Fractais com o *Mathematica*

E. Marques de Sá
DMUC, 2009

Dou alguns exemplos de fractais e respectivas imagens que podem facilmente obter-se usando o programa *Mathematica*. O texto explica brevemente a parte teórica das questões e o modo de implementação no *Mathematica*. No final, vão alguns desenhos ilustrativos.

O Triângulo de Sierpinski[1]

A forma mais simples de obter uma boa imagem deste fractal famoso é por geração duma nuvem de "pontos aleatórios", com as restrições que veremos.

Fixa-se um triângulo equilátero, de vértices A, B, C. Inicia-se um processo iterativo gerador de uma sequência de pontos $P_0, P_1, P_2, \ldots$ com as seguintes características:

$$P_k := \text{ponto médio do segmento } [P_{k-1}V_{k-1}], \text{ onde } V_{k-1}$$
$$\text{é escolhido aleatoriamente em } \{A, B, C\}.$$

Claro que teremos que escolher um ponto inicial da iteração; essa escolha é algo irrelevante. Eis uma implementação do algoritmo em que escolhi para pontos iniciais os vértices A, B, C que serão também representados. As letras usadas são minúsculas por motivos que já conhece.

```
a={0,0}; b={0,2}; c={1,Tan[Pi/3]}; o=(a+b+c)/3
n=20000; (*numero de pontos da nuvem*)
p[0]=a; p[1]=b; p[2]=c;
Do[p[i]=(p[i-1]+p[Random[Integer,{0, 2}]])/2, {i,3,n}]
nuvem=Table[{AbsolutePointSize[1], Point[p[i]]}, {i, 1, n}];
Show[Graphics[nuvem], AspectRatio -> Automatic];
```

A directiva `AbsolutePointSize[1]` significa que cada ponto da nuvem se representa por um píxel apenas. Veja desenhos no final do texto.

[1] Waclaw Sierpinski, 1882–1969.

O *Escaravelho* de Mandelbrot

O famoso escaravelho resulta do estudo da função complex $f_c(x) = z^2 + c$, onde c é um parâmetro complexo. Considera-se a sucessão $(z_0, z_1, \dots, z_n, \dots)$, definida por $z_0 = 0$, $z_{n+1} = f_c(z_n)$, chamada *órbita-c*. O *conjunto $\mathscr{M}$ de Mandelbrot* é, por definição, constituído pelos complexos c tais que a correspondente órbita-c é limitada.

Eis um lemma com interesse prático imediato:

Lema. *Um complexo c pertence a $\mathscr{M}$ se e só a órbita-c está toda dentro de $\mathscr{C}_2$, o círculo fechado de raio 2 e centro na origem. Em particular, $\mathscr{M}$ está contido em $\mathscr{C}_2$.*

Prova. Claro que, se a órbita-c está toda dentro de $\mathscr{C}_2$, então $c \in \mathscr{M}$. Reciprocamente, admitamos que $|z_w| > 2$, para certo w natural; vamos mostrar que $(|z_n|)$ converge para $+\infty$, o que implicará $c \notin \mathscr{M}$.

Primeiro vejamos o que acontece quando $|c| > 2$. Nesse caso $|z_1| = |c| > 2$. Admitamos, por hipótese indutiva, que $|z_k| \geqslant |c|$. Então

$$|z_{k+1}| \geqslant |z_k|^2 - |c| \geqslant |z_k|^2 - |z_k| \geqslant |z_k|(|c| - 1).$$

Obtemos, por indução, que $|z_n| \geqslant |c|$ e que $|z_{n+1}| > |c|(|c| - 1)^n$, para $n \geqslant 1$. Portanto a órbita-c converge para infinito.

Agora, o caso $|c| \leqslant 2$. Recorde-se que $|z_w| > 2$. Então

$$|z_{w+1}| \geqslant |z_w|^2 - |c| \geqslant |z_w|(|z_w| - 1) > |z_w|.$$

Por aqui se vê, por indução, que a sucessão $(|z_n|)$ é estritamente crescente para $n \geqslant w$. E resulta, também, que $|z_{w+t}| \geqslant |z_w|(|z_w| - 1)^t$, para todo o t positivo. Isto prova o que pretendíamos. $\square$

É costume usar estratégias de representação de $\mathscr{M}$ baseadas no “tempo de escape” da órbita-c. Aqui, *escape* significa cair fora do círculo $\mathscr{C}_2$. Por definição, o *tempo de escape* da órbita-c é (quando existe) o menor natural T tal que z_T está fora de $\mathscr{C}_2$. Quando todos os z_n estão em $\mathscr{C}_2$, dizemos que $T = +\infty$. Obtêm-se imagens de $\mathscr{M}$ muito interessantes colorindo cada ponto c de acordo com o seu tempo de escape.

Vamos desenhar um programa de representação gráfica de $\mathscr{M}$. Na instrução principal do programa tenta-se calcular, para cada complexo `c`, o respectivo tempo de escape, `tempo[c]`. O problema é que, para pontos $c \in \mathscr{M}$, por definição!, `tempo[c]`$= \infty$ e, para pontos fora de $\mathscr{M}$, mas próximos da fronteira de $\mathscr{M}$, `tempo[c]` pode atingir valores astronómicos. Portanto, a definição de `tempo[c]` tem que ser mitigada por algo que se possa calcular em prazo curto. A definição poderá ser

```
tempo[c_]:=(n = -1; z = 0;
    While[Abs[z]<=2 && n<Tmax, (z=z^2+c; n=n+1)]; n)
```

Aqui, `Tmax` é o tempo máximo admissível por nós fixado no início. O ciclo `While` vai ser repetido enquanto os sucessivos valores de `z` se mantêm dentro de $\mathscr{C}_2$ e `n` se mantém abaixo de `Tmax`. O tempo de execução do programa será tanto maior quanto maior for `Tmax`.

> Note que `c` e `z`, acima, são complexos e `z^2` é o quadrado complexo. Nós, humanos, por hábito e vício, identificamos o par `{a,b}` com o complexo `a+I*b`. Mas o *Mathematica* não sabe fazer essa identificação abusiva; mais precisamente, se o mandar calcular `{a,b}^2`, ele produz {a^2,b^2}. Conclusão, se quiser calcular `tempo[c]` tem que colocar um complexo no lugar de `c`; a execução de `tempo[{0,1}]` dá erro.
>
> Pode preferir outra definição de `tempo[{x,y}]` aplicável a pares ordenados. Eu prefiro a seguinte forma que vou utilizar no seguimento:

```
tempo[{a_,b_}]:=(t = -1; z = 0; c=a+I*b;
    While[Abs[z]<=2 && t<Tmax, (z=z^2+c; t=t+1)]; t)
```

Agora os acessórios do programa. Antes de tudo é preciso fixar um rectângulo de visualização em $\mathbb{R}^2$. Recomendo que comece por

```
RV={{x0,x1},{y0,y1}}={{-2.1,.8},{-1.2,1.2}};
```

Claro que apenas poderemos calcular os tempos de escape para um número finito de pontos de `RV`. As duas estratégias mais óbvias de escolha de pontos em `RV` são: (I) determinação de uma nuvem de pontos aleatórios, (II) determinação de uma grelha regular de pontos. (I) é mais fácil de executar, mas produz pinturas em estilo pontilhista; (II) dá mais trabalho a programar, mas os resultados são muito melhores.

O estilo pontilhista. Determina-se uma nuvem de `n` pontos aleatórios em `RV`:

```
pa:={Random[Real,{x0,x1}],Random[Real,{y0,y1}]}
nuvem=Table[pa,{j,1,n}]
```

Cada ponto da nuvem vai ser colorido de acordo com o seu tempo de escape. A cada tempo `t=0,1,...,Tmax` teremos que atribuir uma cor. A coisa mais simples será trabalhar com gradações de cinzento, usando a instrução GrayLevel[g], onde g é um real entre 0 e 1. Por exemplo, defina-se

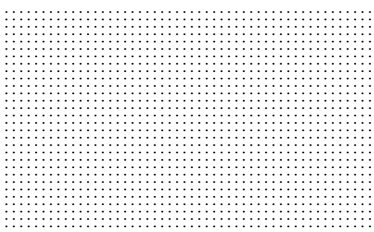
```
cor[t_]:=GrayLevel[1-t/Tmax]
```

Os pontos de $\mathscr{M}$ serão pintados a negro; além desses, serão negros todos os pontos `c` com tempos de escape $\geqslant$ `Tmax`. Quanto mais baixo for o valor de `Tmax` mais pontos haverá fora de $\mathscr{M}$ pintados a negro; isto diminui o recorte da figura nas imediações da fronteira de $\mathscr{M}$. Para melhorar o recorte, aumente `Tmax`; a figura será mais nítida mas, para o mesmo número `n` de pontos, demorará mais tempo a calcular.

O programa completo poderá ser o seguinte:

```
RV={{x0,x1},{y0,y1}}={{-2.1,.8},{-1.2,1.2}};
Tmax=60; n=20000;
tempo[{a_,b_}]:=(t = -1; z = 0; c=a+I*b;
            While[Abs[z]<=2 && t<Tmax, (z=z^2+c; t=t+1)]; t)
cor[t_]:=GrayLevel[1-t/Tmax]
pa:={Random[Real,{x0,x1}],Random[Real,{y0,y1}]}
nuvem=Table[pa,{j,1,n}]
pintura=Map[{cor[tempo[#]],Point[#]} &, nuvem];
Show[Graphics[pintura], AspectRatio->Automatic];
```

Grelha regular. Pensemos numa grelha regular de `nX`$\times$`nY` pontos, como na figura que representa o caso 50×30. Aqui, a nuvem tem os seus pontos igualmente espaçados. Se colorirmos cada ponto de acordo com o critério



anterior (*vd.* `pintura`), obtemos uma representação do conjunto de Mandelbrot. No programa, fixei `nX=150`, que é o número de colunas da grelha; o número de linhas, `nY`, calcula-se em função de nX e das dimensões do rectângulo de visualização, `RV`:

```
nX=150; d=(x1-x0)/nX; nY=Floor[(y1-y0)/d];
```

Note que cada ponto, com os seus vizinhos a norte, oeste e noroeste constituem os vértices dum quadradinho $\mathtt{d} \times \mathtt{d}$.

Feito deste modo, o desenho de $\mathscr{M}$ aparecerá com um aspecto pontilhado, regular mas pontilhado. Para evitar espaços vazios entre os pontos, damos a cada ponto o tamanho de 1 píxel —com a directiva `AbsolutePointSize[1]`— e mandamos que a imagem tenha largura de `nX` píxeis e altura `nY` píxeis — com a opção `ImageSize->{nX,nY}`. Aqui vai o programa completo (note que as funções `cor` e `tempo` estão definidas no programa anterior);

```
RV={{x0,x1},{y0,y1}}={{-2.1, .8},{-1.2,1.2}};
Tmax=60; nX=150; d=(x1 - x0)/nX; nY=Floor[(y1-y0)/d];
nuvem=Flatten[Table[{x0+i*d,y0+j*d},{i,0,nX},{j,0,nY}],1];
pintura=
  Map[{AbsolutePointSize[1],cor[tempo[#]],Point[#]}&,nuvem];
Show[Graphics[{pintura}], AspectRatio->Automatic,
                                        ImageSize->{nX,nY} ];
```

No final do texto, poderá comparar os resultados obtidos com nuvens de igual número de pontos, uma aleatória, a outra regular.

> Note uma consequência óbvia desta metodologia: o tamanho da imagem depende do `nX` fixado no início. Por exemplo, no final do texto, cada escaravelho tem 150 píxeis de largura, o que dá imagens pequenas. Mas, se fizer `nX=300`, o total de píxeis do boneco aumenta 4 vezes e o peso em bytes também (os escaravelhos no final pesam 3.25 Mb cada). Mais uma dica: no canto inferior esquerdo da janela do Mathematica, coloque o *zoom* a 100%; com mais do que isso, os píxeis não baterão certo com o que pretende, resultando uma imagem estilo tecido escocês.

Passeio aleatório

> Um indivíduo embriagado deambula aleatoriamente num plano. Os passos que dá têm todos o mesmo comprimento mas, após cada passo, a direcção do próximo é escolhida ao acaso. Desenhe uma trajectória de muitos passos nestas condições. Veja se descobre experimentalmente a relação (estatística!) entre o número n de passos dados e a maior distância à origem atingida no decorrer desse passeio de n passos.

A parte matemática deste problema pode ver-se assim: o dito indivíduo vai ocupando sucessivamente os pontos $P_0, P_1, \ldots, P_k, \ldots$. A relação entre P_n

e P_{n+1} é

$$P_{k+1} = P_k + (\cos\theta_k, \sin\theta_k)\,,$$

onde θ_k é aleatoriamente escolhido em $[0, 2\pi[$, em cada passo. Vamos supor que o indivíduo não dá preferência especial a nenhum sector para onde se deslocar, *i.e.*, a distribuição de θ em $[0, 2\pi[$ é uniforme. Vamos também supor que o ponto inicial é $P_0 = (0, 0)$.

A formulação na linguagem Mathematica pode ser esta: definimos passo a passo a função `p[k]`, definimos `caminho` como sendo a linha que une, ordenadamente, os `p[k]` com `k` de `0` a `n`. Depois basta mostrar o gráfico do modo habitual:

```
n=10000; p[0]={0, 0};
Do[t=Random[Real,{0,2Pi}]; p[k+1]=p[k]+{Cos[t],Sin[t]},
                                                  {k,0,n}]
caminho=Line[Table[p[k], {k, 0, n}]];
Show[Graphics[caminho], AspectRatio -> Automatic];
```