Método de Newton para calcular raízes de polinômios

Exercício Computacional
MAP3122 - Quadrimestral 2020
Prof. Antoine Laurain

Seja $f(x) = p(x)/q(x)$, onde $p, q : \mathbb{R} \to \mathbb{R}$ são duas funções diferenciáveis.

Questão 1. *Verifique que o método de Newton para calcular uma raiz de f pode ser escrito na forma*

$$x_{k+1} = x_k - \frac{1}{\frac{p'(x_k)}{p(x_k)} - \frac{q'(x_k)}{q(x_k)}}. \tag{1}$$

Podemos usar esta propriedade para calcular todas as raízes de um polinômio

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

com coeficientes a_k reais ou complexos. Observe que um polinômio de ordem n sempre tem n raízes complexas (não necessariamente distintas), esse é o teorema fundamental da álgebra. Chamamos então de $z_1, \ldots, z_n \in \mathbb{C}$ as raízes complexas de p. A tarefa principal desse exercício prático é de calcular uma aproximação numérica de todas as raízes z_k.

Começamos calculando uma raiz de p usando o método de Newton usual. Para calcular as outras raízes, usamos o raciocínio seguinte. Primeiro, o polinômio p pode ser escrito como

$$p(x) = a_n(x - z_1)(x - z_2)\ldots(x - z_n).$$

Suponhamos que já temos a disposição as raízes $z_1, z_2, \ldots, z_\ell$, com $1 \leq \ell < n$. Definimos

$$q(x) = (x - z_1)(x - z_2)\ldots(x - z_\ell).$$

Observamos que

$$f(x) = \frac{p(x)}{q(x)} = a_1(x - z_{\ell+1})\ldots(x - z_n).$$

Esta função f é um polinômio cujas raízes são exatamente $z_{\ell+1}, z_{\ell+2}, \ldots, z_n$. Assim, aplicando o método de Newton para f, podemos achar uma nova raiz de p, diferente das raízes de p já calculadas. Como f é da forma $f = p/q$, podemos usar a fórmula (1). Observamos que a derivada de $q(x)$ satisfaz a propriedade seguinte:

$$\frac{q'(x)}{q(x)} = \frac{1}{x - z_1} + \frac{1}{x - z_2} + \cdots + \frac{1}{x - z_\ell}. \tag{2}$$

Vamos usar a expressão (2) na iteração de Newton (1).

Cada nova raiz calculada com esse método tem um valor mais impreciso que a anterior, uma vez que as raízes já calculadas contem erros, e são usadas no método de Newton para calcular uma nova raiz, assim os erros vão se acumulando. Portanto, depois de aproximar uma raiz z_k usando (1), vamos recalcular uma nova aproximação de z_k usando o método de Newton simples para o polinômio p. Para inicializar essa iteração de Newton, use o valor z_k obtido com o outro algoritmo.

O objetivo deste exercício prático é de escrever uma função `z=polyzeros(a)`, que calcula todas as raízes de um polinômio $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ dado, onde $a = [a_0, a_1, \ldots, a_n]$ é o vetor dos coeficientes de p. A função `z=polyzeros(a)` tem o vetor a como entrada e o vetor das raízes $z = [z_1, \ldots, z_n]$ como saída.

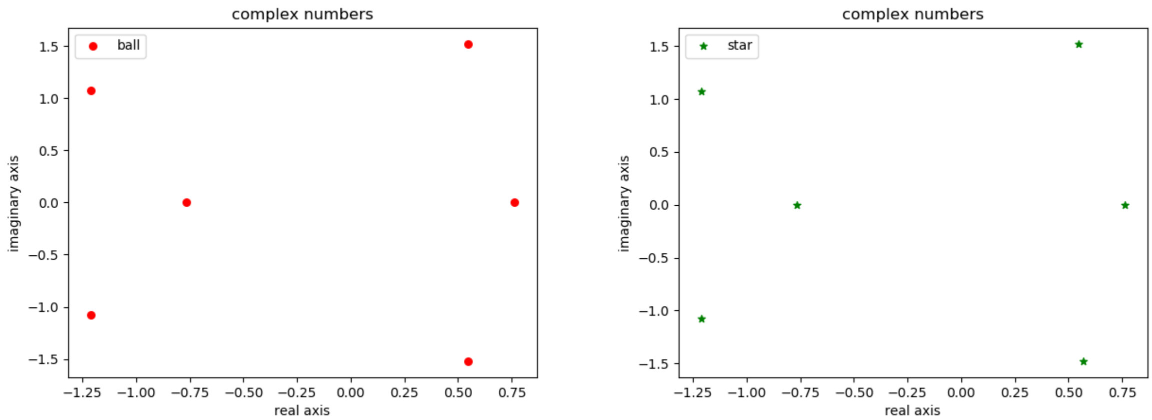

Figura 1: Comparação entre as 6 raízes do polinômio $p(x) = 1.5x^6 + 2x^5 + 3x^4 + 4x^3 + 8x^2 - 3x - 6$ calculadas com a função `numpy.roots(p)` (esquerda) e as raízes calculadas com a função `z=polyzeros(a)` (direita).

Instruções, observações e dicas

- O programa deverá ser escrito em Python, usando o pacote `numpy`. O seu código deverá estar bem comentado e estruturado. A entrada e a saída deverão ser feitas de forma a ajudar o usuário a executar o programa e devem facilitar a análise dos resultados. Se o seu programa precisa de arquivos de entrada, considere que os mesmos encontram-se na mesma pasta do executável, ou faça de forma que solicite o caminho/nome do arquivo ao usuário.
- A entrega deverá conter um relatório (em .pdf), contendo a análise do problema estudado, e o código usado para as simulações computacionais (arquivo .py). A entrega também deverá ser feita em um arquivo compactado único.
- O uso de LaTeX para escrever o relatório é fortemente incentivado. Os relatórios escritos em Latex receberão um bónus de 0.5 pontos.
- Números complexos em Python têm a forma `a+b*1j` ou `a+bj`, onde `a`, `b` são valores numéricos.
- Para capturar as raízes complexas, você precisará usar valores iniciais complexos no método de Newton.
- Para a inicialização do método de Newton, será melhor utilizar valores iniciais aleatórios. O comando para isto é `numpy.random.rand`
- Um polinômio p poderá ser definido a partir de um vetor de coeficientes utilizando a função `numpy.poly1d`. A derivada de um tal polinômio p poderá ser calculada com `numpy.polyder`. O valor de p em um ponto particular também poderá ser calculado usando a função `numpy.polyval`.
- Será necessário usar um critério de parada para o método de Newton. O módulo $|p(x_k)|$ de $p(x_k)$ (cuidado, pois aqui $p(x_k)$ é um número complexo) pode ser usado para um critério de parada da seguinte maneira. A cada iteração, a condição $|p(x_k)| < \varepsilon$ é testada, onde ε é um valor pequeno escolhido pelo usuário. Se $|p(x_k)| < \varepsilon$ é satisfeita, então paramos a iteração de Newton, caso contrário, continuamos. Podemos escolher por exemplo $\varepsilon = 10^{-16}$; experimente com outros valores para ver se isto muda o resultado. Além disto, é necessário impor um número máximo de iterações `itmax` para evitar um loop infinito. Um valor razoável para o método de Newton é `itmax` entre 7 e 10.
- Como usamos valores aleatórios para inicializar a iteração de Newton, acontece as vezes que o algoritmo fornece valores errados ou valores do tipo `NaN`. Por isso, é aconselhável rodar algumas vezes o programa para cada exemplo, para eliminar casos degenerados. Isso é particularmente verdadeiro quando a ordem do polinômio fica maior e os resultados do programa se tornam mais instáveis.
- Vamos testar o programa usando dois métodos:

1. Escolha $z_1, \dots, z_n \in \mathbb{C}$, e calcule os coeficientes a_k do polinômio $p(x) = (x - z_1)(x - z_2) \dots (x - z_n)$. Depois use o programa com esses coeficientes. O programa deverá entregar as mesmas raízes $z_1, \dots, z_n \in \mathbb{C}$, ou uma boa aproximação delas (ver Tabela 1).
2. Para um polinômio dado na forma $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, compare o resultado obtido com sua função `z=polyzeros(a)` e com a função `numpy.roots(p)` (ver Tabela 2).

- Os resultados de sua função `z=polyzeros(a)` e de `numpy.roots(p)` devem ser plotados em duas figuras diferentes, como na Figura 1. Cada raiz será representada por um ponto em cada figura. O eixo horizontal representa a parte real das raízes, e o eixo vertical a parte imaginária. Deve-se observar que as duas figuras fornecem visualmente os mesmos pontos para alguns exemplos. Observe que quando os coeficientes $a = [a_0, a_1, \dots, a_n]$ são reais, as raízes complexas z_k sempre são *conjugadas*, i.e. para cada raiz da forma $z = a + ib$ com $b \neq 0$, existe uma outra raiz conjugada $\overline{z} = a - ib$; esse fenômeno pode ser observado na Figura 1. Você deveria observar isso na suas figuras também.

Forma do relatório

O relatório (em .pdf) será constituído dos seguintes elementos:

- Resposta à Questão 1.
- Testes e análise de erro:
  - Escolha dois polinômios (de ordem entre 4 e 10) e para cada um destes polinômios, faça uma análise de erro seguindo as instruções da Tabela 1. Comente os resultados.
  - Escolha dois polinômios (de ordem entre 5 e 10) e para cada um destes polinômios, faça uma análise de resíduo do tipo da Tabela 2. Comente os resultados.
- Figuras: escolha três polinômios (de ordem entre 4 e 10) e plote as raízes obtidas com sua função `z=polyzeros(a)` e com a função `numpy.roots(p)` em duas figuras separadas, como na Figura 1. Comente os resultados. Um ponto interessante, é de discutir a estabilidade do cálculo das raízes com respeito à ordem dos polinômios, usando `z=polyzeros(a)` e `numpy.roots(p)`.

Critérios de Correção

- Questão 1 (0.3 pts)
- Implementação de `z=polyzeros(a)` (serão julgadas: a exatidão dos resultados e a eficiência da implementação).
  - Método de Newton para raízes de $f(x) = p(x)/q(x)$. (1.6 pts)
  - Método de Newton para refinamento das raízes de $p(x)$. (1.6 pts)
- Código bem documentado: comentários, legibilidade. (1.5 pts)
- Testes e análise de erro (relevância dos testes, apresentação dos resultados e comentários).
  - Análise de erro do tipo da Tabela 1. (1.5 pts)
  - Análise de resíduo do tipo da Tabela 2. (1.5 pts)
- Figuras (relevância dos testes, qualidade das figuras). (1 pts)
- Qualidade do relatório (relevância dos comentários e apresentação geral). (1 pts)
- Uso de LaTeX(0.5 pts extra)
- Será verificado se o programa entregue roda e produz saídas consistentes com os resultados apresentados no relatório.
- Em caso de atraso de até 48h, -2 pontos. Após isso, o EP não será aceito.

Tabela 1: Tabela comparando os erros cometidos usando as funções `z=polyzeros(a)` e `numpy.roots(p)` para um polinômio p conhecido cujas raizes são conhecidas. Para este tipo de tabela, escolha as raizes exatas $z_1, \ldots, z_k \in \mathbb{R}$, e considera o polinômio $p(x) = (x - z_1)(x - z_2) \ldots (x - z_n)$. Como as raizes são reais, é fácil ordenar as raizes obtidas com `z=polyzeros(a)` e `numpy.roots(p)` para calcular os erros. Pode usar tambem raizes complexas, mas fica mais difícil ordenar as raizes para calcular os erros.

raizes exatas	`polyzeros`	erro `polyzeros`	`numpy.roots`	erro `numpy.roots`
z_1	$\tilde{z}_1$	$\lvert z_1 - \tilde{z}_1\rvert$	$\hat{z}_1$	$\lvert z_1 - \hat{z}_1\rvert$
z_2	$\tilde{z}_2$	$\lvert z_2 - \tilde{z}_2\rvert$	$\hat{z}_2$	$\lvert z_2 - \hat{z}_2\rvert$
z_3	$\tilde{z}_3$	$\lvert z_3 - \tilde{z}_3\rvert$	$\hat{z}_3$	$\lvert z_3 - \hat{z}_3\rvert$
z_4	$\tilde{z}_4$	$\lvert z_4 - \tilde{z}_4\rvert$	$\hat{z}_4$	$\lvert z_4 - \hat{z}_4\rvert$

Tabela 2: Neste caso temos um polinômio dado na forma $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, e calculamos as raizes (que podem ser complexas) usando `z=polyzeros(a)` e `numpy.roots(p)`. Como as raizes exatas não são conhecidas, não podemos calcular o erro cometido como na Tabela 1, mas podemos calcular o *resíduo*, que é simplesmente o valor do polinômio p em uma raiz aproximada. Se a raiz for exata, o resíduo vale zero, então o resíduo deveria ser o mínimo possível. Assim, o valor do résiduo fornece uma informação sobre a precisão da aproximação da raiz.

`polyzeros`	resíduo `polyzeros`	`numpy.roots`	resíduo `numpy.roots`
$\tilde{z}_1$	$\lvert p(\tilde{z}_1)\rvert$	$\hat{z}_1$	$\lvert p(\hat{z}_1)\rvert$
$\tilde{z}_2$	$\lvert p(\tilde{z}_2)\rvert$	$\hat{z}_2$	$\lvert p(\hat{z}_2)\rvert$
$\tilde{z}_3$	$\lvert p(\tilde{z}_3)\rvert$	$\hat{z}_3$	$\lvert p(\hat{z}_3)\rvert$
$\tilde{z}_4$	$\lvert p(\tilde{z}_4)\rvert$	$\hat{z}_4$	$\lvert p(\hat{z}_4)\rvert$