Universidade Estadual de Santa Cruz – UESC

Relatórios de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por João Carlos Ribas Chaves Júnior

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2022.1

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2022

ÍNDICE

Linguagem(ns) Escolhida(s) e justificativas

A linguagem escolhida para o desenvolvimento dos métodos foi a linguagem de programação Python, uma linguagem interpretada de alto nível que possui diversas vantagens como uma estrutura de código limpa e legibilidade com seu recuo significativo perceptível. Outra vantagem é a vasta gama de biblioteca matemáticas que se tem para trabalhar facilitando assim no processo de implementação dos métodos.

Durante a implementação dos métodos não obtive nenhuma dificuldade em relação à linguagem escolhida, mas em si ao entendimento dos métodos matemáticos para sua implementação na linguagem de programação.

Em questão de escolha da linguagem foi devido ao estudo sobre a mesma durante o período do desenvolvimento e implementação dos métodos, podendo assim verificar as dificuldades e conseguir obter ainda mais conhecimento sobre a linguagem e assim influenciar a ir mais a fundo do conhecimento da mesma.

1. Método da Bissecção

1.1 Estratégia de Implementação

No começo tive que obter uma forma de fazer a leitura da função a partir do arquivo de texto, em que a linguagem consiga interpretar e resolver. Para isso utilizei **eval()** uma função nativa do python que analisa o argumento da expressão e o avalia como uma expressão python. Posteriormente foi necessário utilizar uma biblioteca externa chamada **numpy**, em que se tem uma vasta gama de funções matemáticas.

A primeira função desenvolvida foi a **fun()**, uma função que recebe a função matemática lida do arquivo a ser calculada e o valor de substituição de x, retornando assim o resultado.

```
def fun(funcao, x)
```

Seguidamente temos a função principal, no qual ele recebe a função vinda do arquivo texto juntamente com os valores de **a**, **b** e **e** (o valor da precisão).

```
def bisseccao_(funcao, a, b, e):
```

Dentro dessa função temos uma condicional, no qual é que se o produto f(a) com f(b) for menor que 0 se tem uma raiz no intervalo [a,b] podendo assim encontrar esse valor, se não for menor que 0 imprimirá na tela "**Não a raiz nesse intervalo!**".

```
if fun(funcao, a) * fun(funcao, b) < 0
else: print("Não a raiz nesse intervalo!\n")
```

Continuando dentro da condição atribuímos 1 a uma variável qualquer, no código escolhi a variável **i** para contar o número de iterações e em seguida fizemos o primeiro cálculo do ponto intermediário do intervalo [a,b] e armazenado em c.

```
i = 1
c = (a + b) / 2
```

Em seguida temos o laço que irá contar as iterações, possuindo uma condição de parada, no qual é enquanto o valor absoluto da diferença entre **b** por **a** for maior que **e**(precisão), o laço continua.

```
while abs(b - a) > e
```

Dentro do laço se tem uma condicional, que se o produto de f(a) por f(c) resultar em um valor menor que 0 atualiza o valor de **b** que recebe o valor de **c**, caso o valor resultante não seja menor que 0 **a** recebe o valor **c.**

```
if fun(funcao, a) * fun(funcao, c) < 0: b = c
else: a = c
```

Calcula-se novamente a raiz e incrementa o valor de **i** por 1.

```
c = (a + b) / 2
i = i + 1
```

Ao final do laço gravamos o resultado no arquivo de saída, com a função utilizada, a tolerância utilizada, o número de iterações até obter o resultado, o valor de **c** e o resultado da função em relação a **c.**

```
fileOutput.write(f"Função: {str(funcao)}")
fileOutput.write(f"Tolerância: {str(b - a)}\n")
fileOutput.write(f"Número de iterações: {i}\n")
fileOutput.write(f"Valor de c: {float(c)}\n")
fileOutput.write(f"f(c): {fun(funcao, c)}\n")
```

1.2 Estrutura dos Arquivos de Entrada/Saída

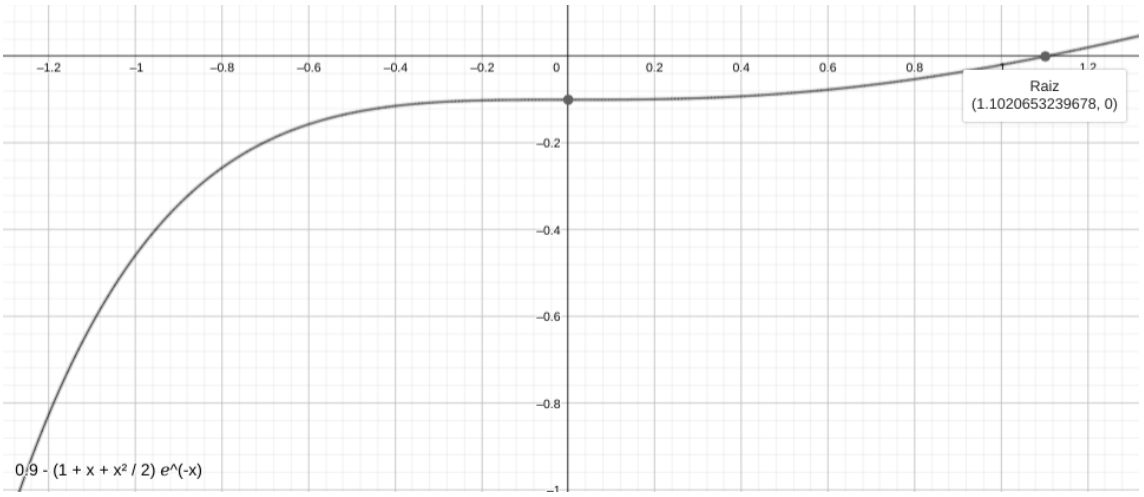
O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente da função, dos extremos ‘a’ e ‘b’ e a ‘e’(precisão). O arquivo texto de saída dividido em função, tolerância número de iterações, valor de c e f(c), respectivamente.

1.3 Problema teste 1, 2, 3…

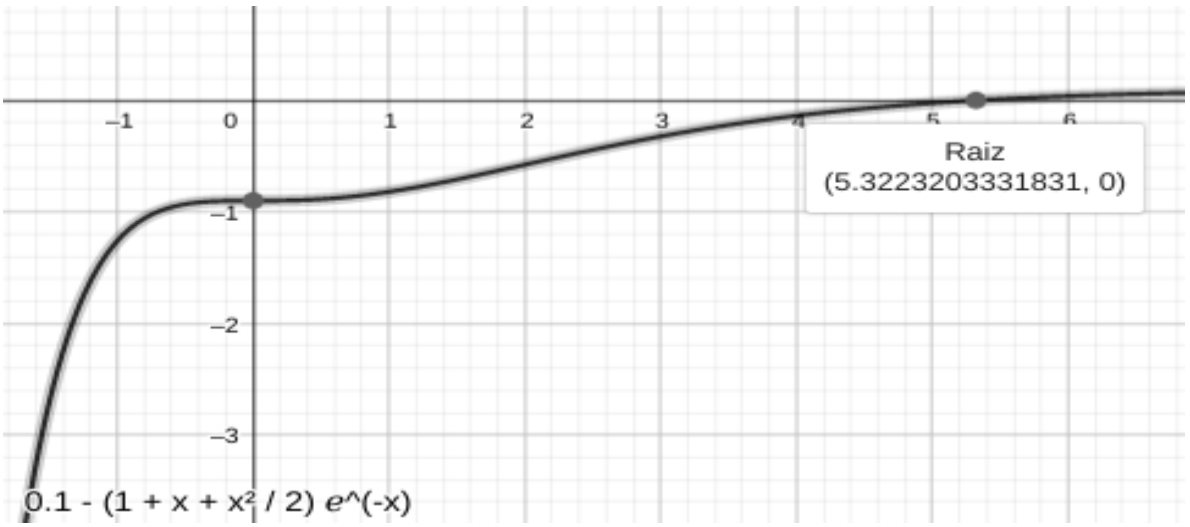
-Problema 3.3 pág. 97 Cálculo Numérico(Neide Franco)

Gráfico da função no GeoGebra:

f(x):0.9 - (1 + x + x² / 2) *e*^(-x)



g(x):0.1 - (1 + x + x² / 2) *e*^(-x)



Dados de entrada f(x):

Função: 0.9-(1+x+((x**(2))/2))*e**(-x)

Extremo a: 0.01

Extremo b: 2

Erro tolerável: 0.000000000000001

Dados de saída f(x):

Função: 0.9-(1+x+((x**(2))/2))*e**(-x)

Tolerância: 8.881784197001252e-16

Número de iterações: 52

Valor de c: 1.1020653282493211

f(c): 1.1102230246251565e-16

Dados de entrada g(x):

Função: 0.1-(1+x+((x**(2))/2))*e**(-x)

Extremo a: 4

Extremo b: 6

Erro tolerável: 0.000000000000001

Dados de saída g(x):

Função: 0.1-(1+x+((x**(2))/2))*e**(-x)

Tolerância: 8.881784197001252e-16

Número de iterações: 52

Valor de c: 5.32232033783421

f(c): -1.3877787807814457e-17

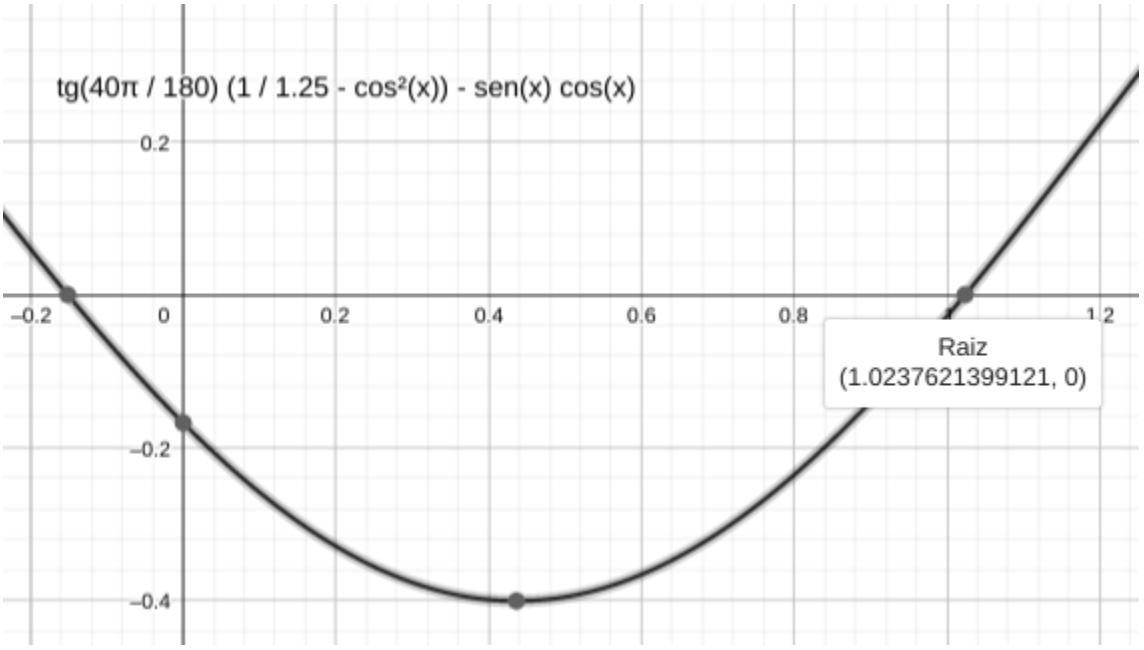
Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram f(x) = **1.1020653282493211** e g(x) = **5.32232033783421** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

-Problema 3.6 página 100 Cálculo Numérico (Neide Franco)

Gráfico da função no GeoGebra:

Função h(x): tg(40π / 180) (1 / 1.25 - cos²(x)) - sen(x) cos(x)



Dados de entrada h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Extremo a: 0.1

Extremo b: 2

Erro tolerável: 0.000000000000001

Dados de saída h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Tolerância: 8.881784197001252e-16

Número de iterações: 52

Valor de c: 1.0237621399089827

f(c): -1.6653345369377348e-16

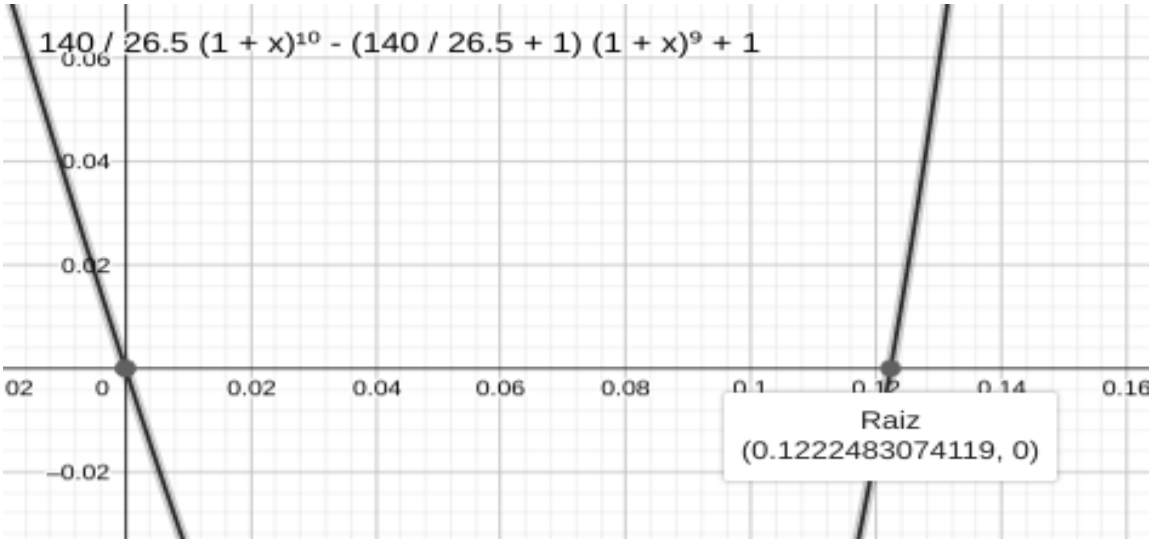
Análise dos resultados:

Comparando o valor obtido através do método implementado que foi f(x) = **1.0237621399089827** com o valor apresentado pelo gráfico tivemos um bom retorno do método implementado.

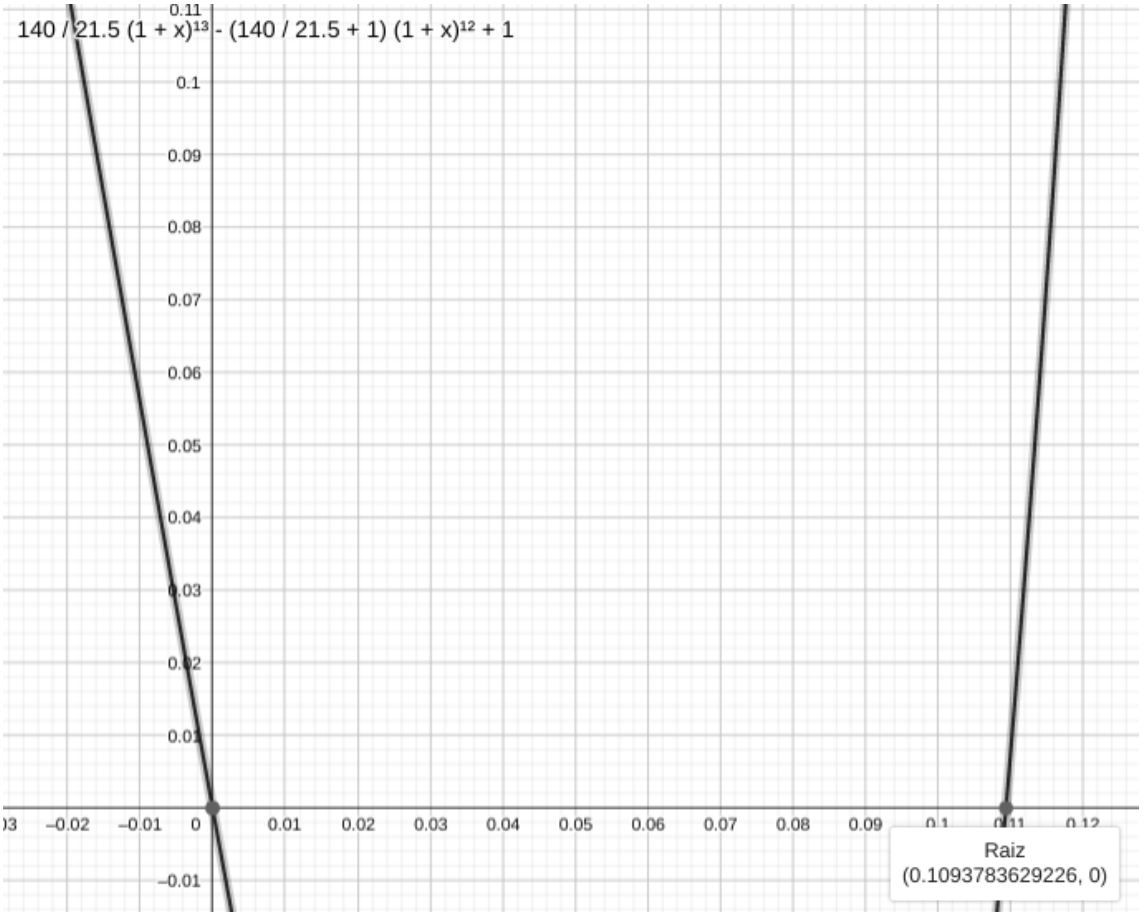
-Problema 3.8 pág. 100 Cálculo Numérico(Neide Franco)

Gráficos das funções no GeoGebra:

Função i(x) = 140 / 26.5 (1 + x)¹⁰ - (140 / 26.5 + 1) (1 + x)⁹ + 1



Função j(x) = 140 / 21.5 (1 + x)¹³ - (140 / 21.5 + 1) (1 + x)¹² + 1



Dados de entrada i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.000000000000001

Dados de saída de i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Tolerância: 8.743006318923108e-16

Número de iterações: 50

Valor de c: 0.12224830741204704

f(c): 0.0

Dados de entrada j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.000000000000001

Dados de saída j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Tolerância: 8.743006318923108e-16

Número de iterações: 50

Valor de c: 0.10937836292454994

f(c): 3.552713678800501e-15

Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram i(x) = **0.12224830741204704** e j(x) = **0.10937836292454994** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

1.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

2. Método da Posição Falsa

2.1 Estratégia de Implementação

No começo tive que obter uma forma de fazer a leitura da função a partir do arquivo de texto, em que a linguagem consiga interpretar e resolver. Para isso utilizei **eval()** uma função nativa do python que analisa o argumento da expressão e o avalia como uma expressão python. Posteriormente foi necessário utilizar uma biblioteca externa chamada **numpy**, em que se tem uma vasta gama de funções matemáticas.

A primeira função desenvolvida foi a **fun()**, uma função que recebe a função matemática lida do arquivo a ser calculada e o valor de substituição de x, retornando assim o resultado.

```
def fun(funcao, x)
```

Seguidamente temos a função principal, no qual ele recebe a função vinda do arquivo texto juntamente com os valores de **a**, **b** e **e** (o valor da precisão).

```
def posicaoFalsa(funcao, a, b, e):
```

Dentro dessa função temos uma condicional, no qual é que se o produto f(a) com f(b) for menor que 0 se tem uma raiz no intervalo [a,b] podendo assim encontrar esse valor, se não for menor que 0 imprimirá na tela "**Não a raiz nesse intervalo!**".

```
if fun(funcao, a) * fun(funcao, b) < 0:
```

```
else: print("Não a raiz nesse intervalo!\n")
```

Continuando dentro da condição atribuo 1 a uma variável qualquer, no código escolhi a variável **i** para contar o número de iterações e em seguida faz o primeiro cálculo do ponto intermediário do intervalo [a,b] e armazenado em c.

```
i = 1
```

```
c = ((a * fun(funcao, b)) - (b * fun(funcao, a)))/(fun(funcao, b) - fun(funcao, a))
```

Em seguida temos o laço que irá contar as iterações, possuindo uma condição de parada, no qual é enquanto o valor absoluto de f(c) for maior que a **e**(precisão), o laço continua.

```
while abs(fun(funcao, c)) > e:
```

Dentro do laço tem uma condicional, no qual é se o produto de f(a) por f(c) resultar em um valor menor que 0 atualiza o valor de **b** que recebe o valor de **c**, caso o valor resultante não seja menor que 0 **a** recebe o valor **c.**

```
if fun(funcao, a) * fun(funcao, c) < 0: b = c

else: a = c
```

Calcula-se novamente a raiz e incrementa o valor de **i** por 1.

```
c = ((a * fun(funcao, b)) - (b * fun(funcao, a)))/(fun(funcao, b) - fun(funcao, a))

i = i + 1
```

Ao final do laço gravamos o resultado no arquivo de saída, com a função utilizada, a tolerância utilizada, o número de iterações até obter o resultado, o valor de **c** e o resultado da função em relação a **c.**

```
fileOutput.write(f"Tolerância: {str(b - a)}\n")

fileOutput.write(f"Número de iterações: {i}\n")

fileOutput.write(f"Valor de c: {float(c)}\n")

fileOutput.write(f"f(c): {fun(funcao, c)}\n")
```

2.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente da função, dos extremos ‘a’ e

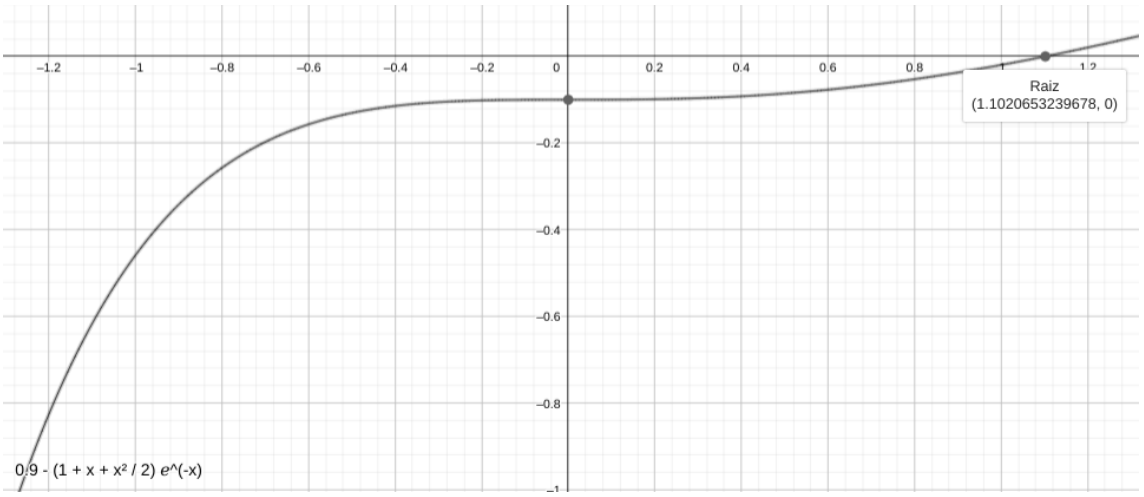
‘b’ e a ‘e’(precisão). O arquivo texto de saída dividido em função, tolerância, número de iterações, valor de c e f(c), respectivamente.

2.3 Problema teste 1, 2, 3...

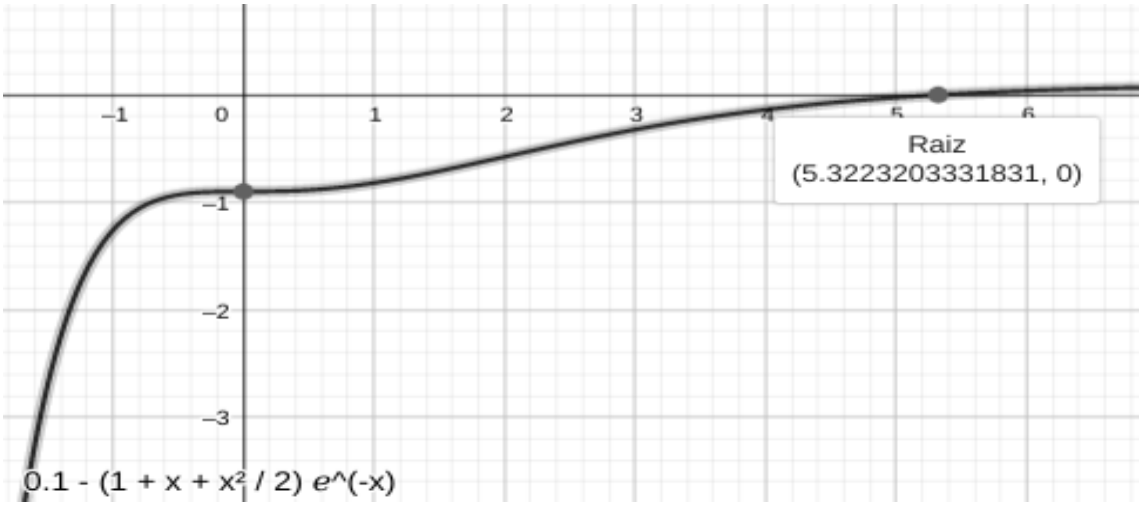
-Problema 3.3 pág. 97 Cálculo Numérico(Neide Franco)

Gráfico da função no GeoGebra:

f(x):0.9 - (1 + x + x² / 2) *e*^(-x)



g(x):0.1 - (1 + x + x² / 2) *e*^(-x)



Dados de entrada f(x):

Função: 0.9-(1+x+((x**(2))/2))*e**(-x)

Extremo a: 0.01

Extremo b: 2

Erro tolerável: 0.0001

Dados de saída f(x):

Função: 0.9-(1+x+((x**(2))/2))*e**(-x)

Tolerância: 0.8989700602368913

Número de iterações: 6

Valor de c: 1.101869571108599

f(c): -3.9486329633464656e-05

Dados de entrada g(x):

Função: 0.1-(1+x+((x**(2))/2))*e**(-x)

Extremo a: 4

Extremo b: 6

Erro tolerável: 0.0001

Dados de saída g(x):

f(x): 0.1-(1+x+((x**(2))/2))*e**(-x)

Tolerância: 1.3256613695352515

Número de iterações: 6

Valor de c: 5.323450063238776

f(c): 7.807957412804545e-05

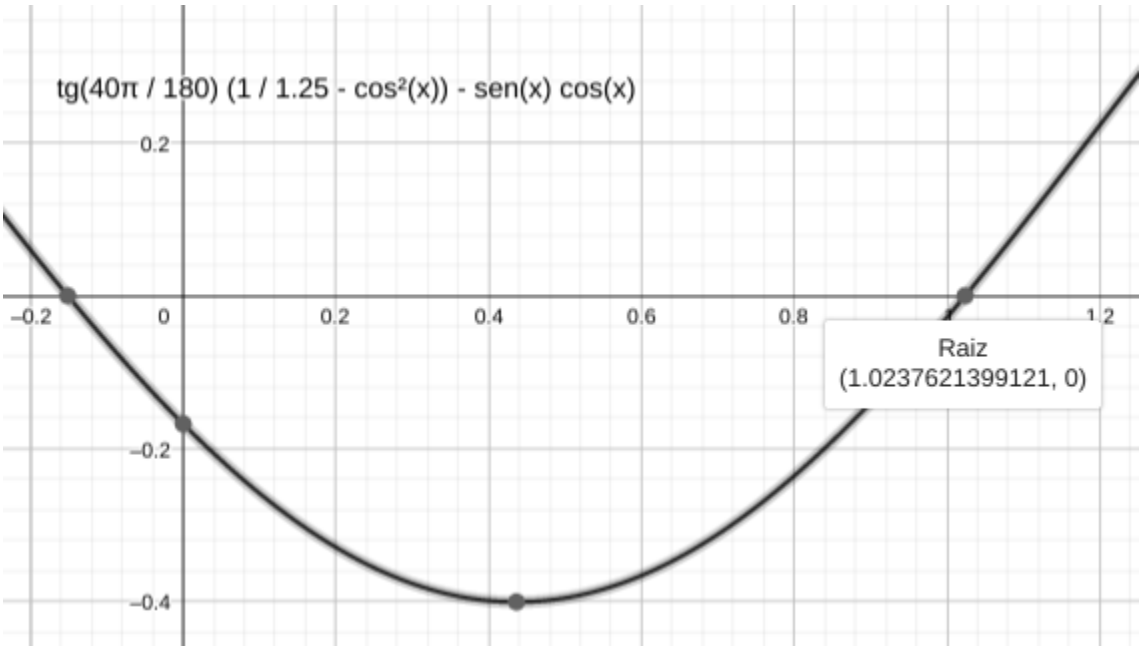
Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram f(x) = **1.101869571108599** e g(x) = **5.323450063238776** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

-Problema 3.6 página 100 Cálculo Numérico (Neide Franco)

Gráfico da função no GeoGebra:

Função h(x): tg(40π / 180) (1 / 1.25 - cos²(x)) - sen(x) cos(x)



Dados de entrada h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Extremo a: 0.1

Extremo b: 2

Erro tolerável: 0.0001

Dados de saída h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Tolerância: 0.014065975658888785

Número de iterações: 5

Valor de c: 1.023760230610521

f(c): -2.299586624676664e-06

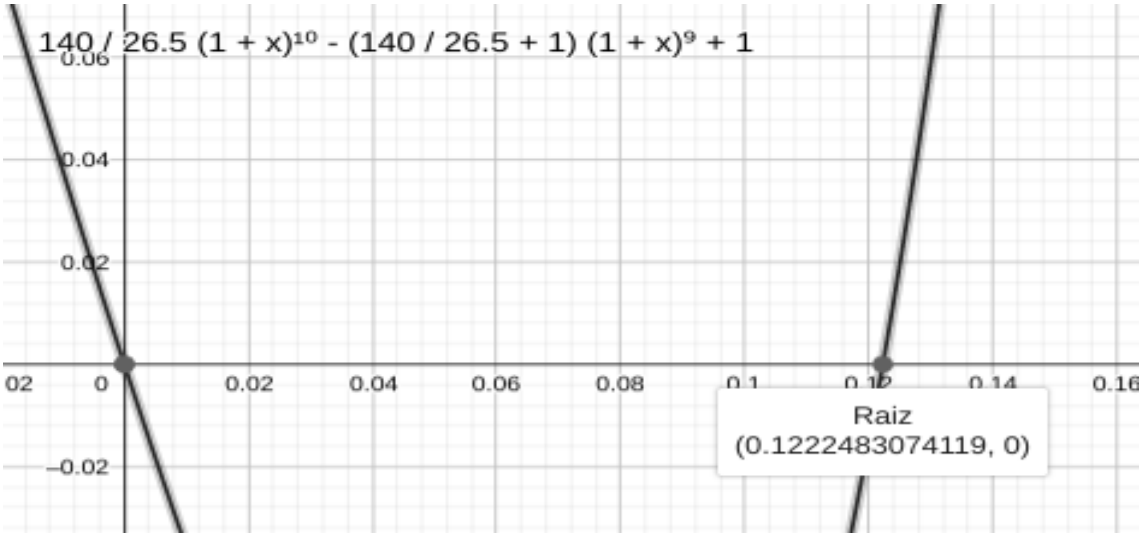
Análise dos resultados:

Comparando o valor obtido através do método implementado que foi f(x) = **1.023760230610521** com o valor apresentado pelo gráfico tivemos um bom retorno do método implementado.

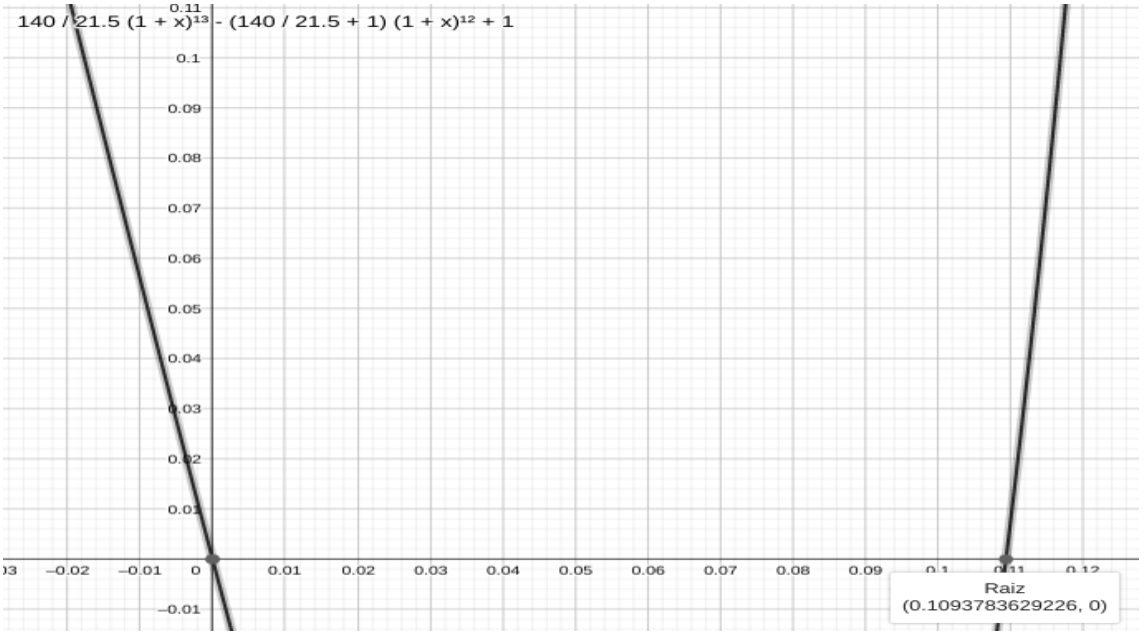
-Problema 3.8 pág. 100 Cálculo Numérico(Neide Franco)

Gráficos das funções no GeoGebra:

Função $i(x) = 140 / 26.5 (1 + x)^{10} - (140 / 26.5 + 1) (1 + x)^{9} + 1$



Função $j(x) = 140 / 21.5 (1 + x)^{13} - (140 / 21.5 + 1) (1 + x)^{12} + 1$



Dados de entrada i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.0001

Dados de saída de i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Tolerância: 0.3777662964776999

Número de iterações: 302

Valor de c: 0.12223429699639743

f(c): -9.661797973592456e-05

Dados de entrada j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.0001

Dados de saída j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Tolerância: 0.3906301744275894

Número de iterações: 838

Valor de c: 0.10936995967589536

f(c): -9.92382463671504e-05

Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram i(x) = **0.12223429699639743** e j(x) = **0.10936995967589536** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

2.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

3. Método de Newton-Raphson

3.1 Estratégia de Implementação

No começo tive que obter uma forma de fazer a leitura da função a partir do arquivo de texto, em que a linguagem consiga interpretar e resolver. Para isso utilizei **eval()** uma função nativa do python que analisa o argumento da expressão e o avalia como uma expressão python. Posteriormente foi necessário utilizar uma biblioteca externa chamada **sympy** e utilizar a função 'diff()', que é um função que recebe uma expressão como parâmetro e retorna a derivada dessa função, utilizando assim para derivar a função recebida do arquivo texto de entrada.

A primeira função desenvolvida foi a **fun()**, uma função que recebe a função matemática lida do arquivo a ser calculada e o valor de substituição de x, retornando assim o resultado.

```
def fun(funcao, x)
```

Seguidamente temos a função principal, no qual ele recebe a função vinda do arquivo texto juntamente com os valores de **a**, **b**, **devf** (a função derivada recebida do arquivo de entrada) e **e** (o valor da precisão).

```
def newton_(funcao, a, b, devf, e):
```

Dentro dessa função temos uma condicional, no qual é que se o produto f(a) com f(b) for menor que 0 se tem uma raiz no intervalo [a,b] podendo assim encontrar esse valor, se não for menor que 0 imprimirá na tela "**Não a raiz nesse intervalo!**".

```
if fun(funcao, a) * fun(funcao, b) < 0
else: print("Não a raiz nesse intervalo!\n")
```

Continuando dentro da condição atribuo 1 a uma variável qualquer, no código escolhi a variável **i** para contar o número de iterações e em seguida faz o primeiro cálculo do ponto intermediário do intervalo [a,b] e armazenado em c.

```
i = 1
```

```
c = (a+b)/2
```

Em seguida temos o laço que irá contar as iterações, possuindo uma condição de parada, no qual é enquanto o valor absoluto de **f(c)** for maior que **e**(precisão), o laço continua.

```
while abs(fun(funcao, c)) > e:
```

Dentro do laço calculamos novamente a raiz e incrementamos o valor de i por 1. O cálculo da raiz é a diferença entre **c** e a divisão de **f(c)** por **f'(c)** que é a derivada da função.

```
c = c - (fun(funcao, c)/fun(devf, c))
```

```
i = i + 1
```

Ao final do laço gravamos o resultado no arquivo de saída, com a função utilizada, o número de iterações até obter o resultado, o valor de **c** e o resultado da função em relação a **c.**

```
fileOutput.write(f"Função: {str(funcao)}")
```

```
fileOutput.write(f"Número de iterações: {i}\n")
```

```
fileOutput.write(f"Valor de c: {float(c)}\n")
```

```
fileOutput.write(f"f(c): {fun(funcao, c)}\n\n")
```

3.2 Estrutura dos Arquivos de Entrada/Saída

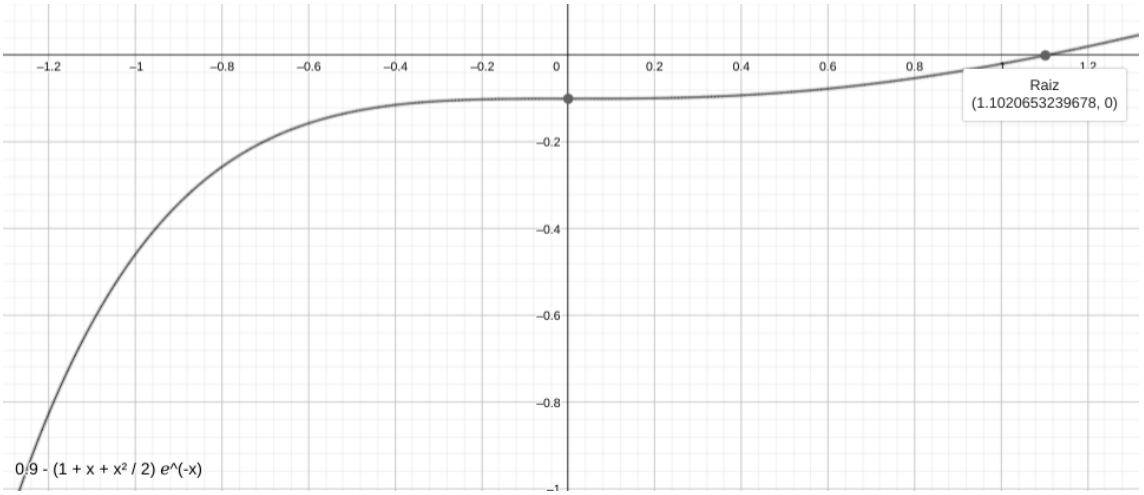
O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente da função, dos extremos 'a' e 'b' e a 'e'(precisão). O arquivo texto de saída dividido em função, número de iterações, valor de c e f(c), respectivamente.

3.3 Problema teste 1, 2, 3...

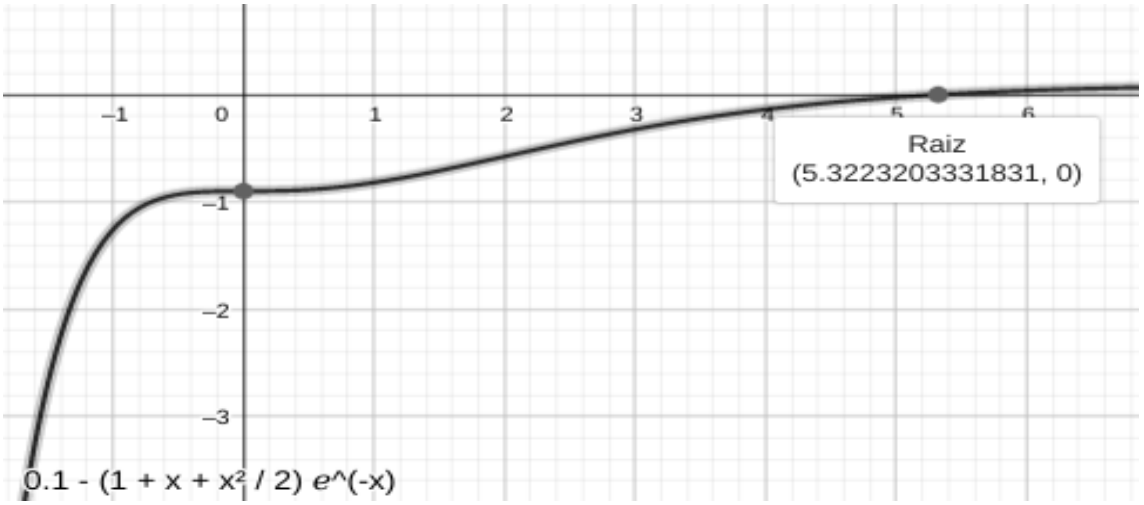
-Problema 3.3 pág. 97 Cálculo Numérico(Neide Franco)

Gráfico da função no GeoGebra:

f(x):0.9 - (1 + x + x² / 2) *e*^(-x)



g(x):0.1 - (1 + x + x² / 2) *e*^(-x)



Dados de entrada f(x):

Função: 0.9-(1+x+((x**(2))/2))*2.718281828459**(-x)

Extremo a: 0.01

Extremo b: 2

Erro tolerável: 0.0001

Dados de saída f(x):

Função: 0.9-(1+x+((x**(2))/2))*2.718281828459**(-x)

Número de iterações: 3

Valor de c: 1.1020735481623742

f(c): 1.6581830295514521e-06

Dados de entrada g(x):

Função: 0.1-(1+x+((x**(2))/2))*2.718281828459**(-x)

Extremo a: 4

Extremo b: 6

Erro tolerável: 0.0001

Dados de saída g(x):

Função: 0.1-(1+x+((x**(2))/2))*2.718281828459**(-x)

Número de iterações: 3

Valor de c: 5.322048692650423

f(c): -1.8782634262026754e-05

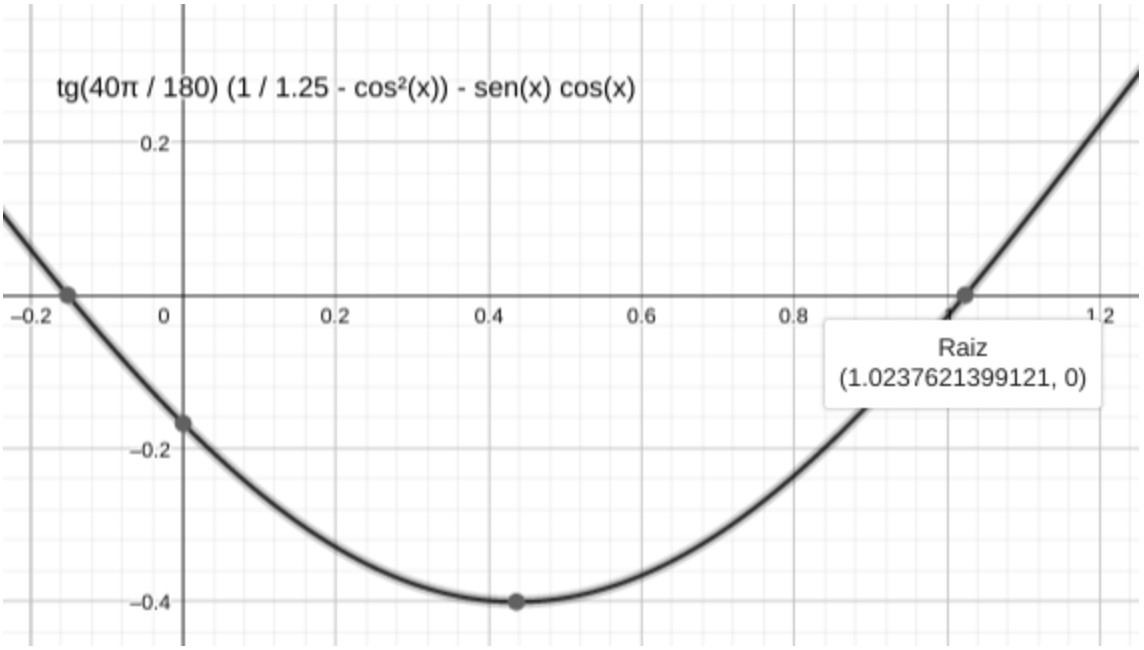
Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram f(x) = **1.1020735481623742** e g(x) = **5.322048692650423** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

-Problema 3.6 página 100 Cálculo Numérico (Neide Franco)

Gráfico da função no GeoGebra:

Função h(x): tg(40π / 180) (1 / 1.25 - cos²(x)) - sen(x) cos(x)



Dados de entrada h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Extremo a: 0.1

Extremo b: 2

Erro tolerável: 0.0001

Dados de saída h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Número de iterações: 3

Valor de c: 1.0237621677453808

f(c): 3.352658656252672e-08

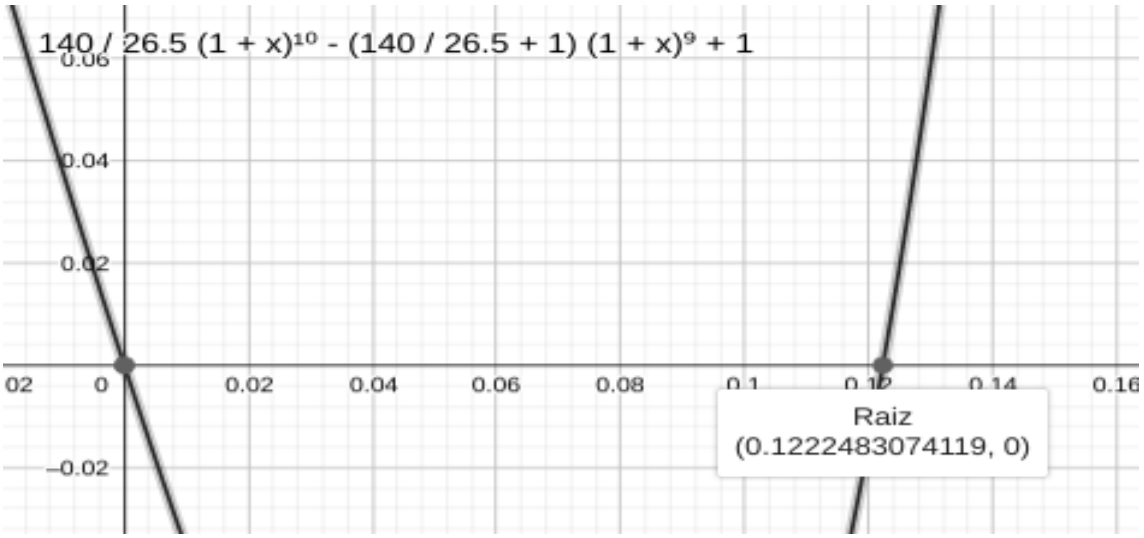
Análise dos resultados:

Comparando o valor obtido através do método implementado que foi f(x) = **1.0237621677453808** com o valor apresentado pelo gráfico tivemos um retorno bastante satisfatório em relação aos outros métodos implementados.

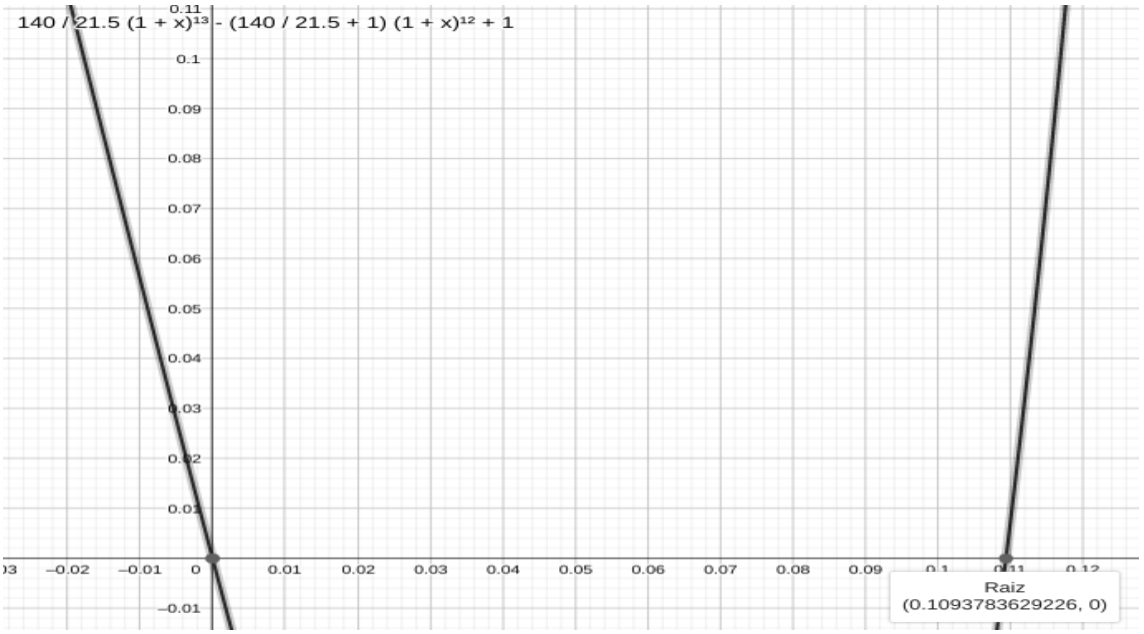
-Problema 3.8 pág. 100 Cálculo Numérico(Neide Franco)

Gráficos das funções no GeoGebra:

Função i(x) = 140 / 26.5 (1 + x)¹⁰ - (140 / 26.5 + 1) (1 + x)⁹ + 1



Função j(x) = 140 / 21.5 (1 + x)¹³ - (140 / 21.5 + 1) (1 + x)¹² + 1



Dados de entrada i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.0001

Dados de saída de i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Número de iterações: 6

Valor de c: 0.12225444118027043

f(c): 4.231065755888608e-05

Dados de entrada j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.0001

Dados de saída j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Número de iterações: 7

Valor de c: 0.10937901076059792

f(c): 7.651746788184255e-06

Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram i(x) = **0.12225444118027043** e j(x) = **0.10937901076059792** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

3.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

4. Método da Secante

4.1 Estratégia de Implementação

No começo tive que obter uma forma de fazer a leitura da função a partir do arquivo de texto, em que a linguagem consiga interpretar e resolver. Para isso utilizei **eval()** uma função nativa do python que analisa o argumento da expressão e o avalia como uma expressão python. Posteriormente foi necessário utilizar uma biblioteca externa chamada **math**, em que se tem uma vasta gama de funções matemáticas.

A primeira função desenvolvida foi a **fun()**, uma função que recebe a função matemática lida do arquivo a ser calculada e o valor de substituição de x, retornando assim o resultado.

```
def fun(funcao, x)
```

Seguidamente temos a função principal, no qual ele recebe a função vinda do arquivo texto juntamente com os valores de **a**, **b** e **e** (o valor da precisão).

```
def secante_(funcao, a, b, e):
```

Dentro dessa função temos uma condicional, no qual é que se o produto **f(a)** com **f(b)** for menor que 0 se tem uma raiz no intervalo [a,b] podendo assim encontrar esse valor, se não for menor que 0 imprimirá na tela "**Não a raiz nesse intervalo!**".

```
if fun(funcao, a) * fun(funcao, b) < 0:

else: print("Não a raiz nesse intervalo!\n")
```

Continuando dentro da condição atribuo 1 a uma variável qualquer, no código escolhi a variável **i** para contar o número de iterações e em seguida faz o primeiro cálculo do ponto intermediário do intervalo [a,b] e armazenado em **c**.

```
i = 1
```

```
c = b - (fun(funcao, b) * (b - a)) / (fun(funcao, b) - fun(funcao, a))
```

Em seguida temos o laço que irá contar as iterações, possuindo uma condição de parada, no qual é enquanto o valor absoluto de **f(c)** for maior que a **e**(precisão), o laço continua.

```
while abs(fun(funcao, c)) > e:
```

Dentro do laço fazemos a troca dos valores de **a** e **b** por **b** e **c** respectivamente, calculamos novamente o ponto intermediário do intervalo [a,b] e incrementamos o valor de **i** por 1.

```
a,b = b, c
```

```
c = b - (fun(funcao, b) * (b - a)) / (fun(funcao, b) - fun(funcao, a))
```

```
i = i + 1
```

Ao final do laço gravamos o resultado no arquivo de saída, com a função utilizada, o número de iterações até obter o resultado, o valor de **c** e o resultado da função em relação a **c.**

```
fileOutput.write(f"Função: {str(funcao)}")

fileOutput.write(f"Número de iterações: {i}\n")

fileOutput.write(f"Valor de c: {float(c)}\n")

fileOutput.write(f"f(c): {fun(funcao, c)}\n\n")
```

4.2 Estrutura dos Arquivos de Entrada/Saída

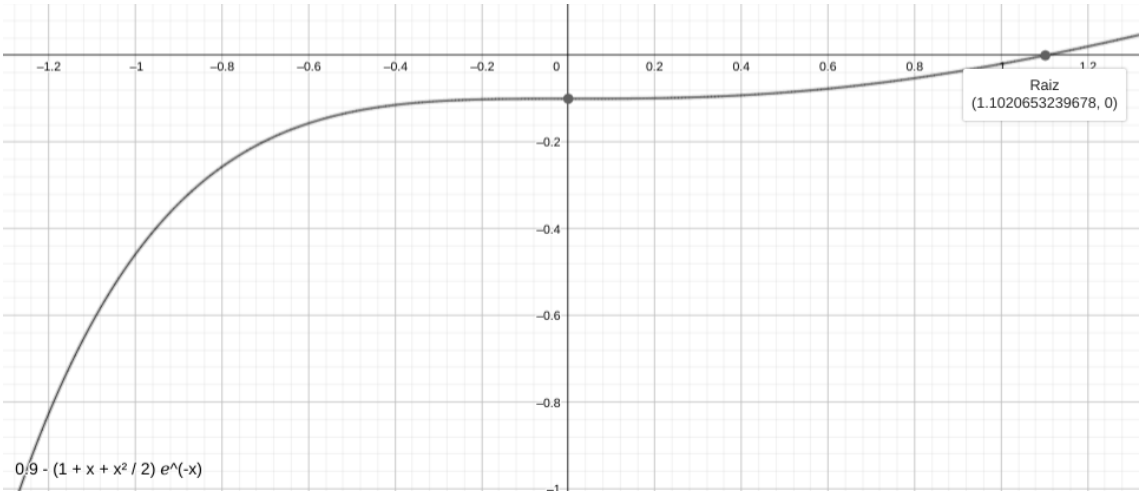
O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. O script está padronizado para fazer a leitura respectivamente da função, dos extremos ‘a’ e ‘b’ e a ‘e’(precisão). O arquivo texto de saída dividido em função, número de iterações, valor de c e f(c), respectivamente.

4.3 Problema teste 1, 2, 3...

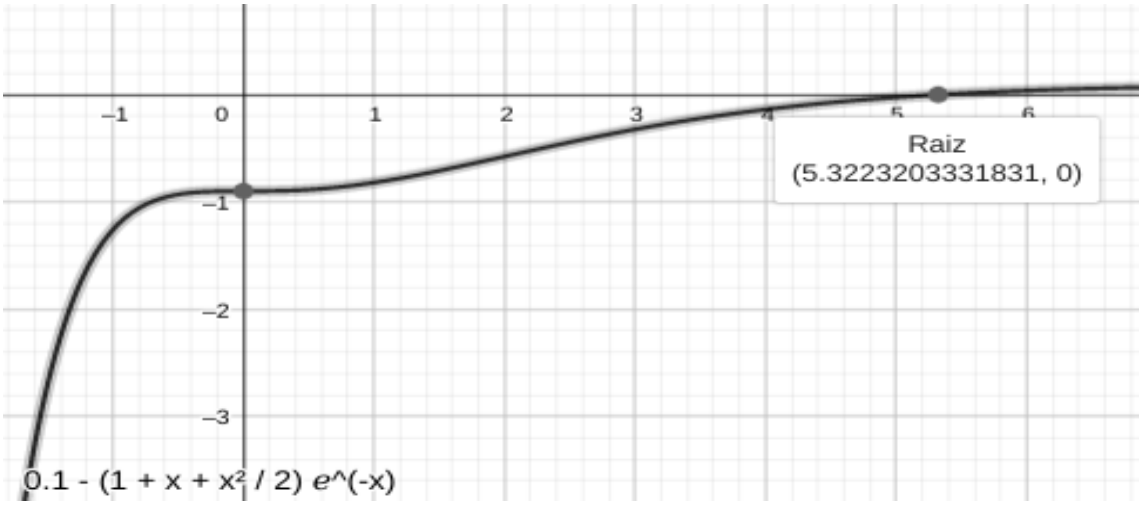
-Problema 3.3 pág. 97 Cálculo Numérico(Neide Franco)

Gráfico da função no GeoGebra:

f(x):0.9 - (1 + x + x² / 2) *e*^(-x)



g(x):0.1 - (1 + x + x² / 2) *e*^(-x)



Dados de entrada f(x):

Função: 0.9-(1+x+((x**(2))/2))*e**(-x)

Extremo a: 0.01

Extremo b: 2

Erro tolerável: 0.0001

Dados de saída f(x):

Função: 0.9-(1+x+((x**(2))/2))*e**(-x)

Número de iterações: 5

Valor de c: 1.1020194754696127

f(c): -9.249565845803609e-06

Dados de entrada g(x):

Função: 0.1-(1+x+((x**(2))/2))*e**(-x)

Extremo a: 4

Extremo b: 6

Erro tolerável: 0.0001

Dados de saída g(x):

Função: 0.1-(1+x+((x**(2))/2))*e**(-x)

Número de iterações: 4

Valor de c: 5.322406054983901

f(c): 5.926165466824118e-06

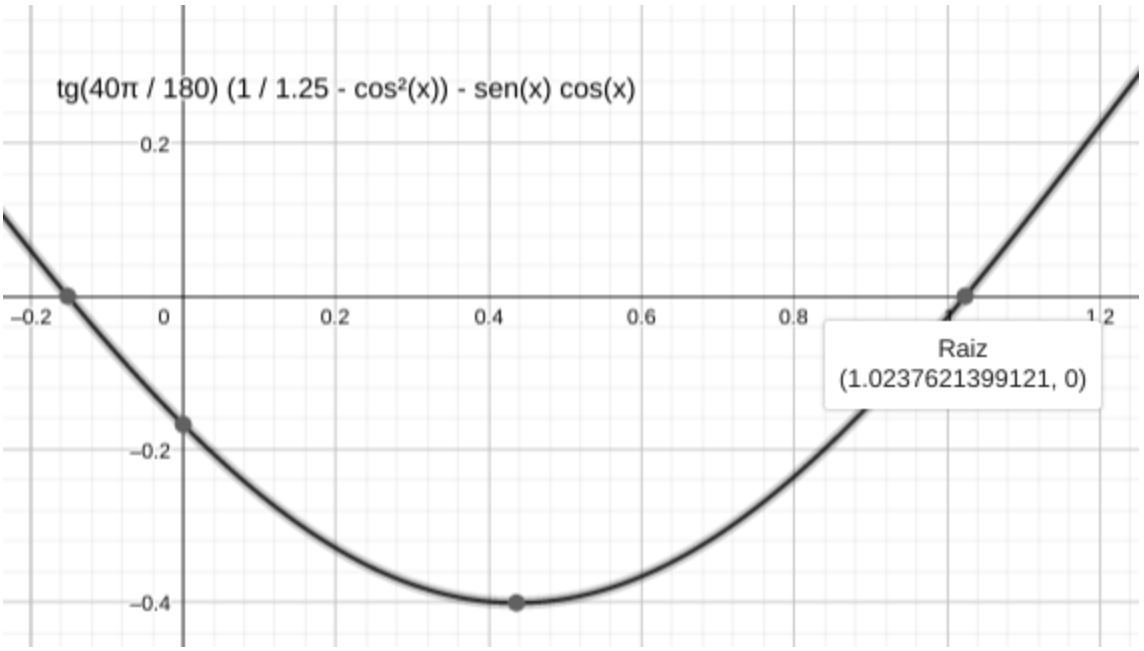
Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram f(x) = **1.1020194754696127** e g(x) = **5.322406054983901** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

-Problema 3.6 página 100 Cálculo Numérico (Neide Franco)

Gráfico da função no GeoGebra:

Função h(x): tg(40π / 180) (1 / 1.25 - cos²(x)) - sen(x) cos(x)



Dados de entrada h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Extremo a: 0.1

Extremo b: 2

Erro tolerável: 0.0001

Dados de saída h(x):

Função: tan(40*pi/180)*((1/1.25)-cos(x)**2)-sin(x)*cos(x)

Número de iterações: 5

Valor de c: 1.0237518309286284

f(c): -1.2416241133406114e-05

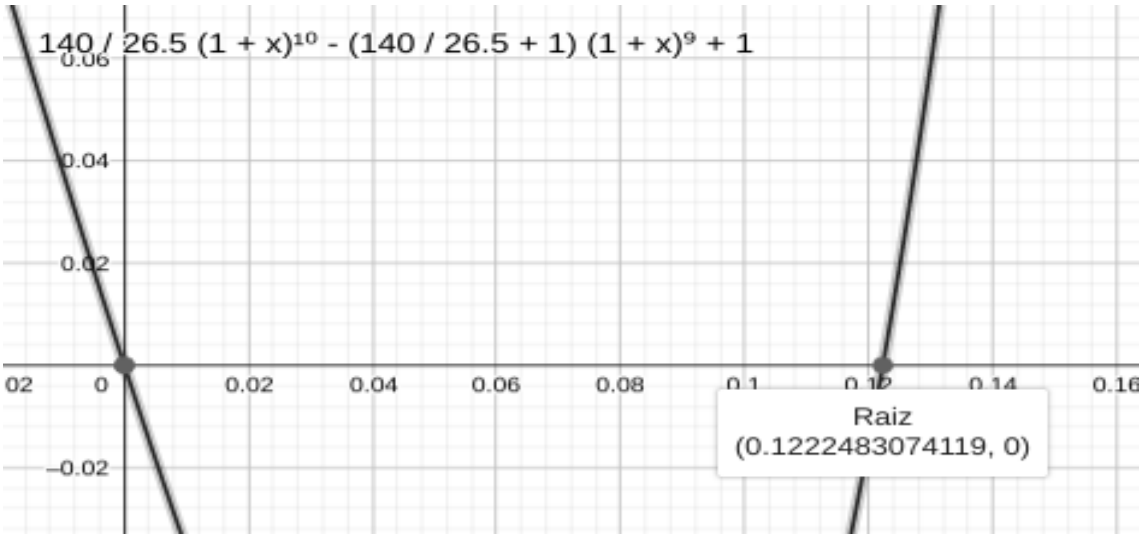
Análise dos resultados:

Comparando o valor obtido através do método implementado que foi f(x) = **1.0237518309286284** com o valor apresentado pelo gráfico tivemos um bom retorno do método implementado.

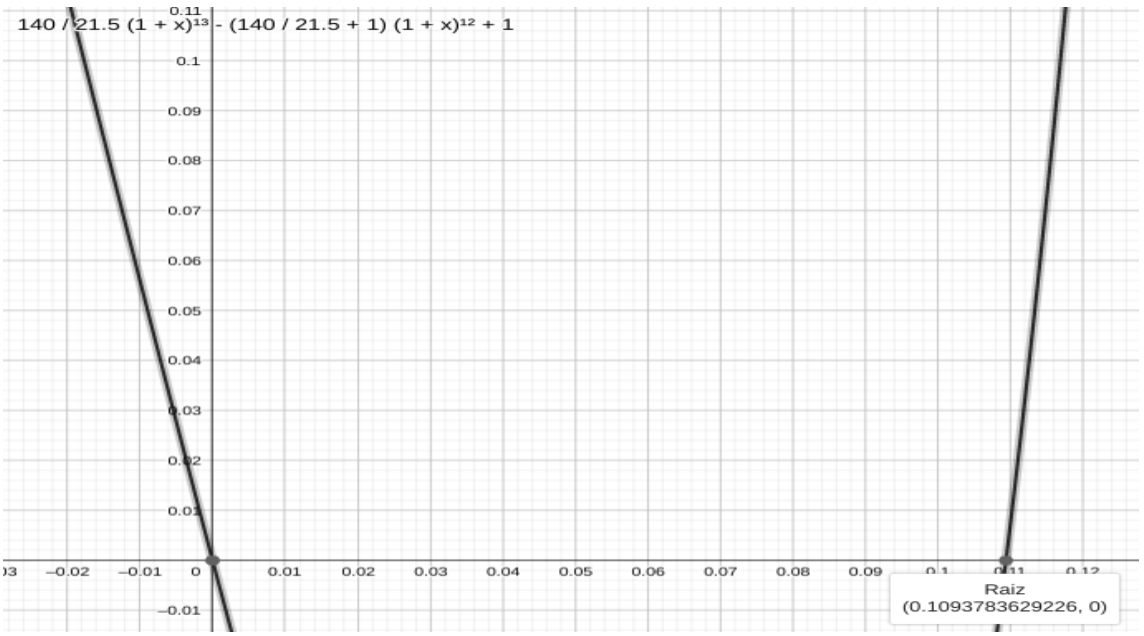
-Problema 3.8 pág. 100 Cálculo Numérico(Neide Franco)

Gráficos das funções no GeoGebra:

Função i(x) = 140 / 26.5 (1 + x)¹⁰ - (140 / 26.5 + 1) (1 + x)⁹ + 1



Função j(x) = 140 / 21.5 (1 + x)¹³ - (140 / 21.5 + 1) (1 + x)¹² + 1



Dados de entrada i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.0001

Dados de saída de i(x):

Função: (140/26.50)*((1+x)**10)-((140/26.50)+1)*((1+x)**9)+1

Número de iterações: 7

Valor de c: 0.12224823665621064

f(c): -4.88032913636971e-07

Dados de entrada j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Extremo a: 0.01

Extremo b: 0.5

Erro tolerável: 0.0001

Dados de saída j(x):

Função: (140/21.50)*((1+x)**13)-((140/21.50)+1)*((1+x)**12)+1

Número de iterações: 5

Valor de c: 0.10937097405046807

f(c): -8.726041249218497e-05

Análise dos resultados:

Comparando os valores obtidos através do método implementado que foram i(x) = **0.12224823665621064** e j(x) = **0.10937097405046807** com os valores apresentados pelos gráficos tivemos um bom retorno do método implementado.

4.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

5. Método da Eliminação de Gauss

5.1 Estratégia de Implementação

De início implementei o método **eliminaçãoGauss(A, b)** que recebe matriz **A** e vetor **b**, dentro desse método pego a ordem da matriz A e atribuo a uma variável, no código a variável **n**, chamo outro método **pivo()** passando os vetores A e b,respectivamente. O método **pivo()** foi criado para fazer a verificação de cada coluna se o pivô é diferente do zero. Fazendo a verificação também se o pivô é o maior número da sua coluna, em que caso o pivô não seja o maior número da sua coluna é feita a troca.

```
n = len(A)
```

```
pivo(A, b)
```

Seguindo no método faço um laço 1 geral de repetição para fazer a triangulação.

```
for k in range(0, n-1):
```

Continuando dentro do laço 1 tem-se outro laço 2 para atualizar a linha i

```
for i in range(k+1, n):
```

Dentro desse laço 2 calcula-se o fator m, que é o multiplicador da linha.

```
m = -A[i][k]/A[k][k]
```

Continuando tem-se outro laço 3 para atualizar a linha i da matriz, percorrendo todas as colunas j.

```
for j in range(k+1, n):
```

Dentro do laço 3 faço o cálculo do produto do fator m, que é o multiplicador da linha, pela linha do pivô mais a linha em questão.

```
A[i][j] = (m * A[k][j]) + A[i][j]
```

No laço 2 atualizo o vetor b na linha i e zero o elemento na posição A[i][k]

```
b[i] = m * b[k] + b[i]
```

```
A[i][k] = 0
```

Ao fim do laço 1 chamo método **substituicaoRetroativa(A, b)** para resolver o sistema triangular superior, passando os vetores A e b como parâmetro. Dentro desse método pego a ordem da matriz A e atribuo a variável n e inicializo um vetor **x** com tamanho **n** e elementos iguais a 0.

```
n = len(A)
```

```
x = n * [0]
```

Seguindo no método faço um laço 1 que vai variar do último elemento até o primeiro, resolvendo o sistema de baixo para cima extraindo os resultados.

```
for i in range(n-1, -1, -1):
```

Dentro do laço 1 atribuo a uma variável soma o valor de 0 para realizar o cálculo somatório.

```
soma = 0
```

Continuando faço outro laço 2 para fazer o somatório.

```
for j in range(i+1, n):
```

No laço 2 realizar o cálculo do somatório.

```
soma = soma + A[i][j] * x[j]
```

No laço 1 extraio os valores resultantes da matriz.

```
x[i] = (b[i] - soma) / A[i][i]
```

No final do método retorno o vetor solução para o método **eliminacaoGauss()**.

```
return x
```

Na '__main__' passo por parâmetro as matrizes **A** e **B** para o método **eliminacaoGauss(A, B)** e gravo o retorno no arquivo texto de saída.

5.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. De início tive que obter uma forma de pegar os valores do arquivo de entrada e organizar em forma de matriz. Utilizei alguns métodos **readline()** para fazer a leitura da linha por linha , **replace('\n', ' ')** para fazer a remoção da quebra de linha e inserir um espaço vazio no lugar e **split(' ')** para fazer a divisão dos valores de acordo com os espaços; todos os métodos nativos do python.

O arquivo texto de entrada está dividido na altura da matriz e nos seus elementos, respectivamente. O arquivo texto de saída é o vetor resultante.

Para fazer o cálculo de todos os testes terá que apagar a matriz já calculada do arquivo texto de entrada, para assim fazer a resolução de cada teste separadamente e rodar novamente o código para obter os resultados no arquivo texto de saída.

5.3 Problema teste 1, 2, 3…

- Problema 4.1 pág. 150 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

8I1 -4I2 -2I3 = 10

-4I1 +6I2 -2I3 = 0

-2I1 -2I2 +10I3 = 4

Organizando os dados de entrada no arquivo texto:

8 -4 -2 10

-4 6 -2 0

-2 -2 10 4

Dados de saída I1, I2 e I3, respectivamente:

[2.7586206896551726, 2.310344827586207, 1.4137931034482758]

Substituindo os dados de saída no sistema temos:

8*2.7586206896551726 - 4*2.310344827586207-2*1.4137931034482758 = 10

-4*2.7586206896551726 +6*2.310344827586207 -2*1.4137931034482758 = 0

-2*2.7586206896551726 -2*2.310344827586207+10*1.4137931034482758 = 4

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado nos retornou os valores perfeitamente.

-Problema 4.3 pág. 151 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

4p1 -1p2 +0 +0 -1p5 +0 +0 +0 = 0

-1p1 +4p2 -1p3 +0 +0 -1p6 +0 +0 = 0

0 -1p2 +4p3 -1p4 +0 +0 -1p7 +0 = 0

0 +0 -1p3 +4p4 +0 +0 +0 -1p8 = 0

-1p1 +0 +0 +0 +4p5 -1p6 +0 +0 = 1

0 -1p2 +0 +0 -1p5 +4p6 -1p7 +0 = 1

0 +0 -1p3 +0 +0 -1p6 +4p7 -1p8 = 1

0 +0 +0 -1p4 +0 +0 -1p7 +4p8 = 1

Organizando os dados de entrada no arquivo texto:

8

4 -1 0 0 -1 0 0 0 0

-1 4 -1 0 0 -1 0 0 0

0 -1 4 -1 0 0 -1 0 0

0 0 -1 4 0 0 0 -1 0

-1 0 0 0 4 -1 0 0 1

0 -1 0 0 -1 4 -1 0 1

0 0 -1 0 0 -1 4 -1 1

0 0 0 -1 0 0 -1 4 1

Dados de saída p1, p2, p3 , …,p8, respectivamente:

[0.16842105263157894, 0.2421052631578947, 0.2421052631578947, 0.16842105263157892, 0.43157894736842106, 0.5578947368421052, 0.5578947368421052, 0.43157894736842095]

Substituindo os dados de saída no sistema temos:

4*0.16842105263157894-1*0.2421052631578947+0+0-1*0.43157894736842106 +0 +0 +0 = 0

-1*0.16842105263157894 +4*0.242105263157894 -1*0.2421052631578947 +0 +0 -1*0.5578947368421052 +0 +0 = 0

0 -1*0.2421052631578947 +4*0.2421052631578947 -1*0.16842105263157892 +0 +0 -1*0.5578947368421052 +0 = 0

0 +0 -1*0.2421052631578947 +4*0.16842105263157892 +0 +0 +0 -1*0.43157894736842095 = 0

-1*0.16842105263157894 +0 +0 +0 +4*0.43157894736842106
-1*0.5578947368421052 +0 +0 = 1

0 -1*0.2421052631578947 +0 +0 -1*0.43157894736842106
+4*0.5578947368421052 -1*0.5578947368421052 +0 = 1

0 +0 -1*0.2421052631578947 +0 +0 -1*0.5578947368421052
+4*0.5578947368421052 -1*0.43157894736842095 = 1

0 +0 +0 -1*0.16842105263157892 +0 +0 -1*0.5578947368421052
+4*0.43157894736842095 = 1

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado nos retornou os valores perfeitamente.

-Problema 4.6 pág. 153 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

14I1 +4I2 +4Ix = 100

4I1 +7I2 +19Ix = 100

4I1 +7I2 +18Ix = 100

Organizando os dados de entrada no arquivo texto:

14 4 4 100

4 7 19 100

4 7 18 100

Dados de saída I1, I2 e Ix, respectivamente:

[3.658536585365854, 12.195121951219512, -0.0]

Substituindo os dados de saída no sistema temos:

14*3.658536585365854 +4*12.195121951219512 +4(-0.0) = 100

4*3.658536585365854 +7*12.195121951219512 +19*(-0.0) = 100

4*3.658536585365854 +7*12.195121951219512 +18*(-0.0) = 100

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado nos retornou os valores perfeitamente.

5.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

6. Método da Fatoração LU

6.1 Estratégia de Implementação

De início implementei o método **lu(A)** que recebe matriz **A**, esse método decompõe a matriz A no produto de duas matrizes(L e U), sendo L uma matriz triangular inferior e U triangular superior.

```
def lu(A):
```

Dentro desse método pego a ordem da matriz A e atribuo a uma variável, no código a variável **n**, inicializo o vetor L com o vetor retornado da função **identidade(n)**,que é responsável de criar uma matriz com 1's na sua diagonal principal e 0's na triangular superior e inferior.

```
L = identidade(n)
```

Seguindo no método faço um laço 1 geral de repetição para fazer a triangulação.

```
for k in range(0, n-1):
```

Continuando dentro do laço 1 tem-se outro laço 2 para atualizar a linha i.

```
for i in range(k+1, n):
```

Dentro do laço 2 calcula-se o fator m, que é o multiplicador da linha.

```
m = -A[i][k]/A[k][k]
```

Copio o valor de -m para a matriz L na posição [i][k].

```
L[i][k] = -m
```

Continuando tem-se outro laço 3 para atualizar a linha i da matriz, percorrendo todas as colunas j.

```
for j in range(k+1, n):
```

Dentro do laço 3 faço o cálculo do produto do fator m, que é o multiplicador da linha, pela linha do pivô mais a linha em questão.

```
A[i][j] = (m * A[k][j]) + A[i][j]
```

No laço 2 ao final do laço 3 zero o elemento do vetor A na posição [i][k].

```
A[i][k] = 0
```

Ao fim do laço 1 retornam os vetores L e A para a **'main'** do código.

```
return L, A
```

Na '__main__' passo por parâmetro as matrizes **L**, **U** retornadas do método **lu()** e o vetor **B** para o método **lux(L,U,B)** e gravo o retorno no arquivo texto de saída.

O método **lux()** para resolver o sistema LUx=b, resolvendo os dois sistemas triangulares sendo # L matriz triangular inferior e U matriz triangular superior e b é o vetor.

```
def lux(L,U,b)
```

No método utilizo os métodos **substituicaoSucessiva()** e **substituicaoRetroativa()**, utilizados para resolver os sistemas triangulares inferior e superior,respectivamente.

```
y = substituicaoSucessiva(L, b)
```

```
x = substituicaoRetroativa(U, y)
```

Ao final retorno o vetor x , que é a solução do sistema LUx=b.

No método **substituicaoSucessiva(L,b)**, chamo a função **pivo()** para verificar cada coluna o pivô é diferente de zero e verificar se o pivô é o maior número da sua coluna, logo após pego a ordem da matriz A e atribuo a variável n e inicializo um vetor **x** com tamanho **n** e elementos iguais a 0.

```
n = len(A)
```

```
x = n * [0]
```

Seguindo no método faço um laço 1 que vai variar do primeiro elemento até o último.

```
for i in range(n-1, -1, -1):
```

Dentro do laço 1 atribuo a uma variável soma o valor de 0 para realizar o cálculo somatório.

```
soma = 0
```

Continuando faço outro laço 2 para fazer o somatório.

```
for j in range(i+1, n):
```

No laço 2 realizar o cálculo do somatório.

```
soma = soma + A[i][j] * x[j]
```

No laço 1 extraio os valores resultantes da matriz.

```
x[i] = (b[i] - soma) / A[i][i]
```

No final do método retorno o vetor solução para o método **substituicaoSucessiva()**.

No método **substituicaoRetroativa(U,y)**, chamo a função **pivo()** para verificar cada coluna o pivô é diferente de zero e verificar se o pivô é o maior número da sua coluna, logo após pego a ordem da matriz A e atribuo a variável n e inicializo um vetor **x** com tamanho **n** e elementos iguais a 0.

```
n = len(A)
```

```
x = n * [0]
```

Seguindo no método faço um laço 1 que vai variar do último elemento até o primeiro, resolvendo o sistema de baixo para cima extraindo os resultados.

```
for i in range(n-1, -1, -1):
```

Dentro do laço 1 atribuo a uma variável soma o valor de 0 para realizar o cálculo somatório.

```
soma = 0
```

Continuando faço outro laço 2 para fazer o somatório.

```
for j in range(i+1, n):
```

No laço 2 realizar o cálculo do somatório.

```
soma = soma + A[i][j] * x[j]
```

No laço 1 extraio os valores resultantes da matriz.

```
x[i] = (b[i] - soma) / A[i][i]
```

No final do método retorno o vetor solução para o método **substituicaoRetroativa()**.

```
return x
```

6.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. De início tive que obter uma forma de pegar os valores do arquivo de entrada e organizar em forma de matriz. Utilizei alguns métodos **readline()** para fazer a leitura da linha por linha , **replace('\n', ' ')** para fazer a remoção da quebra de linha e inserir um espaço

vazio no lugar e **split(‘ ‘)** para fazer a divisão dos valores de acordo com os espaços; todos os métodos nativos do python.

O arquivo texto de entrada está dividido na altura da matriz e nos seus elementos, respectivamente. O arquivo texto de saída é o vetor resultante.

Para fazer o cálculo de todos os testes terá que apagar a matriz já calculada do arquivo texto de entrada, para assim fazer a resolução de cada teste separadamente e rodar novamente o código para obter os resultados no arquivo texto de saída.

6.3 Problema teste 1, 2, 3…

- Problema 4.1 pág. 150 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

8I1 -4I2 -2I3 = 10

-4I1 +6I2 -2I3 = 0

-2I1 -2I2 +10I3 = 4

Organizando os dados de entrada no arquivo texto:

8 -4 -2 10

-4 6 -2 0

-2 -2 10 4

Dados de saída I1, I2 e I3, respectivamente:

[2.7586206896551726, 2.310344827586207, 1.4137931034482758]

Substituindo os dados de saída no sistema temos:

8*2.7586206896551726 - 4*2.310344827586207-2*1.4137931034482758 = 10

-4*2.7586206896551726 +6*2.310344827586207 -2*1.4137931034482758 = 0

-2*2.7586206896551726 -2*2.310344827586207+10*1.4137931034482758 = 4

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado nos retornou os valores perfeitamente.

-Problema 4.3 pág. 151 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

4p1 -1p2 +0 +0 -1p5 +0 +0 +0 = 0

-1p1 +4p2 -1p3 +0 +0 -1p6 +0 +0 = 0

0 -1p2 +4p3 -1p4 +0 +0 -1p7 +0 = 0

0 +0 -1p3 +4p4 +0 +0 +0 -1p8 = 0

-1p1 +0 +0 +0 +4p5 -1p6 +0 +0 = 1

0 -1p2 +0 +0 -1p5 +4p6 -1p7 +0 = 1

0 +0 -1p3 +0 +0 -1p6 +4p7 -1p8 = 1

0 +0 +0 -1p4 +0 +0 -1p7 +4p8 = 1

Organizando os dados de entrada no arquivo texto:

8

4 -1 0 0 -1 0 0 0 0

-1 4 -1 0 0 -1 0 0 0

0 -1 4 -1 0 0 -1 0 0

0 0 -1 4 0 0 0 -1 0

-1 0 0 0 4 -1 0 0 1

0 -1 0 0 -1 4 -1 0 1

0 0 -1 0 0 -1 4 -1 1

0 0 0 -1 0 0 -1 4 1

Dados de saída p1, p2, p3 , …,p8, respectivamente:

[0.16842105263157894, 0.2421052631578947, 0.2421052631578947, 0.16842105263157894, 0.43157894736842106, 0.5578947368421052, 0.5578947368421052, 0.43157894736842106]

Substituindo os dados de saída no sistema temos:

4*0.16842105263157894-1*0.2421052631578947+0+0-1*0.43157894736842106 +0 +0 +0 = 0

-1*0.16842105263157894 +4*0.242105263157894 -1*0.2421052631578947 +0 +0 -1*0.5578947368421052 +0 +0 = 0

0 -1*0.2421052631578947 +4*0.2421052631578947 -1*0.16842105263157894 +0 +0 -1*0.5578947368421052 +0 = 0

0 +0 -1*0.2421052631578947 +4*0.16842105263157894 +0 +0 +0 -1*0.43157894736842106 = 0

-1*0.16842105263157894 +0 +0 +0 +4*0.43157894736842106 -1*0.5578947368421052 +0 +0 = 1

0 -1*0.2421052631578947 +0 +0 -1*0.43157894736842106 +4*0.5578947368421052 -1*0.5578947368421052 +0 = 1

0 +0 -1*0.2421052631578947 +0 +0 -1*0.5578947368421052 +4*0.5578947368421052 -1*0.43157894736842106 = 1

0 +0 +0 -1*0.16842105263157894 +0 +0 -1*0.5578947368421052 +4*0.43157894736842106 = 1

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

-Problema 4.6 pág. 153 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

14I1 +4I2 +4Ix = 100

4I1 +7I2 +19Ix = 100

4I1 +7I2 +18Ix = 100

Organizando os dados de entrada no arquivo texto:

14 4 4 100

4 7 19 100

4 7 18 100

Dados de saída I1, I2 e Ix, respectivamente:

[3.658536585365854, 12.195121951219512, -0.0]

Substituindo os dados de saída no sistema temos:

14*3.658536585365854 +4*12.195121951219512 +4(-0.0) = 100

4*3.658536585365854 +7*12.195121951219512 +19*(-0.0) = 100

4*3.658536585365854 +7*12.195121951219512 +18*(-0.0) = 100

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado nos retornou os valores perfeitamente.

6.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

7. Método de Gauss-Jacobi

7.1 Estratégia de Implementação

De início implementei o método principal **jacobi_()** recebendo como parâmetro os vetores A e b, e(epsilon - precisão) e quantidade máxima de iterações.

```
def jacobi_(A, b, e, iter)
```

Dentro desse método pego a ordem da matriz A e atribuo a uma variável, no código a variável **n**, inicializo dois **x** e **v** vetores com **n** zeros.

```
n = len(A)
```

```
x = [0] * n
```

```
v = [0] * n
```

Seguindo no método faço laço 1 para dividir cada linha da matriz A e do vetor b por A[i][i].

```
for i in range(0,n):
```

Dentro do laço 1 faço outro laço 2.

```
for j in range(0,n):
```

No laço 2 faço uma condicional, para só dividir se o elemento for diferente diagonal principal.

```
if i ≠ j:
```

Caso seja diferente faço outra verificação para verificar o valor da diagonal principal, sendo diferente de 0 atribuo a posição A[i][j].

```
A[i][j] = A[i][j]/A[i][i]
```

Não sendo diferente de 0 imprimo na tela.

```
print("ERRO! Elemento da diagonal principal igual a 0.")
```

Ao final do laço 2 o vetor b recebe o seu valor dividido pelo elemento da diagonal principal da mesma linha.

```
b[i] = b[i] / A[i][i]
```

Ao final do laço 1 faço uma cópia do vetor b em no vetor x.

```
x = b[:]
```

Em seguida faço laços para calcular o somatório das linhas e o produto pelo resultado em x que é a aproximação anterior, armazenando em somatório.

```
for k in range(1, iter+1):
```

```
for i in range(0,n):
```

Atribuo 0 a variável **soma** para fazer o somatório.

```
soma = 0
```

Faço laço j para percorrer as colunas e calcular o somatório.

```
for j in range(0, n):
```

Dentro do laço j faço verificação para ignorar o elemento da diagonal principal.

```
if i ≠ j:
```

Faço o somatório do produto dos elementos da linha i pela coluna j pela aproximação anterior.

```
soma = soma + A[i][j] * x[j]
```

Ao final do laço j cálculo a aproximação atual.

```
v[i] = b[i] - soma
```

Seguindo chamo a função implementada **norma(v, x)** e atribuo a variável resultado, passando os vetores v e x como parâmetros. A função retorna a

divisão do maior numerador encontrado pelo maior denominador entre os vetores v e x.

```
resultado = norma(v, x)
```

Em seguida verifico se o valor retornado da função norma é menor ou igual a precisão. Se for menor retorno o vetor v, se não repete todo o processo armazenando em x o vetor v.

```
if resultado ⩽ e:
    return v
    x = v[:]
```

7.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. De início tive que obter uma forma de pegar os valores do arquivo de entrada e organizar em forma de matriz. Utilizei alguns métodos **readline()** para fazer a leitura da linha por linha , **replace('\n', ' ')** para fazer a remoção da quebra de linha e inserir um espaço vazio no lugar e **split(' ')** para fazer a divisão dos valores de acordo com os espaços; todos os métodos nativos do python.

O arquivo texto de entrada está dividido na altura da matriz, no valor da precisão e nos seus elementos, respectivamente. O arquivo texto de saída é o vetor resultante

Para fazer o cálculo de todos os testes terá que apagar a matriz já calculada do arquivo texto de entrada, para assim fazer a resolução de cada teste separadamente e rodar novamente o código para obter os resultados no arquivo texto de saída.

7.3 Problema teste 1, 2, 3…

-Problema 5.1 pág. 185 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

4x1 -1x2 +0 -1x4 +0 +0 = 100

-1x1 +4x2 -1x3 +0 -1x5 +0 = 0

0 -1x2 +4x3 +0 +0 -1x6 = 0

-1x1 +0 +0 +4x4 -1x5 +0 = 100

0 -1x2 +0 -1x4 +4x5 -1x6 = 0

0 +0 -1x3 +0 -1x5 +4x6 = 0

Organizando os dados de entrada no arquivo texto:

4 -1 0 -1 0 0 100

-1 4 -1 0 -1 0 0

0 -1 4 0 0 -1 0

-1 0 0 4 -1 0 100

0 -1 0 -1 4 -1 0

0 0 -1 0 -1 4 0

Dados de saída x1, x2, x3, x4, x5 e x6, respectivamente, para precisão 1E-15:

[38.095238095238024, 14.285714285714224, 4.761904761904735, 38.09523809523805, 14.28571428571425, 4.761904761904746]

Substituindo os dados de saída no sistema temos:

4*38.095238095238024 -1*14.285714285714224 +0 -1*38.09523809523805 +0 +0 = 99.99999999999983

-1*38.095238095238024 +4*14.285714285714224 -1*4.761904761904735 +0 -1*14.28571428571425 +0 = 0

0-1*14.285714285714224+4*4.761904761904735+0+0 -1*4.761904761904746 = 0

-1*38.095238095238024+0+0 +4*38.09523809523805 -1*14.28571428571425 +0 = 100

0 -1*14.285714285714224 +0 -1*38.09523809523805 +4*14.28571428571425 -1*4.761904761904746 = 0

0+0-1*4.761904761904735+0 -1*14.28571428571425 +4*4.761904761904746 = 0

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

-Problema 5.2 pág. 185 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

20i1 -10i2 -4i3 = 26

-10i1 +25i2 -5i3 = 0

-4i1 +5i2 +20i3 = 7

Organizando os dados de entrada no arquivo texto:

20 -10 -4 26

-10 25 -5 0

-4 5 20 7

Dados de saída i1, i2 e i3, respectivamente, para precisão 1E-15:

[1.814814814814813, 0.8271604938271597, 0.5061728395061726]

Substituindo os dados de saída no sistema temos:

20*1.814814814814813, -10*0.8271604938271597 -4*0.5061728395061726 = 26

-10*1.814814814814813 +25*0.8271604938271597 -5*0.5061728395061726 = 0

-4*1.814814814814813 +5*0.8271604938271597 +20*0.5061728395061726 = 7

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

-Problema 5.5 pág. 188 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

4u1 -1u2 +0 -1u4 +0 +0 +0 +0 +0 = 50

-1u1 +4u2 -1u3 +0 -1u5 +0 +0 +0 +0 = 50

0 -1u2 +4u3 +0 +0 -1u6 +0 +0 +0 = 150

-1u1 +0 +0 +4u4 -1u5 +0 -1u7 +0 +0 = 0

0 -1u2 +0 -1u4 +4u5 -1u6 +0 -1u8 +0 = 0

0 +0 -1u3 +0 -1u5 +4u6 +0 +0 -1u9 = 100

0 +0 +0 +0 -1u5 +0 +4u7 -1u8 +0 = 50

Organizando os dados de entrada no arquivo texto:

4 -1 0 -1 0 0 0 0 0 50

-1 4 -1 0 -1 0 0 0 0 50

0 -1 4 0 0 -1 0 0 0 150

-1 0 0 4 -1 0 -1 0 0 0

0 -1 0 -1 4 -1 0 -1 0 0

0 0 -1 0 -1 4 0 0 -1 100

0 0 0 0 -1 0 4 -1 0 50

0 0 0 0 -1 0 -1 4 -1 50

0 0 0 0 0 -1 0 -1 4 150

Dados de saída: u1, u2, u3,... e u9, respectivamente, para precisão 1E-15:

**[32.78940886699487, 50.55418719211802, 68.13423645320186,
30.603448275861865, 51.29310344827566, 71.98275862068955,
38.33128078817724, 52.032019704433395, 68.50369458128074]**

Substituindo os dados de saída no sistema temos:

**4*32.78940886699487 -1*50.55418719211802 +0 -1*30.603448275861865 +0
+0 +0 +0 +0 = 50**

**-1*32.78940886699487 +4*50.55418719211802 -1*68.13423645320186 +0
-1*51.29310344827566 +0 +0 +0 +0 = 50**

**0 -1*50.55418719211802 +4*68.13423645320186 +0 +0
-1*71.98275862068955 +0 +0 +0 = 150**

**-1*32.78940886699487 +0 +0 +4*30.603448275861865
-1*51.29310344827566 +0 -1*38.33128078817724 +0 +0 = 0**

**0 -1*50.55418719211802 +0 -1*30.603448275861865 +4*51.29310344827566
-1*71.98275862068955 +0 -1*52.032019704433395 +0 = 0**

**0 +0 -1*68.13423645320186 +0 -1*51.29310344827566
+4*71.98275862068955 +0 +0 -1*68.50369458128074 = 100**

**0 +0 +0 +0 -1*51.29310344827566 +0 +4*38.33128078817724
-1*52.032019704433395 +0 = 50**

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

7.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

8. Método de Gauss-Seidel

8.1 Estratégia de Implementação

De início implementei o método principal **seidel()** recebendo como parâmetro os vetores A e b, e(epsilon - precisão) e quantidade máxima de iterações.

```
def seidel(A, b, e, iter)
```

Dentro desse método pego a ordem da matriz A e atribuo a uma variável, no código a variável **n**, inicializo dois **x** e **v** vetores com **n** zeros.

```
n = len(A)
```

```
x = [0] * n
```

```
v = [0] * n
```

Seguindo no método faço laço 1 para dividir cada linha da matriz A e do vetor b por A[i][i].

```
for i in range(0,n):
```

Dentro do laço 1 faço outro laço 2.

```
for j in range(0,n):
```

No laço 2 faço uma condicional, para só dividir se o elemento for diferente diagonal principal.

```
if i ≠ j:
```

Caso seja diferente faço outra verificação para verificar o valor da diagonal principal, sendo diferente de 0 atribuo a posição A[i][j].

```
A[i][j] = A[i][j]/A[i][i]
```

Não sendo diferente de 0 imprimo na tela.

```
print("ERRO! Elemento da diagonal principal igual a 0.")
```

Ao final do laço 2 o vetor b recebe o seu valor dividido pelo elemento da diagonal principal da mesma linha.

```
b[i] = b[i] / A[i][i]
```

Em seguida faço laços para calcular o somatório das linhas e o produto pelo resultado em x que é a aproximação anterior, armazenando em somatório.

```
for k in range(1, iter+1):
```

```
for i in range(0,n):
```

Atribuo 0 a variável **soma** para fazer o somatório.

```
soma = 0
```

Faço laço j para percorrer as colunas e calcular o somatório.

```
for j in range(0, n):
```

Dentro do laço j faço verificação para ignorar o elemento da diagonal principal.

```
if i ≠ j:
```

Faço o somatório do produto dos elementos da linha i pela coluna j pela aproximação anterior.

```
soma = soma + A[i][j] * x[j]
```

Ao final do laço j cálculo a aproximação atual.

```
x[i] = b[i] - soma
```

Seguindo chamo a função implementada **norma(v, x)** e atribuo a variável resultado, passando os vetores x e v como parâmetros, que são o vetor atual e o anterior, respectivamente. A função retorna a divisão do maior numerador encontrado pelo maior denominador entre os vetores x e v.

```
resultado = norma(x, v)
```

Em seguida verifico se o valor retornado da função norma é menor ou igual a precisão. Se for menor retorno o vetor x, se não repete todo o processo armazenando em v o vetor x.

```
if resultado ⩽ e:
    return x
    v= x[:]
```

8.2 Estrutura dos Arquivos de Entrada/Saída

O formato do arquivo escolhido foi o .txt, formato de arquivo texto. A escolha foi justamente pela facilidade de organização dos dados. De início tive que obter uma forma de pegar os valores do arquivo de entrada e organizar em forma de matriz. Utilizei alguns métodos **readline()** para fazer a leitura da linha por linha , **replace('\n', ' ')** para fazer a remoção da quebra de linha e inserir um espaço vazio no lugar e **split(' ')** para fazer a divisão dos valores de acordo com os espaços; todos os métodos nativos do python.

O arquivo texto de entrada está dividido na altura da matriz, no valor da precisão e nos seus elementos, respectivamente. O arquivo texto de saída é o vetor resultante

Para fazer o cálculo de todos os testes terá que apagar a matriz já calculada do arquivo texto de entrada, para assim fazer a resolução de cada teste separadamente e rodar novamente o código para obter os resultados no arquivo texto de saída.

8.3 Problema teste 1, 2, 3…

-Problema 5.1 pág. 185 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

4x1 -1x2 +0 -1x4 +0 +0 = 100

-1x1 +4x2 -1x3 +0 -1x5 +0 = 0

0 -1x2 +4x3 +0 +0 -1x6 = 0

-1x1 +0 +0 +4x4 -1x5 +0 = 100

0 -1x2 +0 -1x4 +4x5 -1x6 = 0

0 +0 -1x3 +0 -1x5 +4x6 = 0

Organizando os dados de entrada no arquivo texto:

4 -1 0 -1 0 0 100

-1 4 -1 0 -1 0 0

0 -1 4 0 0 -1 0

-1 0 0 4 -1 0 100

0 -1 0 -1 4 -1 0

0 0 -1 0 -1 4 0

Dados de saída x1, x2, x3, x4, x5 e x6, respectivamente, para precisão 1E-15:

[38.095238095238, 14.28571428571421, 4.76190476190473, 38.095238095238045, 14.28571428571424, 4.761904761904742]

Substituindo os dados de saída no sistema temos:

4*38.095238095238 -1*14.28571428571421 +0 -1*38.095238095238045 +0 +0 = 100

-1*38.095238095238 +4*14.28571428571421 -1*4.76190476190473 +0 -1*14.28571428571424 +0 = 0

0 -1*14.28571428571421 +4*4.76190476190473 +0 +0 -1*4.761904761904742 = 0

-1*38.095238095238 +0 +0 +4*38.095238095238045 -1*14.28571428571424 +0 = 100

0 -1*14.28571428571421 +0 -1*38.095238095238045 +4*14.28571428571424 -1*4.761904761904742 = 0

0 +0 -1*4.76190476190473 +0 -1*14.28571428571424 +4*4.761904761904742 = 0

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

-Problema 5.2 pág. 185 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

20i1 -10i2 -4i3 = 26

-10i1 +25i2 -5i3 = 0

-4i1 +5i2 +20i3 = 7

Organizando os dados de entrada no arquivo texto:

20 -10 -4 26

-10 25 -5 0

-4 5 20 7

Dados de saída i1, i2 e i3, respectivamente, para precisão 1E-15:

[1.8148148148148127, 0.8271604938271595, 0.5061728395061726]

Substituindo os dados de saída no sistema temos:

20*1.8148148148148127 -10*0.8271604938271595 -4*0.5061728395061726 = 26

-10*1.8148148148148127 +25*0.8271604938271595 -5*0.5061728395061726 = 0

-4*1.8148148148148127 +5*0.8271604938271595 +20*0.5061728395061726 = 7

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

-Problema 5.5 pág. 188 Cálculo Numérico (Neide Franco)

Do problema temos o sistema:

4u1 -1u2 +0 -1u4 +0 +0 +0 +0 +0 = 50

-1u1 +4u2 -1u3 +0 -1u5 +0 +0 +0 +0 = 50

0 -1u2 +4u3 +0 +0 -1u6 +0 +0 +0 = 150

-1u1 +0 +0 +4u4 -1u5 +0 -1u7 +0 +0 = 0

0 -1u2 +0 -1u4 +4u5 -1u6 +0 -1u8 +0 = 0

0 +0 -1u3 +0 -1u5 +4u6 +0 +0 -1u9 = 100

0 +0 +0 +0 -1u5 +0 +4u7 -1u8 +0 = 50

Organizando os dados de entrada no arquivo texto:

4 -1 0 -1 0 0 0 0 0 50

-1 4 -1 0 -1 0 0 0 0 50

0 -1 4 0 0 -1 0 0 0 150

-1 0 0 4 -1 0 -1 0 0 0

0 -1 0 -1 4 -1 0 -1 0 0

0 0 -1 0 -1 4 0 0 -1 100

0 0 0 0 -1 0 4 -1 0 50

0 0 0 0 -1 0 -1 4 -1 50

0 0 0 0 0 -1 0 -1 4 150

Dados de saída: u1, u2, u3,... e u9, respectivamente, para precisão 1E-15:

[32.78940886699486, 50.55418719211802, 68.13423645320186, 30.603448275861858, 51.29310344827566, 71.98275862068955, 38.33128078817724, 52.032019704433395, 68.50369458128074]

Substituindo os dados de saída no sistema temos:

4*32.78940886699486 -1*50.55418719211802 +0 -1*30.603448275861858 +0 +0 +0 +0 +0 = 50

-1*32.78940886699486 +4*50.55418719211802 -1*68.13423645320186 +0 -1*51.29310344827566 +0 +0 +0 +0 = 50

0 -1*50.55418719211802 +4*68.13423645320186 +0 +0 -1*71.98275862068955 +0 +0 +0 = 150

-1*32.78940886699486 +0 +0 +4*30.603448275861858 -1*51.29310344827566 +0 -1*38.33128078817724 +0 +0 = 0

0 -1*50.55418719211802 +0 -1*30.603448275861858 +4*51.29310344827566 -1*71.98275862068955 +0 -1*52.032019704433395 +0 = 0

0 +0 -1*68.13423645320186 +0 -1*51.29310344827566 +4*71.98275862068955 +0 +0 -1*68.50369458128074 = 100

0 +0 +0 +0 -1*51.29310344827566 +0 +4*38.33128078817724 -1*52.032019704433395 +0 = 50

Análise dos resultados:

Com os resultados obtidos podemos ver que o método implementado teve bom retorno.

8.4 Dificuldades enfrentadas

A dificuldade encontrada foi só a parte de entendimento do método para a implementação do código na linguagem de programação, mas após pesquisas conseguir sanar as dúvidas.

Considerações Finais

Para rodar os códigos implementados necessita ter na máquina o python na versão 3.

Instruções para instalação do python 3 no windows:

Acesse: https://python.org.br/instalacao-windows/

Faço o download do instalador do python 3, com base na sua arquitetura 32 ou 64 bits.

Clique duas vezes no executável que foi baixado, faça o seguinte:

1. Marque a opção "Add Python to PATH"
2. Clique em "Install Now"

Abra o terminal e verifique se o python foi instalado:

python --version

Execute também o comando abaixo para verificar se o pip foi instalado, sendo ele o gerenciador de pacotes do python:

pip --version

Instalar as bibliotecas numpy e sympy.

Execute no seu terminal o seguinte comando:

pip install numpy

pip install sympy

Após a instalação, está tudo pronto para execução dos códigos,

Execução dos códigos com os testes

Para realizar os testes necessita que o arquivo ‘input.txt’ esteja no mesmo diretório que o método, executando o método a solução estará no arquivo de saída ‘output.txt’.