float(...)
    c = complex(re, im)
    z = 0
    for t in range(n):
        print("z%d : %s, abs(z%d)=%2g"%(t, z, t, abs(z)))
        z = z*z + c
```

28.6 Conjunto de Mandelbrot

Definido algoritmicamente.

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

Exemplos:

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

Fato. Um número complexo c está no conjunto de Mandelbrot se e somente se $|z_t| < 2$ para todo t.

Demonstração (por mike hurley `mgh3@po.cwru.edu`): Temos que a sequência definida por $\mathtt{f(z)} = \mathtt{z}^2 + \mathtt{c}$ e começando por 0 é

$$0, \mathtt{c}, \mathtt{c}^2+\mathtt{c}, (\mathtt{c}^2+\mathtt{c})^2+\mathtt{c} = \mathtt{c}^4+2\mathtt{c}^3+\mathtt{c}^2+\mathtt{c}, \ldots$$

Como antes, chamaremos os termos da sequência de $\mathtt{z}_0, \mathtt{z}_1, \mathtt{z}_2, \ldots$

É suficiente mostrar que essa sequência será ilimitada se a norma de qualquer dos seus termos for maior que 2.

A ideia principal da desmonstração é que se $|\mathtt{z}|$ é maior 2 e $|\mathtt{c}|$, então

$$\frac{|\mathtt{f(z)}|}{|\mathtt{z}|} > |\mathtt{z}| - 1 > 1.$$

Com isso, por indução, a sequência das normas dos elementos da sequência crescerão em uma progressão geométrica maior que 1.

Para verifica a desigualdade, no que

$$\begin{aligned} |\mathtt{f(z)}|/|\mathtt{z}| &= |\mathtt{z}^2+\mathtt{c}|/|\mathtt{z}| \\ &\geq (|\mathtt{z}|^2 - |\mathtt{c}|)/|\mathtt{z}| && (1) \\ &= |\mathtt{z}| - (|\mathtt{c}|/|\mathtt{z}|) \\ &> |\mathtt{z}| - 1 && (2) \\ &> 1, && (3) \end{aligned}$$

onde (1) é devida a desigualdade triangular e (2) e (3) da hipótese de que $|\mathtt{z}| > \max 2, |\mathtt{c}|$.

Se $|\mathtt{c}| \leq 2$ e $|\mathtt{z_t}| > 2$ então certamente $\mathtt{z} = \mathtt{z_t}$ satisfaz essas hipóteses e portanto a sequência é ilimitada.

Se $|\mathtt{c}| > 2$, então $|\mathtt{c}^2+\mathtt{c}| \geq |\mathtt{c}|^2 - |\mathtt{c}| = |\mathtt{c}|(|\mathtt{c}|-1) > |\mathtt{c}| > 2$ e portanto $\mathtt{z} = \mathtt{z}_2$ satisfaz as hipótes do argumento acima.

Usaremos esse fato na função a seguir.

28.7 Função básica

A função básica é a seguinte:

```
#-----------------------------------------------------
def mandel(c, maxi):
    '''(complex, int) -> int
```

```
    Retorna o menor inteiro t < maxi tal que a sequência iterada
    começando em 0 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 = 0
    for t in range(maxi):
        if abs(z) > 2: return t  # norma de complexos
        z = z*z + c              # multiplicação de complexos
    return maxi
```

28.8 Problema

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

Ideia: representar o conjunto através de uma imagem onde associamos a cada número complexo $z = x + yi$ ao ponto (x, y). Por sua vez cada ponto no conjunto pode ser pintado como uma certa cor retornada por uma função `mandel(c,z)`.

Como podemos fazer isso?

Lembrar dos EP02, EP03, EP04 e EP05.

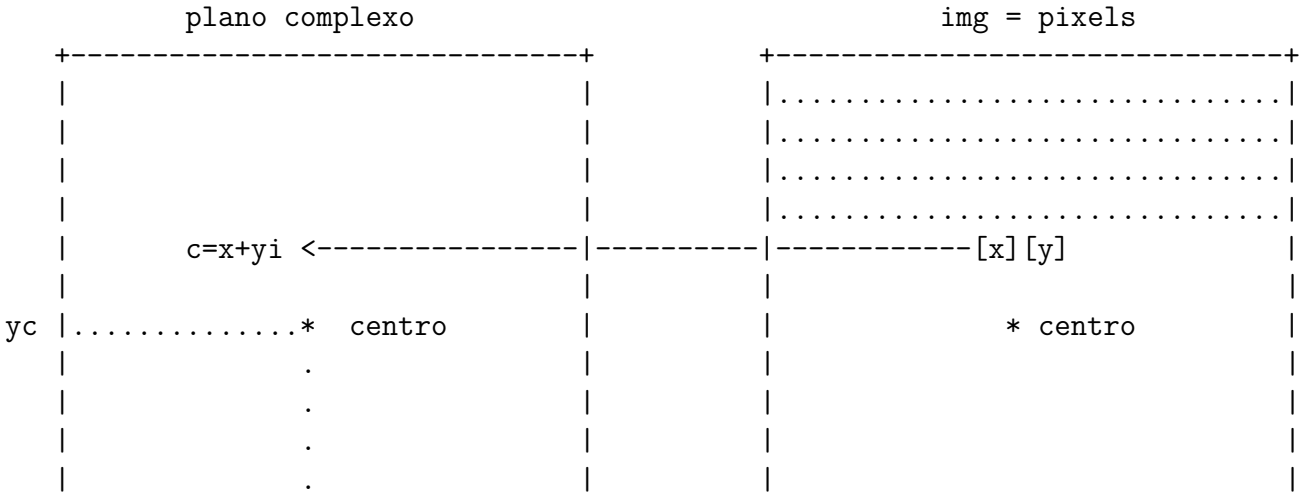
Podemos utilizar uma matriz `img` de dimensão `n` × `n` como uma representação discreta do conjunto de Madelbrot. A cada *pixel* `[i][j]` da matriz será associado a um número complexo `z` = `x` + `yi` no plano complexo da seguinte maneira:

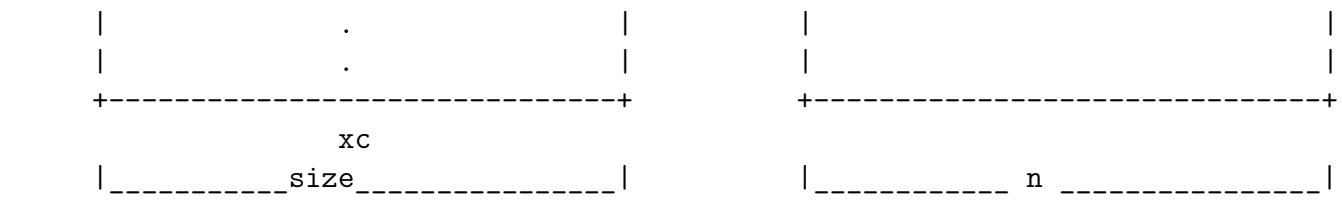
```
x = xc - size/2 + i*size/n
y = yc + size/2 - j*size/n
```

Os sinais na expressão acima dependem de onde está a origem da biblioteca que exibirá a imagem. Parece que `MatPlotLib` e `PyGame` tem origem pontos diferentes da tela.

Agora associamos ao pixel `[i][j]` a *cor* correspondente ao menor inteiro `t` tal que $|z_t| > 2$.

Se a sequência iterada com início em `z_0` não deixar o círculo de raio 2 em `range(MAX)` iterações declaramos que o ponto $x_0 + y0i$ está no conjunto e associamos a *cor* `MAX` ao pixel correspondente





28.9 Cliente

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
from matplotlib import pyplot as plt

# N, COLOR_MAP, max
from config import *

def main():
    # centro da imagem
    xc   = float(...)
    yc   = float(...)
    # largura da região do plano complexo
    size = float(...)

    # palete para cores
    color_map = COLOR_MAP

    n    = N    # cria uma imagem n-por-n
    maxi = MAX  # número máximo de iterações
    img = [[0 for col in range(n)] for lin in range(n)]
    for lin in range(n):
        for col in range(n):
            x = xc - size/2 + col*size/n
            y = yc + size/2 - lin*size/n
            c = complex(x0, y0)
            cor = maxi - mandel(c, 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()
```

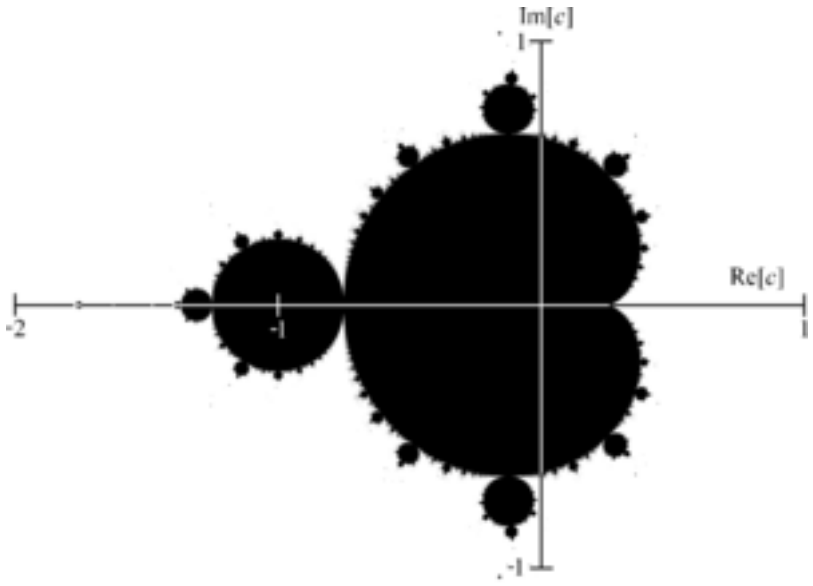


Figure 1: Conjunto de Mandelbrot

28.10 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
(-0.74364388703715870475219150611477 4, 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
```

Mandelbrot anim

```
# EP04
from numpymagem import NumPymagem

# EP05
from numpymutil import mostre_video
from numpymutil import salve_video

# N, COLOR_MAP, FRAMES, WHITE, ZOOM
from config import *

def main():
    # centro da imagem
    xc   = float(...);
    yc   = float(...);
    # largura da região do plano complexo
    size = float(...);

    # palete para cores
    color_map = COLOR_MAP

    n    = N    # create N-by-N image
    maxi = MAX  # maximum number of iterations
    video = [None]*FRAMES
    for k in range(FRAMES):
        print("gerando imagem:", k, "de", FRAMES)
        img = NumPymagem(N, N, WHITE)
        for lin in range(n):
            for col in range(n):
                x = xc - size/2 + col*size/n
                y = yc + size/2 - lin*size/n
                c = complex(x, y)
                cor = maxi - mandel(c, maxi)
                img.put(lin, col, cor)
        video[k] = img
        size /= ZOOM # ZOOM e levemente maior que 1

    # crie e mostre animação
    salve_video(video)
    mostre_video(video)
```

28.11 Complex

Uma implementação de números complexos em Python. Python tem a classe `complex` nativa.

```
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')
```

28.12 Computabilidade

Este texto sobre computabilidade é um resumo da página 5.4 Computability.

Conjectura de Collatz

Ver *Collatz conjecture* na Wikipedia.

Conjectura: A função `collatz()` alcançará o número 1, independentemente de qual número inteiro positivo for escolhido inicialmente.

```
def collatz(n):
   if n == 1: return
   if n % 2 == 1: return collazt(3*n+1)
   return collatz(n//2)
```

Conjectura verificada por computadores até 87×2^{60}.

Conjectura de Goldbach

A conjectura de Goldbach é um dos problemas em aberto mais antigos e mais conhecidos na teoria dos números e em toda a matemática. Afirma:

> Todo inteiro par maior que 2 pode ser expresso como a soma de dois números primos.

```
def goldbach(n):
    for i in range(2,n-2):
        if primo(i) and primo(n-i):
           return True
    return False # while 2 > 1: i = 1
```

Conjecture verificada para inteiros menores que 4×10^{18}

Números perfeitos

Um inteiro número é **perfeito** se ele é igual a soma de seus divisores positivos próprios. Por exemplo, 6 é perfeito pois $6 = 1 + 2 + 3$.

No século XVIII, Euler provou que todos os números perfeitos pares são da forma $(2^k - 1) \times 2^{(k-1)}$ para algum `k`. Não se sabe se existem infinitos números perfeitos pares.

É conjecturado que não existe números perfeitos ímpares.

Se existir, deve ser maior que 10^{300}. Ver *perfect number* na Wikipedia.

```
def perfeito():
    '''(None) -> ?
    Procura por um número ímpar perfeito
    '''
    n = 1
    while True
        soma = 0
```

```
    for i in range(n):
        if n % i == 0: soma += i
    print("soma dos divisores de %s = %s"%(n,soma))
    if soma == n: return
    n += 2 # próximo ímpar
```

Se existe algum número ímpar perfeito então esse programa pára: supondo que a memória do computador não acabe primeiro já que um tal inteiro pode ser muito grande.

Décimo problema de Hilbert

Em 1900, David Hilbert dirigiu-se ao Congresso Internacional de Matemáticos em Paris e colocou 23 problemas como um desafio para o próximo século XX. O décimo problema de Hilbert pedia para que fosse desenvolvido um processo (= programa, na nossa linguagem) segundo o qual podia ser determinado através de um *número finito* de operações se um dado polinômio multivariado possuia raizes inteiras.

Por exemplo, o polinômio $\texttt{f}(\texttt{x},\texttt{y},\texttt{z}) = 6\texttt{x}^3\texttt{yz}^2 + 3\texttt{xy}^2 - \texttt{x}^3 - 10$ tem uma raizes inteiras pois $\texttt{f}(5,3,0) = 0$. Já o polinômio $\texttt{f}(\texttt{x},\texttt{y}) = \texttt{x}^2 + \texttt{y}^2 - 3$ não tem raizes inteiras.

O problema tem mais de dois mil anos e nos leva aos tempo de Diofantino.

É facil fazermos um programa que para assim que encontra as raizes inteira de um polinômio dado, mas se não existir uma raiz?

Nos anos 70, o décimo problema de Hilbert foi resolvido de uma maneira muito surpreendente: Yuri Matiyasevich provou que um tal processo **não existe**. Assim, é impossível criar um programa que leia os coeficientes de um polinômio multivariado arbitrário como entrada e retorne a resposta sim-não apropriada!

Problema da parada

Halt problem: Escrever ums função `halt(f,x)` que recebe uma função `f()` e os seus argumentos `x`, e retorna `True` se `f(x)` para e `False` em caso contrário.

Um tal programa não existe!

Note que com isso *não estamos querendo dizer* que ainda não apareceu alguém esperto ou esperta o suficiente para fazer escrever a função `halt()`. Isso significa que ninguém não escreveu ou escreverá `halt()` pois ela não existe!

Como se demonstra uma coisa desse tipo?

A ideia da demonstração é baseado no seguinte paradoxo. A afirmação a seguir é verdadeira ou falsa? *Esta frase é falsa.*

Paradoxo do barbeiro. Suponha que João seja um barbeiro que diz que barbeia todas as pessoas da cidade que não se barbeiam e **somente** essas pessoas. Suponha ainda que João mora na cidade em que tem o sua barbearia. João se barbeia? Podemos demonstrar, por contradição, que esse barbeiro não pode existir :-)

Suponha, por contradição, que existe uma tal função `hash(f,x)` que retorna `True` se `f(x)` para e false se `f(x)` não para. Logo, `hash(f,x)` sempre para.

Considere a seguinte função

```
def estranha(f):
    if halt(f,f):
       while True: pass # halt(f,f) == True
    return # halt(f,f) == False
```

Considere que se `f(f)` *explode*, então `f(f)` para.

Logo:

- se `f(f)` não para, então `estranha(f)` para
- se `f(f)` para, então `estranha(f)` não para

Note que aqui `f` é nossa variável livre.

Agora, o que acontece quando chamamos `estranha(estranha)`? Ou seja, tome `f = estranha`.

- se `estranha(estranha)` não para, então `estranha(estranha)` para
- se `estranha(estranha)` para, então `estranha(estranha)` não para

Vixe!

Chegamos a uma contradição. Essa contradição é devida ao fato que supormos que uma tal função `halt()` possa existir.

28.13 Perspectiva

1. o caminho para desenvolver habilidades computacionais (nosso método pedagógico?)
2. MAC0110

- comandos básicos
- problemas básicos

3. MAC0122

- mais tipos de dados
- orientação a objetos criar e representar novos tipos (abstrair)
- mais ferramentas:
  - recursão
  - análise assintótica experimental e analítica
  - programação dinâmica
  - simulação
  - etc

4. Hoje

- computabilidade