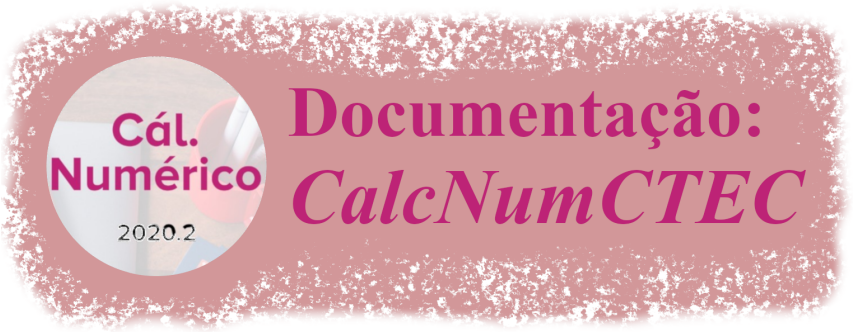

Lucas Henrique Correia da Rocha

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 [61]:
```
include("C:\\Users\\lucas\\Documents\\GitHub\\CalcNumCTEC\\CalcNumCTEC\\code\\CalcNumCTEC.jl")
```

Out[61]:
```
"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
Derivadas	MDFCentradas

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 → *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 [62]:
```
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 $\exists x \in [0,1] \mid f(x) = 0$. 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 [63]:
```
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 [64]:
```
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 [65]:
```
xᵣ = Zeros_Cordas(f, 0, 1)
xᵣ₂ = Zeros_Cordas(f, 0, 1, tol=0.01, klim=100)

println("Raiz de f(x) (Método das Cordas - tolerância padrão) = $xᵣ")
println("Raiz de f(x) (Método das Cordas - baixa tolerância). = $xᵣ₂")
```

```
Raiz de f(x) (Método das Cordas - tolerância padrão) = 0.865474033101691
Raiz de f(x) (Método das Cordas - 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 o chute inicial, desta vez caracterizado por um único valor de x, é possível também determinar a tolerância e a quantidade limite de iterações do método.

A função utilizada para o cálculo da derivada, porém, será descrita mais adiante.

In [66]:
```
xᵣ = Zeros_NR(f, 1)
xᵣ₂ = Zeros_NR(f, 1, tol=0.1, klim=10)

println("Raiz de f(x) (Método NR - tolerância padrão) = $xᵣ")
println("Raiz de f(x) (Método NR - baixa tolerância). = $xᵣ₂")
```

```
Raiz de f(x) (Método NR - tolerância padrão) = 0.8654740331016162
Raiz de f(x) (Método NR - baixa tolerância). = 0.8803328995715387
```

4. SISTEMAS DE EQUAÇÕES LINEARES

O próximo conjunto de funções está relacionado à resolução de sistemas lineares abrangendo métodos diretos, iterativos e até mesmo a verificação dos critérios de convergência.

- Eliminação de Gauss $\rightarrow$ *MÉTODO* = EliminGauss

O primeiro método para solução de sistemas de equações lineares trata-se de um método direto, denominado Método da Eliminação de Gauss, que consiste em realizar operações lineares nas linhas da matriz e do vetor que definem o sistema a fim de escalonar a matriz, ou seja, fazer com que ela tome a forma de uma matriz triangular superior, com os elementos abaixo da diagonal principal nulos.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

Com a matrix devidamente escalonada, o resultado pode ser calculado da última linha para a primeira utilizando-se os valores obtidos no passo anterior.

A função pode então ser chamada conforme o código abaixo. Veja ainda que podemos ainda validar o exemplo com o valor obtido através do operador nativo de Julia para o cálculo de sistemas de equações lineares com $\approx$ (\approx) :

In [67]:
```
A = randn(5,5)
b = randn(5)

x = SEL_ElimGauss(A, b)

println("Vetor solução:")

using DelimitedFiles
writedlm(stdout, x)

println("\n", x ≈ A\b)
```

```
Vetor solução:
0.21268382261445784
-1.4770318082746958
0.10043993657177547
2.585401811995677
-0.18573291019424595

true
```

- CRITÉRIOS DE CONVERGÊNCIA $\rightarrow$ *MÉTODO* = CritConverg

Usados para determinar o grau de convergência do sistema para os métodos iterativos a partir das características da matriz de coeficientes dos sistemas. Na verdade, o que estes critéios medem realmente é o grau de dominância dos termos Os métodos implementados são três:

a) Critério das linhas (β_i)

Calcula a soma dos elementos da linha dividido pelo elemento da diagonal principal na linha correspondente, compondo então um vetor com os valores β_i para cada linha i e retornando o maior deles.

$$\beta_i = \frac{\sum_{i=1}^{n} a_{ij}}{a_{ii}}$$

b) Critério das colunas (β_i)

Semelhantemente ao anterior, calcula a soma dos elementos da coluna dividido pelo elemento da diagonal principal na linha correspondente, armazenando então um vetor com os valores β_j para cada coluna j e retornando o maior deles.

$$\beta_j = \frac{\sum_{j=1}^{n} a_{ij}}{a_{jj}}$$

c) Critério de Sassenfield

Este, porém, é parecido com o primeiro, a primeira linha é calculada de maneira semelhante ao critério das linhas. Entretanto, na segunda linha em diante, os valores de β obtidos nas linhas anteriores são utilizados como multiplicadores para os elementos cujo índice da coluna é correspondente, ou seja:

$$\beta_1 = \frac{\sum_{i=1}^{n} a_{ij}}{a_{11}}$$
$$\beta_2 = \beta_1 a_{21} + \frac{\sum_{i=2}^{n} a_{ij}}{a_{11}}$$
$$\beta_3 = \beta_1 a_{31} + \beta_2 a_{32} + \frac{\sum_{i=3}^{n} a_{ij}}{a_{33}}$$
$$\dots$$

Para quaisquer dos três métodos, quanto maior for o valor retornado pela função, mais difícil será a convergência dos métodos iterativos, ou seja, o método deverá convergir com erros maiores e para um número maior de iterações. Todavia, é suficiente para garantir a convergência que ao menos um destes valores estejam entre 0 e 1.

Assim, foi implementada apenas uma função para responder aos três critérios, de maneira que a escolha é feita através do parâmetro *modo*:

- `'l'` ou `'L'` para o critério das linhas
- `'c'` ou `'C'` para o critério das colunas</code>
- `'s'` ou `'S'` para o critério de Sassenfield

In [72]:
```
println("\nConvergência de A pelo critério das linhas....: ", SEL_CritConverg(A, 'l'))
println("Convergência de A pelo critério das colunas...: ", SEL_CritConverg(A, 'c'))
println("Convergência de A pelo critério de Sassenfield: ", SEL_CritConverg(A, 's'))
```

```
Convergência de A pelo critério das linhas....: 9.496148069437828
Convergência de A pelo critério das colunas...: 11.04846703832951
Convergência de A pelo critério de Sassenfield: 1300.1595201331788
```