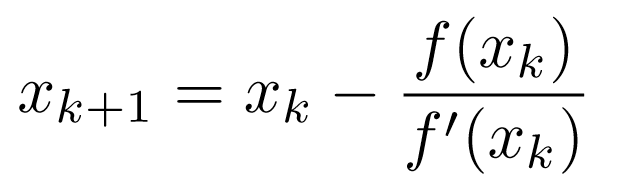
**Introdução**

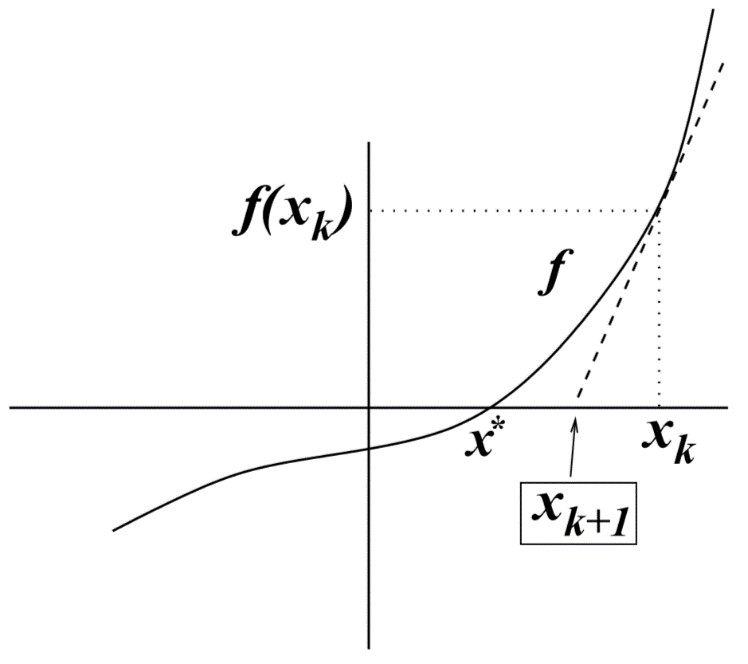
Asano & Colli dizem que o Método de Newton consiste em fazer a iteração:



A partir de uma condição inicial bem escolhida x0, e assim obter aproximações sucessivas de alguma raiz x∗ de f.

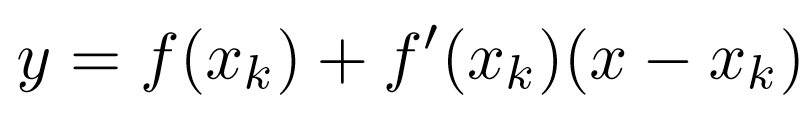
A maneira de achar x1 em função de x0, e igualmente depois de achar xk+1 em função de xk, tem uma forte inspiração geométrica: olhamos para a reta tangente ao gráfico de f no ponto (xk,f(xk)) e definimos xk+1 como sendo o ponto de encontro dessa reta com a abscissa.

***Figura 1****: reta tangente*

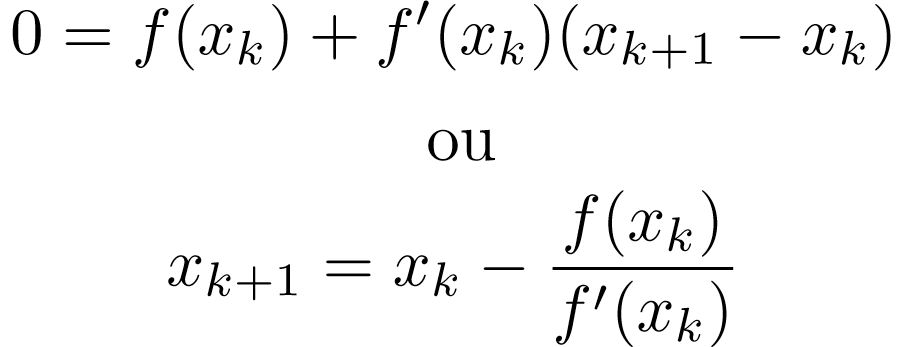
**

***Fonte****: Cálculo Numérico — Fundamentos e Aplicações*

Vejamos como a fórmula acima se relaciona com esta ideia geométrica. Para isso, notamos que a inclinação da reta tangente ao gráfico de f no ponto (xk,f(xk)) é dada pela derivada f′(xk). A única reta com inclinação f′(xk) que passa por (xk,f(xk)) é dada por:



O ponto xk+1 é definido como o valor de x para o qual y = 0, isto é:



**Desenvolvimento teórico**

Foi proposto a criação de um algoritmo capaz de aplicar o método de Newton-Raphson para resolver ex + x−3 = 0 com erro = 0,0001. A derivada f’(xk) será calculada de forma numérica como no trabalho anterior.

Para isso foi elaborado o seguinte código:

Função para calcular a derivada f’(xk)

float derivada(float X0){ //função que calcula a derivada f'(x) e^x + x - 3

float X, amplitude = 0.00001, fx, fx0, resultado;

X = amplitude+X0;

fx = pow(M\_E,X) + X - 3;

fx0 = pow(M\_E,X0) + X0 - 3;

resultado = (fx - fx0)/amplitude;

return(resultado);

}

O código do trabalho anterior foi reutilizado em forma de função, onde se entra com o valor xk e sai com o valor da derivada f'(x) e^x + x – 3, a amplitude escolhida para a realização dos cálculos foi 0.00001.

**Função para calcular a função f(xk)**

float funcao(float X){ //função que calcula a função f(x) e^x + x - 3

float resultado;

resultado = pow(M\_E,X) + X - 3;

return(resultado);

}

Essa função calcula a função e^x+x-3, ‘pow’ é uma função da biblioteca math.h que realiza elevação, M\_E é uma constante para ‘e’, portanto pow(M\_E,X) + X – 3 = e^x+x-3.

**Função para modulo**

float modulo(float X){ //função que faz modulo de X

if (X < 0) X = X \* -1;

return (X);

}

Eu entro com um valor e caso ele seja negativo, é multiplicado por -1.

**Entrada de dados**

printf("Digite o chute inicial 'Xk': ");

scanf("%f", &Xk);

printf("Digite o erro(criterio de parada): ");

scanf("%f", &erro);

O usuário coloca o chute inicial Xk e o critério de parada.

**Aplicando a formula**

do

{

quant ++;

if (quant != 1) X\_anterior = X\_atual;

else X\_anterior = Xk;

X\_atual = X\_anterior -(funcao(X\_anterior)/derivada(X\_anterior));

result\_erro = X\_atual - X\_anterior;

printf("X%d = %f, erro = %f\n", quant, X\_atual, modulo(result\_erro));

}

while (modulo(result\_erro) > erro);

Foi usado um DoWhile onde o parâmetro para sair é o erro calculado ser menor que o parâmetro de parada. A cada repetição um contado recebe +1, foram usadas duas variáveis para o cálculo, X\_anterior e X\_atual, se tiver no primeiro ciclo X\_anterior recebe o chute inicial, depois disso X\_anterior começa a receber X\_atual.

Depois que é calculado um novo X atual e é verificado o erro, caso a condição não seja satisfeita o processo se repete.

**Resultados**

Testando com e^x + x−3 = 0 com erro = 0,0001. Xk0 = Chute inicial.

|  |  |  |
| --- | --- | --- |
| Xk0 | 1 | Erro |
| Xk1 | 0,807188 | 0,192812 |
| Xk2 | 0,792160 | 0,015028 |
| Xk3 | 0,792060 | 0,000100 |
| Xk4 | 0,792060 | 0,000000 |

|  |  |  |
| --- | --- | --- |
| Xk0 | 2 | Erro |
| Xk1 | 1,238703 | 0,761297 |
| Xk2 | 0,859682 | 0,379021 |
| Xk3 | 0,793734 | 0,065948 |
| Xk4 | 0,792063 | 0,001671 |
| Xk5 | 0,792060 | 0,000003 |

|  |  |  |
| --- | --- | --- |
| Xk0 | 3 | Erro |
| Xk1 | 2,051297 | 0,948703 |
| Xk2 | 1,272925 | 0,778371 |
| Xk3 | 0,870052 | 0,402874 |
| Xk4 | 0,794249 | 0,075803 |
| Xk5 | 0,792065 | 0,002184 |
| Xk6 | 0,792060 | 0,000005 |

|  |  |  |
| --- | --- | --- |
| Xk0 | -1 | Erro |
| Xk1 | 1,626589 | 2,626589 |
| Xk2 | 1,018237 | 0,608352 |
| Xk3 | 0,809768 | 0,208469 |
| Xk4 | 0,792192 | 0,017577 |
| Xk5 | 0,792060 | 0,000132 |
| Xk6 | 0,792060 | 0,000000 |

Pude observar que se o chute inicial for maior que o resultado final, proporcionalmente maior vai ser a quantidade de iterações, quando o chute inicial é 1 acontecem 4 iterações, já quando o chute inicial foi 3 foram 6 iterações. Outros valores:

|  |  |
| --- | --- |
| Chute inicial | Quantidade de iteração |
| 10 | 13 |
| 20 | 22 |
| 30 | 32 |
| 40 | 43 |
| 50 | 55 |
| 88 | 89 |

O limite para o chute inicial foi 88, sendo 89 o número de iterações, porem independente, do chute o valor final apresentado sempre foi 0,792060.

Mudando o critério de parada para 0.00001 os resultados foram o mesmo.

# **Bibliografia**

Asano, C. H., & Colli, E. (2009). *Cálculo Numérico — Fundamentos e Aplicações.* Departamento de Matemática Aplicada – IME-USP.

**Apêndice**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

float derivada(float X0){ //função que calcula a derivada F'(x) e^x + 1 de forma numérica

float X, amplitude = 0.00001, fx, fx0, resultado;

X = amplitude+X0;

fx = pow(M\_E,X) + X - 3;

fx0 = pow(M\_E,X0) + X0 - 3;

resultado = (fx - fx0)/amplitude;

return(resultado);

}

float funcao(float X){ //função que calcula a função F(x) e^x + x - 3

float resultado;

resultado = pow(M\_E,X) + X - 3;

return(resultado);

}

float modulo(float X){ //função que faz modulo de X

if (X < 0) X = X \* -1;

return (X);

}

int main()

{

int quant=0;

float Xk, X\_atual, X\_anterior, erro, result\_erro;

printf("Digite o chute inicial 'Xk': ");

scanf("%f", &Xk);

printf("Digite o erro(criterio de parada): ");

scanf("%f", &erro);

printf("\n");

do

{

quant ++;

if (quant != 1) X\_anterior = X\_atual;

else X\_anterior = Xk;

X\_atual = X\_anterior -(funcao(X\_anterior)/derivada(X\_anterior));

result\_erro = X\_atual - X\_anterior;

printf("X%d = %f, erro = %f\n", quant, X\_atual, modulo(result\_erro));

}

while (modulo(result\_erro) > erro);

printf("\nNumero total de iteracoes = %d\nresultado final = %f\nerro = %f", quant, X\_atual, modulo(result\_erro));

return 0;

}