Comentários sobre convergência de um método e condições de parada.

# Ingredientes de um problema (computacional)

Em geral, temos os seguintes ingredientes:

1) Problema matemático (e preferencialmente um teorema sobre existência de soluções). Por exemplo: achar raiz de $f(x)$; minimizar $f(x)$; etc.

2) Método numérico: Algoritmo que gera uma sequência $x_1, x_2, \ldots, x_n$ de aproximações para a solução exata do problema.

3) Teorema de convergência sobre o método numérico:
(Sob certas condições sobre os dados iniciais e parâmetros, o método converge.)

Exemplo Teo. de convergência para o método da secante (referência Atkinson, Intro. to Num. Analysis)

Suponha que $f(\alpha) = 0$ e $f'(\alpha) \neq 0$, onde $f(x)$, $f'(x)$ e $f''(x)$ são contínuas em um intervalo contendo $\alpha$. Se os pontos iniciais $x_0$ e $x_1$ estão suficientemente próximos de $\alpha$, então a sequência de aproximações (do método da secante) converge para $\alpha$.

# Implementação em um programa

Suponha que implementamos um método em Python.

Idealmente, se sabemos que as entradas do programa a seguir fazem as condições de convergência do método (dada por algum teorema), a condição de parada das iterações do método poderia ser

Repita
⋮

Até que a sequência de aproximações convirja:

$$|x_n - x_{n-1}| < TOL$$

Todavia, na prática, para uma certa entrada para o programa, pode ocorrer:

1) A iteração não converge

2) A iteração converge, mas demora muito

Logo, na prática é útil introduzir um número máximo de iterações No para o nosso programa. por segurança. Veja o exemplo abaixo:

# Pseudo código do método da secante

## Secant

To find a solution to $f(x) = 0$ given initial approximations $p_0$ and $p_1$:

INPUT initial approximations $p_0, p_1$; tolerance $TOL$; maximum number of iterations $N_0$.

OUTPUT approximate solution $p$ or message of failure.

*Step 1* Set $i = 2$;
$q_0 = f(p_0)$;
$q_1 = f(p_1)$.

*Step 2* While $i \leq N_0$ do Steps 3–6.

*Step 3* Set $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$. (*Compute* $p_i$.)

*Step 4* If $|p - p_1| < TOL$ then
OUTPUT $(p)$; (*The procedure was successful.*)
STOP.

*Step 5* Set $i = i + 1$.

*Step 6* Set $p_0 = p_1$; (*Update* $p_0, q_0, p_1, q_1$.)
$q_0 = q_1$;
$p_1 = p$;
$q_1 = f(p)$.

*Step 7* OUTPUT ('The method failed after $N_0$ iterations, $N_0 =$', $N_0$);
(*The procedure was unsuccessful.*)
STOP.

Fonte: Burden e Faires

## Computational example (Fonti: Wikipedia)

Below, the secant method is implemented in the Python programming language.

It is then applied to find a root of the function $f(x) = x^2 - 612$ with initial points $x_0 = 10$ and $x_1 = 30$

```python
def secant_method(f, x0: int, x1: int, iterations: int) -> float:
    """Return the root calculated using the secant method."""
    for i in range(iterations):
        x2 = x1 - f(x1) * (x1 - x0) / float(f(x1) - f(x0))
        x0, x1 = x1, x2
        # Apply a stopping criterion here (see below)
    return x2

def f_example(x):
    return x ** 2 - 612

root = secant_method(f_example, 10, 30, 5)

print(f"Root: {root}")  # Root: 24.738633748750722
```

It is very important to have a good stopping criterion above, otherwise, due to limited numerical precision of floating point numbers, the algorithm can return inaccurate results if running for too many iterations. For example, the loop above can stop when one of these is reached first: `abs(x0 - x1) < tol`, or `abs(x0/x1-1) < tol`, or `abs(f(x1)) < tol`. [4]