EP1 - Cálculo Numérico

Professor: Arnaldo Gammal
Monitor: Raphael

Aluno: Jhoão Gabriel Martins Campos de Almeida Arneiro
Nº USP: 10819721

Item a

Para resolver a equação $x^3 - \cos(x) = 0$ pelo método da bissecção, nós devemos nos atentar em utilizar uma função da forma $f(x) = 0$. Felizmente, a equação já se coloca desta forma e assim o nosso problema se resume a encontrar a raiz da função $f(x) = x^3 - \cos(x) = 0$, utilizando o método de bissecção. Para isso, nós começamos avaliando um intervalo de -2 a 2, e para escolher um critério de parada para o programa recordamos que, utilizando precisão simples, o computador armazena até 7 casas decimais. Sendo assim, o critério de parada ϵ é escolhido como sendo de 6 casas decimais ($\epsilon = 0.000001$), de forma que a precisão do nosso programa não entra em conflito com a precisão do computador, evitando possíveis erros por arredondamento. Apesar de que o Python se utilize, por padrão, da dupla precisão, a margem de 6 casas decimais já se faz uma ótima aproximação do valor exato da raiz, não sendo necessário reduzir ϵ para um intervalo ainda menor.

Abaixo segue o programa utilizado para encontrar a raiz, seguido de uma tabela com os valores relevantes ao método de bissecção, isto é, o limite inferior do intervalo de análise, x_1, o limite superior, x_2, a média simples dos limites (ou o meio do intervalo, se preferir), x_m, a função avaliada no início do intervalo, $f(x_1)$, a função avaliada no meio do intervalo, $f(x_m)$, e por fim, o tamanho do intervalo, $e_n = x_2 - x_1$.

```python
from math import cos

#Constantes
epsilon = 0.000001

#Funcoes uteis
def f(x):
    ''' Funcao que toma um valor de x e calcula o valor da funcao
        f(x) = x^3 - cos(x) neste ponto '''

    f = x**3 - cos(x)

    return f
```

```python
def bisec(f,a,b):
    ''' A partir de um dado intervalo [a,b] e uma funcao f(x), eh calculada
        a raiz de f(x) a partir do metodo da bisseccao '''

    #O intervalo comeca indo de a ate b
    x_1 = a
    x_2 = b

    #Realizamos um loop ate que o tamanho do intervalo seja menor que a
        precisao
    #desejada
    while abs(x_2 - x_1) > epsilon:

        #Eh calculado o ponto medio entre x_1 e x_2
        x_m = (x_1 + x_2)/2

        #Se a funcao calculada nos pontos x_1 e x_m tiverem o mesmo sinal,
        #entao o seu produto sera positivo, e o intervalo passa a ser
        #[x_1,x_2] -> [x_m,x_2]
        if f(x_1)*f(x_m) > 0:

            x_1 = x_m

        #Caso o produto das duas seja negativo, entao elas possuem sinal
            diferente
        #e o intervalo eh atualizado para [x_1,x_2] -> [x_1,x_m]
        else:

            x_2 = x_m

    #Ao fim do loop, tomamos o valor de x_1 como sendo o valor da raiz de f(x)
    return x_1

#Inicio do programa
#Eh escolhido o intervalo de [-2,2]
x_a = -2
x_b = 2

#A raiz x_0 da funcao eh calculado usando o algoritmo de bisseccao
x_0 = bisec(f,x_a,x_b)

#Tendo o valor de x_0, imprimimos ele
print("A raiz x_0 da funcao eh: %.6f" %x_0)

>>> A raiz x_0 da funcao eh: 0.865474
```

x_1	x_2	x_m	$f(x_1)$	$f(x_m)$	e_n
-2	2	0	-7.58385	-1	4
0	2	1	-1	0.459698	2
0	1	0.5	-1	-0.752583	1
0.5	1	0.75	-0.752583	-0.309814	0.5
0.75	1	0.875	-0.309814	0.028925	0.25
0.75	0.875	0.8125	-0.309814	-0.151309	0.125
0.8125	0.875	0.84375	-0.151309	-0.0639882	0.0625
0.84375	0.875	0.859375	-0.0639882	-0.0182407	0.03125
0.859375	0.875	0.867188	-0.0182407	0.00516361	0.015625
0.859375	0.867188	0.863281	-0.0182407	-0.00658304	0.0078125
0.863281	0.867188	0.865234	-0.00658304	-0.000720853	0.00390625
0.865234	0.867188	0.866211	-0.000720853	0.00221859	0.00195312
0.865234	0.866211	0.865723	-0.000720853	0.000748173	0.000976562
0.865234	0.865723	0.865479	-0.000720853	1.34859e-05	0.000488281
0.865234	0.865479	0.865356	-0.000720853	-0.000353727	0.000244141
0.865356	0.865479	0.865417	-0.000353727	-0.000170131	0.00012207
0.865417	0.865479	0.865448	-0.000170131	-7.83255e-05	6.10352e-05
0.865448	0.865479	0.865463	-7.83255e-05	-3.24205e-05	3.05176e-05
0.865463	0.865479	0.865471	-3.24205e-05	-9.46745e-06	1.52588e-05
0.865471	0.865479	0.865475	-9.46745e-06	2.00918e-06	7.62939e-06
0.865471	0.865475	0.865473	-9.46745e-06	-3.72915e-06	3.8147e-06
0.865473	0.865475	0.865474	-3.72915e-06	-8.59985e-07	1.90735e-06

Como podemos notar, encontramos que a raiz da função $f(x) = x^3 - \cos(x)$, e portanto, a solução da equação $x^3 - \cos(x) = 0$ é, com uma precisão de 6 casas decimais, $x = 0.865474$.

Se quisermos analisar se existem outras raízes para a função utilizada, podemos fazer duas análises distintas, uma qualitativa e outra visual, comecemos pela qualitativa:

Podemos notar que a nossa função pode ser reescrita como $f(x) = g(x) - h(x)$, onde $g(x) = x^3$ e $h(x) = \cos(x)$. Sabemos que $h(x)$ é uma função periódica em torno de $x = 0$, e portanto, possui múltiplas (infinitas!) raízes. Contudo, esta é uma função limitada, com um valor máximo igual a 1 e um valor mínimo igual a -1, enquanto $g(x)$ é uma função estritamente crescente que cruza o eixo x apenas uma única vez em $x = 0$. Tendo isto em mente, podemos imaginar que a única região onde poderia haver raiz deve ser enquanto $|g(x)| < 1$, pois fora deste intervalo temos que $g(x) > h(x)$ quando $x > 0$ - e portanto $g(x) - h(x) > 0$ - e que $g(x) < h(x)$ quando $x < 0$ - e portanto $g(x) - h(x) < 0$. Recordamos então que $h(x)$ é uma função par, com um máximo neste intervalo dado por $h(x = 0) = 1$, e como $g(x = 0) = 0$, então podemos concluir que, para $x \leq 0$, $f(x) = g(x) - h(x) < 0$, de modo que nossa raiz, até agora única, se restringe ao intervalo $[0, 1]$. Por fim, dado que $h(x) = \cos(x)$ é estritamente decrescente neste intervalo, podemos concluir que $f(x)$ cruzará o eixo x apenas uma única vez, e portanto, possui uma única raiz.

Para finalizar esta discussão, podemos realizar uma análise visual desta questão, fazendo um gráfico de $g(x) = x^3$, $h(x) = \cos(x)$ e $f(x) = g(x) - h(x) = x^3 - \cos(x)$. O gráfico foi limitado entre $[-\frac{\pi}{2}, \frac{\pi}{2}]$ para que possamos observar mais de perto o que foi discutido na análise qualitativa, e para facilitar a visualização, plotamos $-h(x)$ no lugar de $h(x)$. Como podemos notar, só existe uma raiz para $f(x)$ entre 0.5 e 1, mais próximo de 1, que indica que o resultado calculado numericamente está próximo do resultado correto.

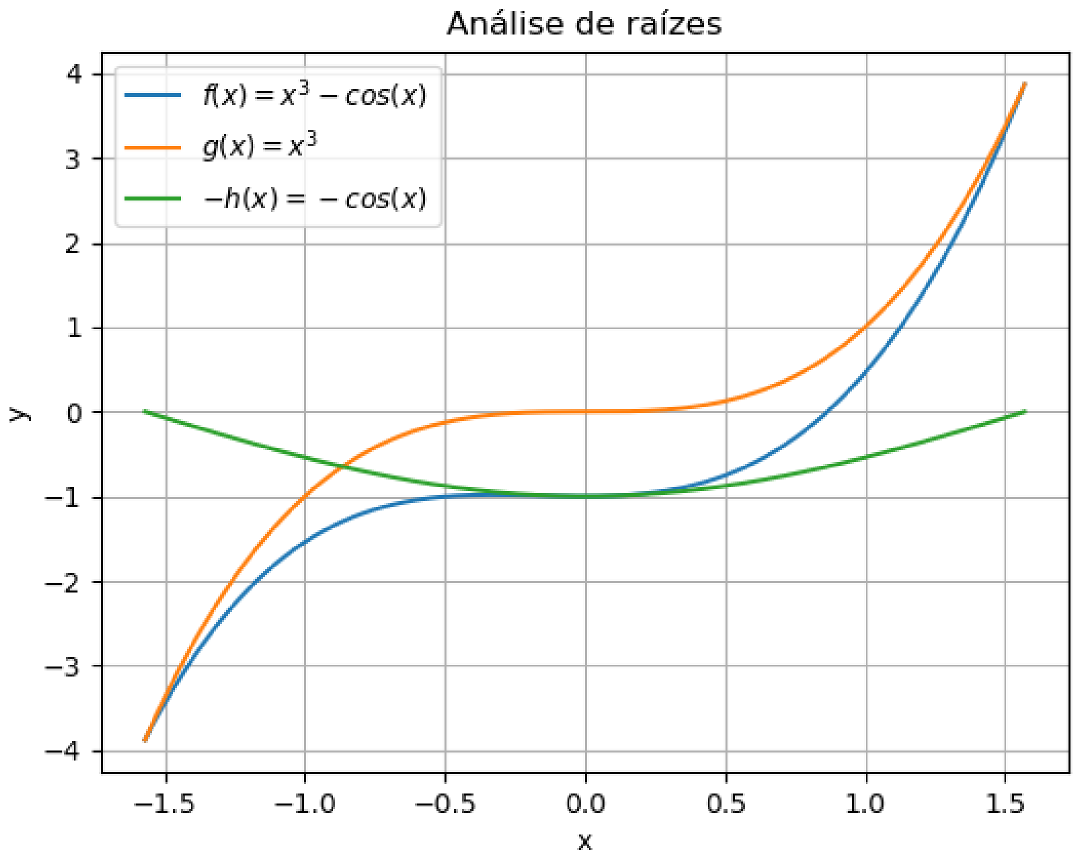

Figura 1: Análise de raízes

Item b

Para resolver a equação $x^3 - \cos(x) = 0$ pelo método de Newton-Raphson, devemos começar tomando a derivada da função $f(x) = x^3 - \cos(x)$, para assim obter a função de recursão dada por:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Não é difícil ver que:

$$f(x) = x^3 - \cos(x) \Rightarrow f'(x) = 3x^2 + \sin(x)$$

De modo que:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^3 - \cos(x_n)}{3x_n^2 + \sin(x_n)}$$

Para determinar quem deve ser o nosso x_0, recordamos que, utilizando o método da bissecção no item (a), a raiz da função se encontra por volta de $x \simeq 0.865$, de modo que precisamos apenas tomar como x_0 um número suficientemente próximo da raiz. Para isso, iniciaremos nossa integração com $x = 1$, e utilizaremos como condição de parada que o erro relativo seja menor que $\epsilon = 10^{-10}$, isto é, a integração deve parar apenas quando

$$e_n = \left|\frac{x_{n+1} - x_n}{x_n}\right| < \epsilon$$

O código pode ser encontrado logo abaixo, e abaixo dele, uma tabela que acompanha o número do passo n, o valor de x_n, além dos cálculos de $f(x_n)$ e $f'(x_n)$, e do erro relativo, e_n, de cada passo.

```
from math import cos, sin

#Precisao
epsilon = 1e-10

def G(x):
    ''' Funcao que toma um valor de x_n e calcula o valor da funcao recursiva
        G(x_n) = x_n - f(x_n)/f'(x_n)
        = x_n - ((x_n)^3 - cos(x_n))/(3(x_n)^2 + sin(x_n))
    '''

    G = x - (x**3 - cos(x))/(3*x**2 + sin(x))

    return G

x_n = 1 #Tomamos o valor inicial x_0 = 1
x_n_1 = G(x_n) #Calculamos o valor de x_{n+1} = x_1

erro = abs((x_n_1 - x_n)/x_n) #Avaliamos o erro relativo entre x_0 e x_1

#Enquanto o erro for maior que a precisao, o programa continua
while erro > epsilon:

    x_n = x_n_1 #Colocamos o valor anterior de x_{n+1} como o atual valor de
        x_n
    x_n_1 = G(x_n) #Recalculamos x_{n+1} a partir do novo x_n

    erro = abs((x_n_1 - x_n)/x_n) #Verificamos o erro relativo do passo atual
```

```
#Saimos do loop com o valor desejado, e o printamos
print("A raiz de f(x) = x^3 - cos(x), com precisao de epsilon = 1e-10, eh de %.10f" %(x_n_1))

>>> A raiz de f(x) = x^3 - cos(x), com precisao de epsilon = 1e-10, eh de 0.8654740331
```

n	x_n	$f(x_n)$	$f'(x_n)$	e_n
0	1	0.459698	3.84147	0.119667
1	0.8803328996	0.0453512	3.09591	0.01664
2	0.8656841632	0.000632313	3.00977	0.000242683
3	0.8654740760	1.2892e-07	3.00854	4.9512e-08
4	0.8654740331	5.21805e-15	3.00854	2.05247e-15

Item c

i)

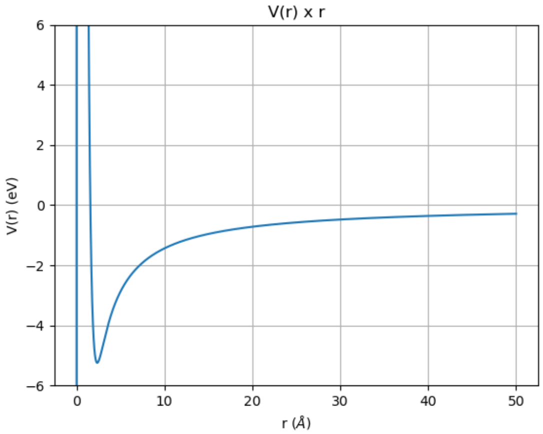

Figura 2: Gráfico do potencial V(r)

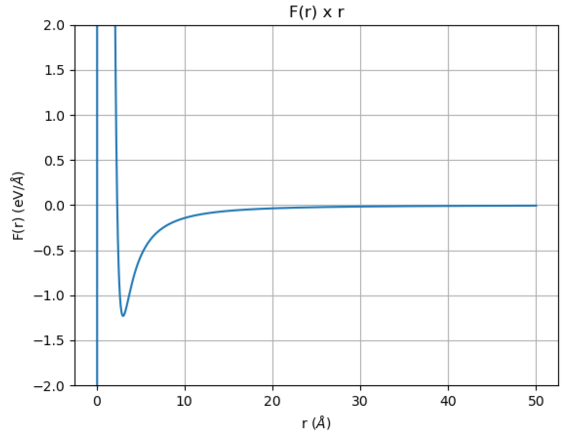

Figura 3: Gráfico da força F(r)

ii)

Para encontrar o ponto de equilíbrio $r = r_{eq}$, nós utilizamos do fato de que, neste ponto, $F(r) = 0$, e portanto, basta apenas encontrar a intersecção de $F(r)$ com o eixo r, para determinar r_{eq}. Faremos esta análise a partir do método das secantes, utilizando como passo de recursão:

$$r_{n+1} = r_n - F(r_n)\frac{r_n - r_{n-1}}{F(r_n) - F(r_{n-1})} \to r_n = r_{n-1} - F(r_{n-1})\frac{r_{n-1} - r_{n-2}}{F(r_{n-1}) - F(r_{n-2})}$$

Onde fizemos a mudança $n \to n-1$, apenas para facilitar a escrita de nosso código. A tabela ao final utiliza a fórmula de recursão para r_{n+1}, mostrada inicialmente.

Com base no gráfico obtido no item anterior, utilizaremos $r_0 = 3$ e $r_1 = 2.5$ como nossos valores iniciais. A precisão utilizada foi de $\epsilon = 10^{-10}$, e a condição de parada é de que o erro relativo $e_n < \epsilon$. O código utilizado para implementar o método, juntamente com uma tabela dos valores de n, r_{n-1}, r_n, $F(r_{n-1})$, $F(r_n)$ e o erro relativo e_n foram colocados abaixo.

```python
from math import exp

#Constantes
K = 14.4 #eV*A - e^2/(4*pi*epsilon_0)
V_0 = 1.09e3 #eV
r_0 = 0.330 #A
epsilon = 1e-10

#Funcoes uteis
def F(r):
    ''' Para um dado valor de r, calcula o valor da forca entre dois ions de
        Na e Cl '''

    f = - K/(r**2) + (V_0/r_0)*exp(-r/r_0)

    return f

def G(r_n_1,r_n_2):
    '''Dado um valor de r_{n-1} e de r_{n-2}, eh calculado o proximo valor,
       r_{n}, a partir da formula de recursao dada atraves do metodo das
       secantes '''

    g = r_n_1 - F(r_n_1)*(r_n_1 - r_n_2)/(F(r_n_1) - F(r_n_2))

    return g

r_n_2 = 3 #r_0 = r_{n-2}
r_n_1 = 2.5 #r_1 = r_{n-1}

#r_2 = r_n = r_{n-1} - F(r_{n-1})*(r_{n-1} - r_{n-2})/(F(r_{n-1}) - F(r_{n-2})
    )
r_n = G(r_n_1,r_n_2)

e_n = abs((r_n - r_n_1)/r_n_1) #Erro relativo entre r_{n-1} e r_n

#Iremos realizar a integracao enquanto o erro relativo for maior que a
#precisao desejada
while e_n > epsilon:

    #Atualizamos as variaveis do loop anterior, para seus respectvos
    #valores pro loop atual. Isto eh: r_{n-2} -> r_{n-1} e
    #r_{n-1} -> r_n
    r_n_2 = r_n_1
    r_n_1 = r_n
```

```
        #Calculamos entao o novo valor de r_n da recursao
        r_n = G(r_n_1,r_n_2)

        #Atualizamos o valor do erro relativo
        e_n = abs((r_n - r_n_1)/r_n_1)

#Printamos o resultado obtido
print("O ponto de equilibrio r = r_eq, tal que F(r) = 0 e V(r) eh minimo")
print("eh dado por r_eq = %.10f A" %r_n)

>>> O ponto de equilibrio r = r_eq, tal que F(r) = 0 e V(r) eh minimo
>>> eh dado por r_eq = 2.3605384842 A
```

n	r_{n-1}	r_n	$F(r_{n-1})$	$F(r_n)$	e_n
0		3		-1.2278	
1	3	2.5	-1.2277961129	-0.610431	0.1666666667
2	2.5	2.0056155862	-0.6104311905	3.99603	0.1977537655
3	2.0056155862	2.4344860743	3.9960347312	-0.364188	0.2138348400
4	2.4344860743	2.3986645909	-0.3641884086	-0.20047	0.0147141870
5	2.3986645909	2.3548020308	-0.200469696	0.0327095	0.0182862416
6	2.3548020308	2.3609549101	0.0327095277	-0.00234749	0.0026129073
7	2.3609549101	2.3605429006	-0.0023474903	-2.49156e-05	0.0001745097
8	2.3605429006	2.3605384807	-2.4915617141e-05	1.92695e-08	1.8723942403e-06
9	2.3605384807	2.3605384842	1.9269454921e-08	-1.58984e-13	1.4469720098e-09
10	2.3605384842	2.3605384842	-1.5898393713e-13	4.44089e-16	1.1852219486e-14

Podemos ver portanto que o método das secantes convergiu rapidamente para o valor, e com ele obtivemos que a distância de ligação r_{eq} de equilíbrio para a molécula de NaCl é de $r_{eq} \simeq 2.3605384842$ Å