Documentação:

CalcNumCTEC

Lucas Henrique Correia da Rocha
lucash.rocha@hotmail.com

Olá! Seja bem-vindo à documentação da biblioteca CalcNumCTEC. Esta biblioteca foi implementada inicialmente como compilação dos métodos estudados na disciplina de Cálculo Numérico a partir da atividade de monitoria no Centro de Tecnologia (CTEC) da Universidade Federal de Alagoas (UFAL). Meu nome é Lucas, fui monitor da disciplina no semestre de 2020.2 e, como resultado dos trabalhos realizados durante o exercício da função de monitor, auxiliando os alunos com a implementação dos métodos em linguagem Julia, resolvi juntar todas elas numa biblioteca.

Entretanto, durante a realização do trabalho, foram surgindo novas ideias, que foram sendo desenvolvidas e agregadas ao trabalho inicial, de forma a expandir a aplicabilidade da biblioteca, implementando novas funções e até mesmo tipos de dados específicos, a fim de facilitar o trabalho acadêmico.

Portanto, espero que esta biblioteca seja útil para você. E não deixe de dar feedback, em especial de melhorias para que possamos evoluir suas funcionalidades. Faça bom proveito! 😀

Introdução

1. INCLUSÃO DA BIBLIOTECA

Com o objetivo de simplificar o desenvolvimento e a utilização da biblioteca, o código está escrito num script com nome da biblioteca e a primeira coisa a ser feita, antes mesmo de usá-la, é colocar uma cópia do arquivo em algum local de fácil acesso. Por conveniência, em especial se o programa que você está escrevendo será compartilhado com outros usuários, é recomendado que ele seja guardado na mesma pasta do arquivo que o está chamando ou numa subpasta dela.

Depois disso, é preciso incluir o *script* no código em que estamos escrevendo! Em Julia, isso é feito através da função *include*, passando o diretório do arquivo. Veja o exemplo abaixo:

In [8]:
```
include("C:\\Users\\lucas\\Documents\\GitHub\\CalcNumCTEC\\CalcNumCTEC\\code\\CalcNu
```

Out[8]:
```
"CalcNumCTEC incluída com sucesso!"
```

Um jeito de garantir que a biblioteca foi incluída com êxito, conforme demonstrado no exemplo anterior, é executar o programa de modo que a chamada seja a última linha. Desta maneira, será exibida uma saída no terminal com a mensagem **"CalcNumCTEC incluída com sucesso!"**. Pode-se ainda testar executando alguma das funções implementadas.

2. NOMENCLATURA DAS FUNÇÕES

A funções implementadas estão estrategicamente nomeadas de maneira lógica, facilitando a utilização com nomes triviais e de fácil memorização. O padrão utilizado para nomenclatura das funções é da forma:

ASSUNTO_MÉTODO

O assunto está associado ao objeto principal da função, ou seja, ao resultado do que a função de fato implementa. São eles:

Assunto	*ASSUNTO*
Zeros de funções	Zeros
Sistemas de equações lineares	SEL
Sistemas de equações não lineares	SENL
Interpolação	Interpolacao
Ajuste	Ajuste
Integral	Integral
Integral dupla	IntegralDupla
Integral tripla	IntegralTripla

O método, por sua vez, como o próprio nome sugere, indica o método numérico utilizado para a determinação do resultado requerido. Veremos então, para cada função o método implementado e como essa nomenclatura é efetivamente utilizada.

3. ZEROS DE FUNÇÕES

A primeira classe de assuntos que veremos é a que determina raízes de funções quaisquer. Aqui, temos os três métodos clássicos para a determinação de zeros de funções:

- Método da Bisseção $\rightarrow$ *MÉTODO* = Bissecao

Classificado como um método intervalar, o Método da Bisseção consiste em estreitar um intervalo inicial a partir do ponto médio do mesmo até que o erro, calculado como $|f(x_k)|$, onde f é a função de análise, seja menor do que a tolerância.

Os parâmetros obrigatórios para sua utilização são: a função a ser analisada e os extremos a e b do intervalo. Para garantir a convergência do método, este só permitirá a execução da função se os valores de $f(a)$ e $f(b)$ possuírem valores simétricos, de maneira que, pelo Teorema do Valor Intermediário (TVI), verificamos a existência da raiz no intervalo dado. Vejamos um exemplo de utilização:

In [9]:

```julia
f(x) = x^3 - cos(x)

println("f(0) = $(f(0))")
println("f(1) = $(f(1))")
```

```
f(0) = -1.0
f(1) = 0.45969769413186023
```

Daí, podemos concluir que \exist$x \in [0, 1] \mid f(x) = 0$. Perceba ainda que, não é possível isolar x na equação $x^3 - cos(x) = 0$ de modo a obter este valor analiticamente, fazendo-nos ter de recorrer a métodos numéricos. Logo, podemos usar a função $Zeros_Bissecao()$ para calcular este valor de x.

In [10]:

```julia
xᵣ = Zeros_Bissecao(f, 0, 1)
println("Raiz de f(x) = $xᵣ")
```

```
Raiz de f(x) = 0.865474033101691
```

Além destes parâmetros obrigatórios, temos parâmetros adicionais referentes à convergência da função, são eles:

> *tol*: Define a tolerância permitida para o erro da aproximação (valor padrão: 10^{-12}) </br> *klim*: Define a quantidade máxima de iterações para o caso do método não convergir (valor padrão: 10^6)

In [11]:

```julia
xᵣ = Zeros_Bissecao(f, 0, 1, tol=0.01, klim=100)
println("Raiz de f(x) = $xᵣ")
```

```
Raiz de f(x) = 0.8671875
```

- Método das Cordas $\rightarrow$ *MÉTODO* = Cordas

Outro método, similar ao anterior, que foi implementado é o Método das Cordas. A única diferença para o anterior consiste na função de recorrência, calculando o valor de x para a próxima iteração com os valores da iteração anterior. Deste modo, a chamada da função é idêntica e é com os mesmos atributos, alterando-se apenas o nome da função.

In [12]:

```julia
xᵣ = Zeros_Cordas(f, 0, 1)
xᵣ₂ = Zeros_Cordas(f, 0, 1, tol=0.01, klim=100)

println("Raiz de f(x) = $xᵣ")
println("Raiz de f(x) (Baixa tolerância) = $xᵣ₂")
```

```
Raiz de f(x) = 0.865474033101691
Raiz de f(x) (Baixa tolerância) = 0.8671875
```

- Método de Newton-Raphson $\rightarrow$ *MÉTODO* = NR

Saindo agora do escopo dos métodos intervalares, o Método de Newton-Raphson difere dos anteriores pelo fato de que o chute inicial, ao invés de iniciar com um intervalo, é dado por um valor para x e o próximo valor é calculado a partir da interseção da reta tangente com o eixo x.

Para a implementação do método, a fim de evitar a determinação da derivada analiticamente, tomou-se a decisão de estimar este valor a partir de uma reta secante pelos pontos $(x_k, f(x_k))$ e $(x_k + tol, f(x_k + tol))$, onde *tol* é a tolerância da função.

De forma semelhante aos métodos anteriores, além de passar na chamada a função a ser analisada e a

In [13]:

```
# g(x) = x^4 - sin(x)
g(x) = x^x
Derivada_Secante(g, 2, 2)
```

Out[13]:

```
13.467671422517917
```