# Cálculo Numérico - MS211F - 1S/2024

Isadora Santos de Souza 257032
Pedro Henrique Martins Belo 267809
Sarah Pereira Teixeira Silva 258968

**[1]**

Escreva em metalinguagem o algoritmo para o método de Newton especificando o critério de parada. Seu algoritmo deve ser claro e permitir imediata tradução para uma linguagem de programação. Isto significa que não pode conter passos como “derive f” ou “compute o ponto de mínimo de…” pois estas não são instruções elementares em linguagens de programação.

**Valores de entrada**:
Crie uma variável para receber a função f(x), uma para o ponto inicial $x_0$, uma para a tolerância $\epsilon$ (para o erro) e defina o número máximo de iterações $N_{max}$ que o usuário desejar:

f(x) = (‘Digite a função’)
$x_0$ = (‘Digite o ponto inicial desejado’)
$\epsilon$ = (‘Digite o erro’)
$N_{max}$ = (‘Digite o número máximo de iterações’)

**Iniciando**:
Defina um contador para as iterações com valor inicial k = 0 e defina como valor inicial x = $x_0$:

int = 0 (‘Contador com valor inicial’)
x = $x_0$ (‘O valor inicial passa a ser $x_0$’)

**Implementando o método:**
Agora calcule a derivada f’(x), a função f(x) e a aproximação da raiz pelo Método de Newton com a equação $x_n = x_i - f(x_i)/f'(x_i)$, para n = 1, 2, …, e i = 0, 1, …:

$f'(x) - (f(x+10^{(-u)}) - f(x))/10^{(-u)}$
(‘Cálculo da derivada tomando um delta tão pequeno quanto, de maneira que a máquina ainda consiga ler e u seja um valor próximo da precisão da máquina de tal maneira que ela leia como um valor diferente de zero’)
$f(x_0)$ = (‘Cálculo de f(x) no ponto $x_0$’)
A = $x_i - (f(x_i)/f'(x_i))$ (‘Fórmula de Newton para cálculo do valor aproximado da raiz desejada’)

**Critério de Parada**:

Defina os critérios de parada, sendo x* a raiz real da função:

1° Critério: $|(A - x^*)| < \epsilon$
2° Critério: $|f(x_i)| < \epsilon$
3° Critério: int = $N_{max}$
4° f'(x) = 0 ('Derivada igual a zero)

**Derivada igual a zero:**
Defina um novo ponto caso a derivada no ponto seja igual a zero.

Se f'(x) = 0:
$x_i = x_k$ ('Onde $x_i$ # $x_k$')

**Repetição:**
Repita o processo até obter o valor tão próximo quanto se queira da raiz de f(x). Seja x* a raiz real da função:

Enquanto $|A - x^*| > \epsilon$ ou $|f(x_i)| > \epsilon$ int = $N_{max}$:
$f'(x) - (f(x+10^{(-u)}) - f(x))/10^{(-u)}$
$f(x_0)$ = ('Cálculo de f(x) no ponto $x_0$')
$A = x_i - (f(x_i)/f'(x_i))$
int = int + 1

**Saída**: Defina a saída do seu programa:
Exiba ("O valor aproximado para a raiz da função f(x) é A")

Fim do programa!

**Pseudocódigo:**

```
f(x) = input()
x0 = input()
ϵ = input()
Nmax = input()

int = 0
x = x0

u = input()
f'(x) = (f(x+10^(-u)) - f(x))/(10^(-u)
f(x) = f(x0)
A = xi - (f(xi)/f'(xi))
```

```
Se f'(x) = 0:
     xi = xk ('Onde xi # xk')
Se não:
     Enquanto |A - x*|  > ϵ ou |f(xi)| > ϵ ou int = Nmax:
          f'(x) = (f(x+10^(-u)) - f(x))/(10^(-u)
          f(x0) = ('Cálculo de f(x) no ponto x0')
          A = xi - (f(xi)/f'(xi))
          int = int + 1
Exiba ("O valor aproximado para a raiz da função f(x) é A")
```

Fim do programa!

Resumo da implementação feita no exercício:
Começamos definindo uma função f(x) e depois definindo sua derivada f'(x), então utilizamos a fórmula do método de newton ($A = x_i - (f(x_i)/f'(x_i))$ ) para conseguir fazer uma aproximação da raiz da fórmula. E fazendo isso a gente consegue achar o valor aproximado para a função f(x).

**[2]**

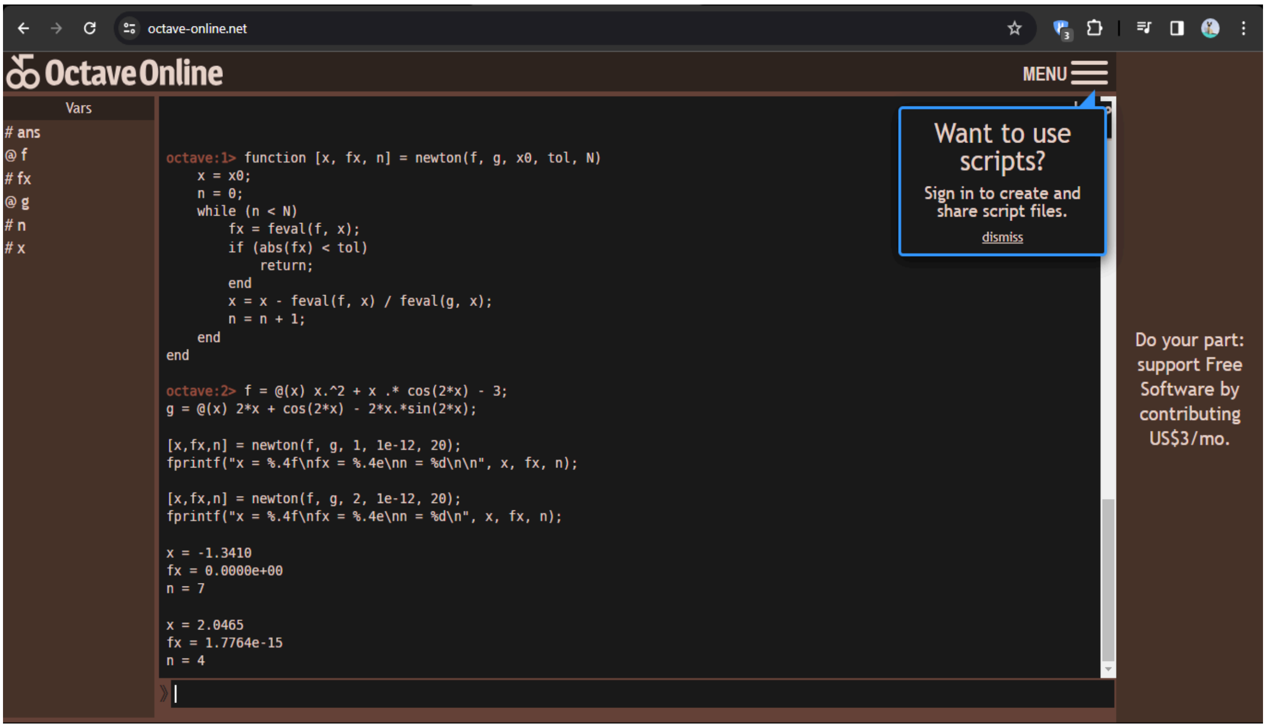
Implemente em Octave o método de Newton para uma função real de uma variável. Sua função deve ter o seguinte protótipo: function [x, fx, n] = newton(f, g, x0, tol, N) onde f é uma função definida inline no Octave, g é uma função definida inline no Octave que computa a derivada da função f, x0 é a aproximação inicial para um zero de f, tol é a tolerância para o critério de parada e N é o limite para a quantidade de passos. Sua implementação não deve retornar a aproximação para a raiz, x, o valor da função em x, fx, e a quantidade de iterações realizadas, n.
Sua implementação deve ser capaz de reproduzir a seguinte saída no Octave (alguma variação pode acontecer depende de escolhas na implementação).

```
>> f = @(x) x^2 + x .* cos(2*x) - 3;
>> g = @(x) 2*x + cos(2*x) - 2*x.*sin(2*x);
>> [x,fx,n] = newton(f, g, 1, 1e-12, 20)
x = -1.3410
fx = 0
n = 7
>>
>> [x,fx,n] = newton(f, g, 2, 1e-12, 20)
x = 2.0465
fx = 1.7764e-15
n = 4
>>
```

O código feito em Octave foi:

```
function [x, fx, n] = newton(f, g, x0, tol, N)
   x = x0;
   n = 0;
   while (n < N)
      fx = feval(f, x);
      if (abs(fx) < tol)
         return;
      end
      x = x - feval(f, x) / feval(g, x);
      n = n + 1;
   end
end
```

Print do código e sua execução, dando as saídas pedidas:




Resumo da implementação utilizada:
Definimos uma função f(x) e então definimos a sua derivada g(x). Depois, implementamos o método de newton, obtendo como uma raiz aproximada x = 2.0465, assim como foi pedido.

**[3]**

Suponha a situação em que o usuário não tem acesso ou não sabe computar a derivada de f. Crie uma nova implementação para o método de Newton onde a avaliação da derivada, ao invés de usar a função fornecida pelo usuário na chamada da rotina, é aproximada pela fórmula de diferenças centradas, computada internamente na rotina. Sua função deve ter o seguinte protótipo: function [x, fx, n] =

newtondc(f, x0, tol, N) onde todos os parâmetros têm os mesmos significados de antes.

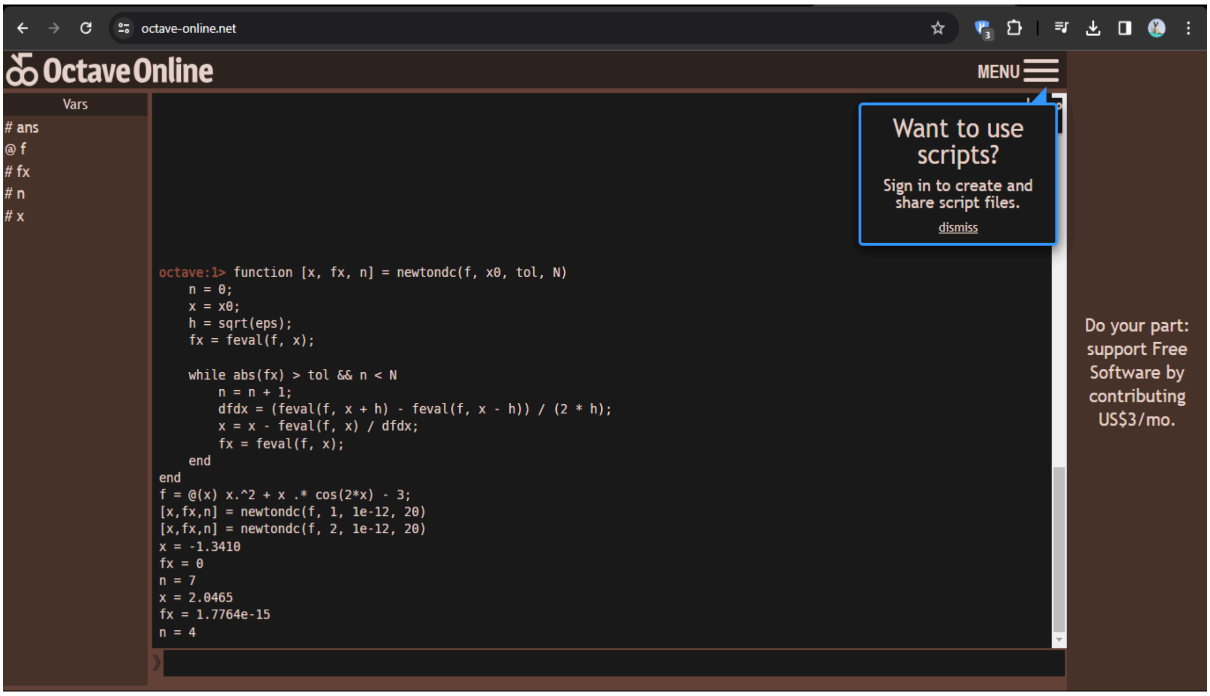
O código feito em Octave, com diferenças centradas, foi:

```
function [x, fx, n] = newtondc(f, x0, tol, N)
    n = 0;
    x = x0;
    h = sqrt(eps);
    fx = feval(f, x);

    while abs(fx) > tol && n < N
```

```
        n = n + 1;
        dfdx = (feval(f, x + h) - feval(f, x - h)) / (2 * h);
        x = x - feval(f, x) / dfdx;
        fx = feval(f, x);
    end
end
```

Print do código e sua execução, dando as saídas pedidas:




Resumo da implementação feita:
Neste exercício fizemos a aproximação das raízes da função através da diferença centrada já que supomos que o usuário do programa não sabe calcular a derivada da função, como esse método achamos a mesma raiz encontrada com o método de newton.

**[4]**

Apresente exemplos e testes que embasam a qualidade e eficácia das implementações propostas. Escolha-os bem, para que sejam representativos e permitam conclusões. Compare o desempenho das duas rotinas, do ponto de vista do número de iterações necessárias, do número de avaliações de função f e de sua derivada (no caso da primeira rotina). Os resultados das duas rotinas são comparáveis?

No exemplo criado, foi definida uma função f(x) = x*(sin(x))²:

Método da bissecção:

```
octave:6> f = @(x) x*(sin(x))^2
[x,fx,n] = newtondc(f, 1, 1e-12, 100)
f =

@(x) x * (sin (x)) ^ 2

x = 7.0358e-05
fx = 3.4829e-13
n = 23
```

Método de Newton:

```
octave:8> f = @(x) x*(sin(x))^2
g = @(x) 2*x*sin(x)*cos(x) + (sin(x))^2
[x,fx,n] = newton(f, g, 0.5, 1e-12, 100)
f =

@(x) x * (sin (x)) ^ 2

g =

@(x) 2 * x * sin (x) * cos (x) + (sin (x)) ^ 2

x = 9.5204e-05
fx = 8.6291e-13
n = 21
```

Comparando os dois métodos, é possível perceber que, apesar do número de iterações do Método de Newton ter sido menor (n = 21, em comparação com n = 23 com o método da bissecção), o valor da raíz com o primeiro método foi bem menor, ou seja, apesar da otimização no número de iterações, houve um aumento do erro com o segundo método.