Terceiro Exercício-Programa – IAG – Atratores no Método de Newton

1. Método de Newton

O método de Newton é um método numérico para calcular aproximações de raízes de uma função real a partir de sua primeira derivada. Considere uma função diferenciável $f\colon \mathbb{R} \to \mathbb{R}$; dado um valor inicial x_0, podemos calcular aproximações sucessivas para uma raiz de f gerando uma sequência de valores

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n \geq 0.$$

Sob certas condições o método *converge*, isto é, o limite $\lim_{n\to\infty} x_n$ existe e é uma raiz de f. Não é difícil entender a idéia por trás do método: começamos com um chute x para uma raiz, então calculamos a tangente da função no ponto x (usando a derivada) e seguimos pela tangente na direção do zero (veja Figura 1).

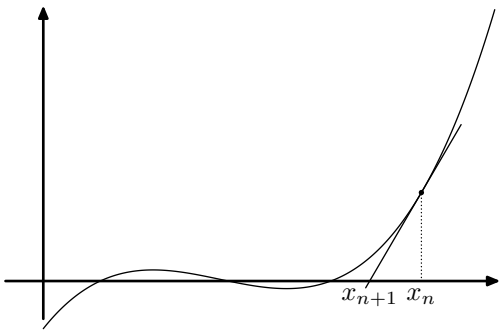

Figura 1. Uma iteração do método de Newton.

2. Bacias de atração

Funções de uma variável complexa também têm derivadas e seguem as mesmas regras de derivação que aprendemos no cálculo. Por exemplo, se $f(x) = (2+i)x^2 - ix + 4i$, então $f'(x) = (4+2i)x - i$; se $f(x) = \sin((1-2i)x)$, então $f'(x) = (1-2i)\cos((1-2i)x)$. O método de Newton também pode ser usado para calcular raízes de uma função complexa, bastando para tanto que possamos calcular a primeira derivada da função.

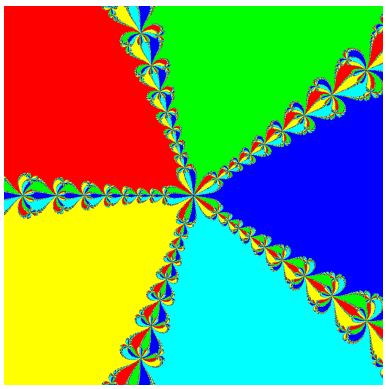

Figura 2. Bacias de atração para $f(x) = x^5 - 1$.

Um fenômeno interessante ocorre quando usamos o método de Newton para encontrar uma raiz de uma função complexa $f\colon \mathbb{C} \to \mathbb{C}$. Para cada ponto inicial x para o qual o método converge, obtemos uma raiz de f. Um conjunto de pontos iniciais que são levados pelo método de Newton a uma mesma raiz é chamado de *bacia de atração*. Podemos visualizar as bacias de atração do método de Newton fazendo um desenho no plano complexo no qual pintamos cada bacia de atração com uma cor diferente. Por exemplo, o polinômio $x^5 - 1$ tem cinco raízes complexas; pintando os pontos do quadrado complexo com extremos $(-5-5i)$ e $(5+5i)$ de acordo com as bacias de atração, obtemos a Figura 2.

Nosso objetivo neste EP é gerar imagens como a da Figura 2. Sua tarefa será dividida em partes, cada uma correspondendo à criação de uma função. Como de costume você deve usar o esqueleto de programa disponível no PACA.

3. Implementação do método de Newton

Primeiramente você deve implementar a função de protótipo

`newton(f, fp, x, eps, maxiter)`,

em que

(1) `f` e `fp` são uma função e sua derivada, respectivamente;
(2) `x` é o ponto inicial, um número complexo;
(3) `eps` é um real positivo, utilizado como critério de parada;
(4) `maxiter` é o número máximo de iterações do método de Newton que devemos executar.

Você deve ter algumas dúvidas no momento. Como podemos passar uma função como argumento para outra função? Como usar números complexos? Essas perguntas serão respondidas nesse enunciado; antes porém convém descrever em detalhes o funcionamento da função `newton`. Cada iteração do método de Newton deve transcorrer da seguinte forma:

(1) No início da iteração, temos um valor x que esperamos estar próximo de uma raiz. Se $|f(x)| <$ `eps`, então x está próximo o bastante de uma raiz: a função `newton` devolve o par (k, x), em que k é o número de iterações que foram concluídas até o momento.
(2) Caso x não esteja próximo de uma raiz, então calculamos a derivada $f'(x)$. Se $|f'(x)| <$ `eps`, então não podemos prosseguir, ou faríamos uma divisão por um número muito próximo de zero; o método de Newton falha e a função `newton` devolve $(-1, 0)$. Caso contrário, substituímos x por $x - f(x)/f'(x)$ e passamos para a próxima iteração.

Se `maxiter` iterações forem executadas sem que uma boa aproximação seja encontrada, então o método de Newton também falha, e a função `newton` devolve $(-1, 0)$.

Você deve implementar o método de Newton exatamente como descrito acima. A única complicação que pode ocorrer é que pode não ser possível calcular $f(x)$ ou $f'(x)$ durante alguma iteração; por exemplo, $\log 0$ não está definido. Nesses

casos, as funções `f` e `fp` devolvem o valor especial `None`. Caso em alguma iteração uma dessas funções devolver `None`, a função `newton` deve devolver $(-1, 0)$ para indicar que falhou. Para testar se um valor é `None`, você deve usar o operador `is`, como no exemplo:

```
val = f(x)
if val is None: return (-1, 0)
# val não é None, deve ser o valor de f em x.
```

3.1. **Números complexos.** Python tem o tipo `complex`, que pode ser usado para representar números complexos:

```
>> a = complex(1, 2)
>> b = complex(0, 1)
>> print(a)
(1+2j)
>> print(b)
1j
>> b ** 2
(-1+0j)
>> abs(a + b)
3.1622776601683795
```

Note que você pode operar livremente com números complexos, da mesma forma que opera com números reais. A função `abs` devolve o módulo de um número complexo.

3.2. **Funções como argumentos.** Em Python, funções são objetos assim como listas, números ou strings. Em particular, elas podem ser passadas como argumentos para outras funções. Considere o seguinte exemplo simples:

```
def soma2(x):
    return x + 2

def aplica(f, L):
    """Aplica f a cada elemento de L e devolve lista resultante."""

    ret = []
    for x in L:
        ret.append(f(x))

    return ret

def main():
    L = aplica(soma2, [ 1, 2, 3, 4 ])
    print(L)
```

Python inclusive fornece uma implementação própria da função `aplica` definida acima, chamada `map`.

A função `newton` recebe duas funções como argumentos. Você pode testar sua implementação da seguinte forma:

```
def p(x): return x ** 5 - 1
def q(x): return 5 * x ** 4

def main():
    print(newton(p, q, complex(-1, 2), 1e-8, 100))
```

Acima, `1e-8` é uma forma de representar o número 10^{-8}. A saída do seu programa deve ser algo como

`(12, (-0.8090169943832873+0.5877852522889782j))`.

4. Geração dos dados para a imagem

Vejamos agora como usar a função `newton` de modo a gerar os dados para fazer a imagem das bacias de atração do método de Newton. Dado um retângulo no plano complexo, consideramos uma grade de pontos uniformemente espaçados dentro do retângulo e os usamos como pontos iniciais para o método de Newton. Cada ponto será levado a uma raiz diferente da função; mais adiante vamos identificar quais pontos foram levados numa mesma raiz para pintá-los com uma mesma cor.

Você deve escrever uma função de protótipo

`faz_matriz(f, fp, x1, y1, x2, y2, N, eps, maxiter)`,

em que

(1) `f` e `fp` são uma função e sua derivada, respectivamente;
(2) `x1`, `y1` e `x2`, `y2` são as coordenadas do cantos *esquerdo inferior* e *direito superior* de um retângulo no plano complexo;

(3) `N` é o número de pontos da grade no lado menor do retângulo;
(4) `eps` e `maxiter` são parâmetros para a função `newton`.

Defina[1] no seu programa $\delta = \min\{\texttt{y2}-\texttt{y1}, \texttt{x2}-\texttt{x1}\}/\texttt{N}$, $m = 1 + \lceil(\texttt{y2}-\texttt{y1})/\delta\rceil$ e $n = 1 + \lceil(\texttt{x2}-\texttt{x1})/\delta\rceil$ (veja Figura 3). Sua função deve devolver matrizes I e R, ambas $m \times n$, tais que $I[i][j] = k$ e $R[i][j] = x$ para cada $0 \le i < m$ e $0 \le j < n$, onde

```
k,x = newton(f, fp, complex(x1+j*delta, y2-i*delta), eps, maxiter)
```

Assim, para cada ponto (i, j) da grade, a matriz I contém o número de iterações executadas pelo método de Newton (ou -1 para indicar que o método falhou) e a matriz R contém a aproximação da raiz encontrada (ou 0 em caso de falha). A assimetria entre a expressão da parte real e da parte complexa do ponto inicial se deve à forma como imagens são representadas no computador: quando desenhamos o gráfico de uma função no papel, os valores do eixo vertical aumentam de baixo para cima, enquanto numa imagem de computador o eixo vertical aumenta de cima para baixo (o pixel no topo esquerdo superior tem coordenadas $(0, 0)$, o pixel logo abaixo tem coordenadas $(0, 1)$ e assim por diante).

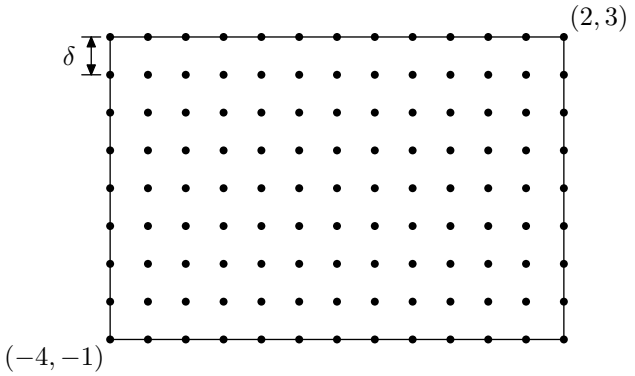

FIGURA 3. Escolha de grade para um retângulo no plano complexo definido por $(x_1, y_1) = (-4, -1)$ e $(x_2, y_2) = (2, 3)$. Se $\texttt{N} = 8$ então temos $\delta = 0.5$, $m = 9$ e $n = 13$.

5. Atribuição de cores

Uma vez que temos as matrizes I e R calculadas pela função `faz_matriz`, precisamos atribuir a cada raiz uma cor diferente. O problema é que o método de Newton é um método numérico, que executa um certo número de iterações até satisfazer um critério aproximado de convergência. Além disso, usamos aritmética de ponto flutuante, que é em si fonte de imprecisões. Por isso, se `x` e `y` são duas aproximações de raízes devolvidas pela função `newton`, faz pouco sentido fazer a comparação `x == y`; o que devemos verificar é se as duas aproximações estão próximas o bastante uma da outra. Em outras palavras, nosso problema é: para cada ponto (i, j) da grade tal que $I[i][j] \neq -1$ temos que $R[i][j]$ é uma raiz[2] aproximada da função; queremos decidir quais raízes aproximadas devemos considerar como sendo iguais e quais não.

Há diversas abordagens para resolver problemas desse tipo[3]. Em nosso programa vamos usar uma abordagem iterativa: a primeira raiz $c[0] = R[i_0][j_0]$ receberá uma certa cor X; todas as demais raizes $R[i][j]$ que estiverem a uma distância $|R[i][j]-c[0]| \le \varrho$ (onde $\varrho > 0$ é um parâmetro dado pelo usuário) receberão a mesma cor X. A próxima raiz $c[1] = R[i_1][j_1]$ que não estiver suficientemente próxima de $c[0]$ receberá uma cor Y, e assim por diante. Note que você deve guardar as raízes $c[0], c[1], \ldots, c[k]$ usadas para definir as cores das bacias de atração, e testar para cada novo $R[i][j]$ se ele é ou não é próximo o suficiente de uma das raízes já conhecidas (ou seja, se $|R[i][j] - c[l]| \le \varrho$ para $l = 0, \ldots, k$) a fim de receber a mesma cor da raiz correspondente. Apenas no caso em que uma nova raiz $R[i][j]$ não for próxima o suficiente de nenhuma das raízes $c[0], c[1], \ldots, c[k]$ já conhecidas, você deve definir $c[k + 1] = R[i][j]$ e atribuir a essa posição uma nova cor Z.

Nas imagens coloridas que iremos gerar, uma cor é uma tripla (r, g, b) de valores inteiros no intervalo $[0, 255]$, que correspondem às intensidades das componentes azul, verde e vermelha, respectivamente. Por exemplo, a tripla $(255, 0, 0)$ representa a cor azul; as triplas $(0, 0, 0)$ e $(255, 255, 255)$ representam preto e branco, respectivamente. Para obter as cores das bacias de atração, você deve usar a função `nova_cor` que se encontra no esqueleto do EP; cada chamada a essa função devolve uma tripla que representa uma cor nova.

Para deixar a imagem ainda mais bonita, usaremos um parâmetro $\alpha \in [0, 1]$ adicional que controlará o sombreamento da bacia (veja Figura 4) através de um fator de escala $(1-\alpha*s[k]/(m*n))$ aplicado a cada pixel (i, j) com $I[i][j] = k \ge 0$[4], onde $s[k]$ é a quantidade de elementos $I[i][j]$ que satisfaz $0 \le I[i][j] < k$, para $k = 0, 1, \ldots, \text{MAX} + 1$ sendo MAX o maior elemento da matriz I. Em outras palavras, se a raiz $R[i][j]$ pertence à bacia com cor (r, g, b) então o pixel (i, j) receberá a cor $(\lfloor\beta_{ij} * r\rfloor, \lfloor\beta_{ij} * g\rfloor, \lfloor\beta_{ij} * b\rfloor)$ onde $\beta_{ij} = (1 - \alpha * s[k]/(m * n))$. O parâmetro α pode ser usado para controlar o quanto o fator de escala depende do número de iterações. Se $\alpha = 0$, então o fator de escala será sempre 1, independentemente do número de iterações: obtemos assim uma figura na qual cada bacia de atração recebe uma cor uniforme (como a Figura 2). Se $\alpha = 1$, os pesos dependem totalmente do número de iterações. Para gerar a Figura 4 usou-se $\alpha = 0.8$.

---

[1] Os símbolos $\lceil x\rceil$ e $\lfloor x\rfloor$ representam respectivamente o teto (arredondamento para cima) e o chão (arredondamento para baixo) do valor x, e podem ser computados em Python com as funções `ceil` e `floor`.

[2] Lembre que as posições da matriz tais que $I[i][j] = -1$ correspondem a pontos iniciais para os quais o método de Newton falhou, e os valores $R[i][j] = 0$ associados não são raízes da função. Sempre que falarmos que $R[i][j]$ é uma raiz você já deve subentender que $I[i][j] \neq -1$.

[3] Este é um tipo de problema comum em computação, chamado de problema de *aglomeração* (em inglês, *clustering*): temos diversos pontos no espaço (que podem representar dados como obras literárias, escolhas de clientes numa loja virtual, sequências de DNA, etc.) e queremos agrupar esses pontos de acordo com alguma medida de similaridade.

[4] Portanto atenção: essa fórmula não se aplica aos pixels pretos com $I[i][j] = -1$.

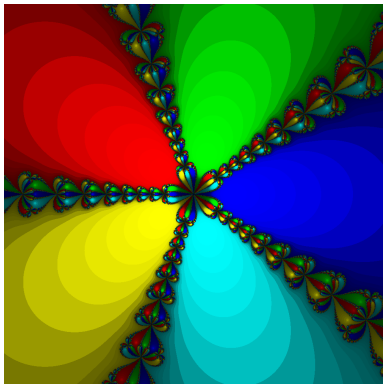

FIGURA 4. Bacias de atração para $f(x) = x^5 - 1$ dentro do quadrado entre $-5 - 5i$ e $5 + 5i$; quanto maior o número de iterações executadas pelo método de Newton num dado ponto, mais escura a cor a ele associada.

Para gerar a matriz de cores você deve escrever uma função

`determina_cores(I, R, ro, alfa)`

que recebe as matrizes I e R devolvidas pela função `faz_matriz` e os parâmetros `ro` $= \varrho$ e `alfa` $= \alpha$ usados na descrição acima, e devolve uma matriz C (com as mesmas dimensões de R) onde cada entrada $C[i][j]$ tal que $I[i][j] = -1$ recebe a cor preta $(0, 0, 0)$, e cada entrada associada a uma raiz recebe a cor da bacia de atração correspondente.

6. Gravação da imagem

Para gravar a imagem de saída usaremos o formato PPM[5], que permite que imagens sejam armazenadas em arquivos de texto. Eis um exemplo de arquivo PPM:

```
P3
3 2
255
255 0 0 0 255 0 0 0 255
38 20 255 0 0 0 255 255 255
```

A primeira linha do arquivo é, por convenção, sempre igual a '`P3`'. A segunda linha traz o número de colunas (largura) e o número de linhas (altura) da imagem, em pixels; a imagem acima, por exemplo, tem largura 3 e altura 2. A terceira linha traz o valor que representa a maior intensidade de cor; no nosso caso, este valor será sempre 255. Cada uma das linhas seguintes representa uma linha da imagem. Cada pixel é representado por três valores consecutivos, um para cada cor fundamental (vermelho, verde e azul). Por exemplo, a primeira linha da imagem tem 3 pixels, o primeiro vermelho, o segundo verde e o terceiro azul; o último pixel da segunda linha é branco.

Você deve escrever uma função de protótipo

`grava_imagem(C, nome_arquivo)`

que recebe a matriz de cores `C` processada pela função `determina_cores` e o nome de um arquivo, e grava a imagem correspondente no arquivo de saída usando o formato PPM.

7. Programa principal

A função `main` já se encontra escrita no esqueleto do EP; você não deve alterá-la. Ela chama as demais funções que você deve implementar para gerar uma imagem.

Para usar o programa final, você deve criar um arquivo de texto contendo a função a ser usada e sua primeira derivada, cada uma em uma linha. A variável da função deve ser `x`. Você pode definir a função escrevendo sua expressão em Python normalmente; você também pode usar as seguintes funções do Python em sua definição:

`sin`, `cos`, `sinh`, `cosh`, `tan`, `exp`, `log`.

Por exemplo, para gerar a Figura 4, você pode criar um arquivo chamado, por exemplo, '`p5.txt`', com o conteúdo:

```
x ** 5 - 1
5 * x ** 4
```

Basta então executar o programa, informando os parâmetros desejados:

```
> python ep3.py
EP3 - Atratores do Método de Newton

Arquivo com as funções: p5.txt
Arquivo de saída:       p5.ppm

Parâmetros da grade
x1: -5
```

[5] Vários programas de visualização de imagens aceitam esse formato. Procure na Wikipedia por "netpbm format" ou veja `https://fileinfo.com/extension/ppm`

```
y1: -5
x2: 5
y2: 5
N : 1000

Parâmetros para o Método de Newton
eps:     1e-8
maxiter: 100

Parâmetros para coloração
ro:   1e-5
alfa: 0.8

Calculando imagem... pronto.
```

Considere também os arquivos de entrada '`trig.txt`':

```
sin(complex(1, 1) * x) + cos(complex(0, 2) * x)
complex(1, 1) * cos(complex(1, 1) * x) - complex(0, 2) * sin(complex(0, 2) * x)
```

e '`pretty.txt`':

```
x ** 8 + 3 * x ** 4 - 4
8 * x ** 7 + 12 * x ** 2
```

Usando esses arquivos com os parâmetros abaixo, obtemos as Figuras 5 e 6.

```
> python ep3.py
EP3 - Atratores do Método de Newton

Arquivo com as funções: trig.txt
Arquivo de saída:       trig.ppm

Parâmetros da grade
x1: -3
y1: -2
x2: 3
y2: 2
N : 400

Parâmetros para o Método de Newton
eps:     1e-8
maxiter: 100

Parâmetros para coloração
ro:   1e-5
alfa: 0.8

Calculando imagem... pronto.
```

```
> python ep3.py
EP3 - Atratores do Método de Newton

Arquivo com as funções: pretty.txt
Arquivo de saída:       pretty.ppm

Parâmetros da grade
x1: 0.09
y1: 0.08
x2: 0.1
y2: 0.091
N : 600

Parâmetros para o Método de Newton
eps:     1e-5
maxiter: 100

Parâmetros para coloração
ro:   1e-5
alfa: 0.8

Calculando imagem... pronto.
```

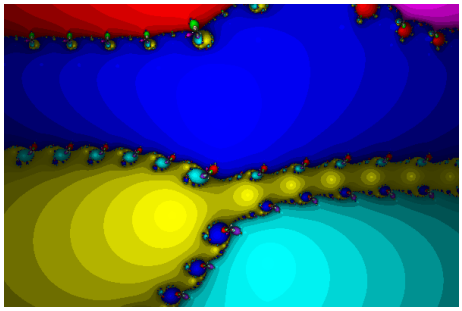

FIGURA 5. Imagem obtida do arquivo '`trig.txt`'.

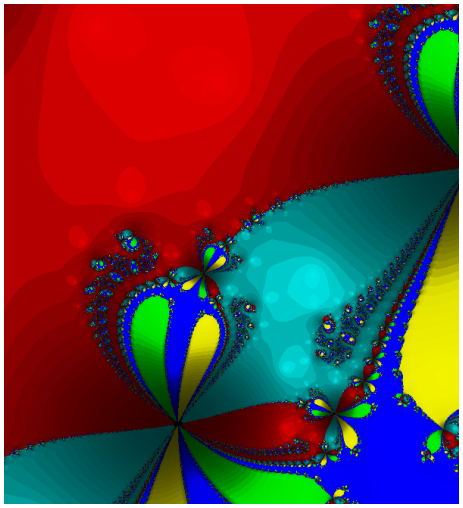

FIGURA 6. Imagem obtida do arquivo '`pretty.txt`'.