Universidade Estadual de Santa Cruz – UESC

Relatório de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por Levy Marlon Souza Santiago

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2017.2

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2017

ÍNDICE

Linguagem(ns) Escolhida(s) e justificativas

A linguagem escolhida foi o Python, pois além de ser uma linguagem em que o autor deste relatório já é familiarizado, existem algumas bibliotecas para cálculos matemáticos e manipulação de funções que já estão implementadas para se utilizar nesta linguagem. A versão do python utilizada foi a versão 3.5.3. O Sistema Operacional usado para implementar os métodos foi o Linux distribuição Ubuntu 17.04.

Especificações Arquivo de Entrada

Biblioteca usada:

Para todos os métodos foi necessário usar uma biblioteca chamada sympy. Para instalar esta biblioteca no Python3, foi utilizado o seguinte comando no terminal Linux: $ sudo pip3 install sympy. Foi preferível usar esta biblioteca para todos os métodos porque além de implementar funções para cálculos matemáticos (exponencial, seno, cosseno...), também implementa manipulação de funções e equações (f(x) = x + y + z, x + y = 1...). Por isso foi aberta esta nova sessão para explicar como são representadas as funções matemáticas a partir desta biblioteca.

Também foi utilizada em todos os métodos uma biblioteca chamada sys. Esta biblioteca permite usar argumentos em python, e ela foi usada para inserir como argumento do programa o caminho do arquivo de entrada.

Para poder executar qualquer método, deve-se digitar pyhon3 (ou python, caso a versão 3 esteja configurada como padrão na máquina), o nome do arquivo .py e depois o nome do arquivo de entrada como parâmetro.

$ python3 nomeDoMetodo.py entrada.txt

Algumas representações

Abaixo se encontram as representações de algumas funções matemáticas que podem ser usadas no arquivo de entrada:

- x^2 = pow(x, 2) ou x**2;
- Raiz quadrada de x = sqrt(x);
- Seno de x = sin(x);
- Arcoseno de x = asin(x)
- Cosseno de x = cos(x);
- Tangente de x = tan(x);
- Número de Euler (e) elevado a x = exp(x);
- Log de x na base y = log(x, y);
- pi é uma constante já definida (3,1415…..);

Observações

Abaixo se encontram algumas observações que é preciso se atentar ao gerar o arquivo de entrada:

- O programa não reconhece 5x, mas sim 5*x;

Método de Bissecção

Estratégia de Implementação:

Como explicado na seção de especificação do arquivo de entrada, foi usada a biblioteca sympy para usar funções em python e a biblioteca sys para usar argumentos.

O critério de parada do método, considerando ‘c’ o meio do intervalo atual, é se f(c) for <= precisão. Caso o valor absoluto do f(início do intervalo) seja maior do que a precisão ou o mesmo para o f(fim do intervalo), então o método retorna None e o programa imprime no arquivo de saída: “Verifique se os intervalos estão corretos.”. Caso a resposta final testada na função resultar em um valor maior que 1 o programa imprime “Resultado maior que 1” e mostra o resultado, para chamar atenção que existe algo errado na resposta, já que a resposta esperada está entre 0 e 1.

Um fato importante é que esse método só aceita a variável ‘x’ para o uso em uma função de entrada. Pois é o símbolo que foi definido no programa.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada do método em questão foi organizado na seguinte ordem: Função, Início do intervalo, Fim do intervalo e Precisão. O arquivo pode conter outras entradas de novas funções seguindo esta mesma ordem, lembrando que cada entrada deve ser separada por um espaço (Enter). Exemplo:

pow(x,2) + 3*x - 2

0

1

0.1

0.25*(x-2) + 0.1*sin(x)

1

2

0.1

O arquivo de saída é gerado informando primeiramente a função dada, o valor da raiz encontrada, e o valor do f(raiz). Sequencialmente, são informados as respostas das outras entradas, caso houveram outras.

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

pow(x + 1, 2) * exp(pow(x, 2) - 2) - 1

0

1

0.1

Resultado:

F(x) = pow(x + 1, 2) * exp(pow(x, 2) - 2) - 1

Raiz ~= 0.875

F(Raiz) = 0.0231052024173779

Problema 2:

pow(x,2) + 3*x - 2

0

1

0.1

Resultado:

F(x) = pow(x,2) + 3*x - 2

Raiz ~= 0.5625

F(Raiz) = 0.00390625

Problema 3:

0.25*(x-2) + 0.1*sin(x)

1

2

0.1

Resultado:

F(x) = 0.25*(x-2) + 0.1*sin(x)

Raiz ~= 1.5

F(Raiz) = 0.0252505013395946

Dificuldades enfrentadas

Uma dificuldade encontrada foi saber como transformar uma função em forma de string recebida pelo arquivo e transformá-la em uma função que o python saiba manipular. Neste método, este problema foi resolvido usando a função eval() do próprio python. Esta função transforma uma string em um número, por exemplo: eval(“2”) = 2. Como neste método usamos função somente para calcular o f(x), então foi criada uma função que recebe o valor de x, e depois calculado e retornado o eval da função recebida do arquivo, já que o valor de x foi recebido.

Método da Posição Falsa

Estratégia de Implementação:

A estratégia de implementação deste método é exatamente a mesma estratégia do método de Bissecção. A única diferença em relação à implementação é que o meio do intervalo é calculado a partir de uma fórmula que na verdade é a característica do método da Posição Falsa. Para o cálculo desta fórmula, foi criada uma função separada chamada “calculaMeio”. E dentro da função do método, ao invés de o meio ser calculado a partir da metade da soma do início e fim do intervalo, este foi calculado chamando a função “calculaMeio”.

Neste método também está definido o ‘x’ como variável única para o uso nas funções de entrada.

Estrutura dos Arquivos de Entrada/Saída

A estrutura tanto do arquivo de entrada como do arquivo de saida é a mesma do que o método anterior.

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

pow(x + 1, 2) * exp(pow(x, 2) - 2) - 1

0

1

0.1

Resultado:

F(x) = pow(x + 1, 2) * exp(pow(x, 2) - 2) - 1

Raiz ~= 0.856495517392990

F(Raiz) = 0.0286168239637636

Problema 2:

pow(x,2) + 3*x - 2

0

1

0.1

Resultado:

F(x) = pow(x,2) + 3*x - 2

Raiz ~= 0.5555555555555556

F(Raiz) = 0.024691358024691246

Problema 3:

0.25*(x-2) + 0.1*sin(x)

1

2

0.1

Resultado:

F(x) = 0.25*(x-2) + 0.1*sin(x)

Raiz ~= 1.64588828436895

F(Raiz) = 0.0111902634464286

Dificuldades enfrentadas

Como este método é quase a mesma implementação do método anterior, então não houve nenhuma dificuldade ao implementá-lo.

Método do Ponto Fixo

Estratégia de Implementação:

O método já recebe todas as possibilidades de isolamento da variável. A partir de cada equação dada, é verificado se a equação tem raiz, se conter raiz, o programa avisa no arquivo de saída que pode ser que a função não convirja, pois pode resultar em um número imaginário. Se não, ele calcula a raiz normalmente usando o método. A condição de parada do método é se o valor absoluto da diferença entre 'x' e o seu valor anterior for <= precisão. A impressão no arquivo de saída foi feita dentro do próprio método, porque o programa imprime o resultado de cada equação dada. Lembrando que a variável definida para uso nas funções de entrada é a variável 'x'.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada tem a seguinte ordem: primeiro a precisão, depois o valor inicial para a iteração e por fim, as equações resultantes dos isolamentos. Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa a função e o resultado encontrado. Para cada entrada existe um resultado e os resultados são separados por um enter ('\n').

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

0.1

0.5

(2 - 3*x)/x

(2 - pow(x, 2))/3

sqrt(2 - 3*x)

Resultado:

Equacao: (2 - 3*x)/x

Valor: 3.557377049180328

Equacao: (2 - pow(x, 2))/3

Valor: 0.5833333333333334

Função com raiz pode não convergir.

Problema 2:

0.1

0.25

pow(x, 2)

sqrt(2 + x)

Resultado:

Equacao: pow(x, 2)

Valor: 0.00390625

Função com raiz pode não convergir.

Problema 3:

0.1

0.15

sin(x) - pi

asin(x + pi)

Resultado:

Equacao: sin(x) - pi

Valor: 3.191579044457225

Equacao: asin(x + pi)

Valor: 2.49314807552479

Dificuldades enfrentadas

Uma dificuldade encontrada foi o fato dar erro quando o programa encontrava uma equação com raiz (sqrt). Mas o problema foi resolvido simplesmente inserindo um ‘if’ no código, então toda vez que o programa encontra um ‘sqrt’ na equação de entrada (string), então ele imprime uma mensagem no arquivo de saída dizendo que se a equação tem uma raiz, então esta pode não convergir, logo a função geral não converge.

Método de Newton Raphson

Estratégia de Implementação:

O método utiliza a variável ‘x’ como único símbolo. Ele recebe como entrada a precisão, o valor inicial para o início da iteração e a função. Para o cálculo da derivada, foi usado uma função da biblioteca sympy chamada diff. E depois de definir um valor inicial, foram calculadas os próximos valores de ‘x’ em função do seu valor anterior. A condição de parada foi se o valor absoluto da diferença entre x e o seu valor anterior for <= à precisão. Depois do cálculo de uma entrada, o resultado é impresso no arquivo de saída e depois é lida uma outra entrada se existir.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada tem a seguinte ordem: primeiro a função, depois a precisão e por fim, o valor inicial para a iteração. Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa a função e o resultado encontrado. Para cada entrada existe um resultado e os resultados são separados por um enter (‘\n’).

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

4*cos(x) - exp(x)

0.01

1.0

Resultado:

F(x) = 4*cos(x) - exp(x)

Resultado: 0.904794061672367

Problema 2:

pow(x,5) - 500

0.1

4

Resultado:

F(x) = pow(x,5) - 500

Resultado: 3.46576466455572

Problema 3:

pow(x,2) + 3*x - 2

0.1

0.5

Resultado:

F(x) = pow(x,2) + 3*x - 2

Resultado: 0.561553030303030

Dificuldades enfrentadas

Uma dificuldade enfrentada foi saber como calcular a derivada de uma função. Mas depois de verificar a documentação da biblioteca sympy, a função diff foi encontrada.

Método da Secante

Estratégia de Implementação:

Nesse método basicamente foi realizada a mesma ideia. A única diferença é que o valor de 'x' é calculado de acordo com a fórmula do método em questão. Como essa fórmula para o x1 não é aplicável, então o x1 é recebido pelo arquivo de entrada assim como o x0, a partir do x2 foi usado o método da Secante. A condição de parada foi se o valor absoluto da diferença entre x e o seu valor anterior for <= à precisão. Assim como o método anterior, depois do cálculo de uma entrada, o resultado é impresso no arquivo de saída e depois é lida uma outra entrada se existir.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada tem a seguinte ordem: primeiro a função, depois a precisão, então o valor inicial e por fim, o segundo valor inicial para a iteração. Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa a função e o resultado encontrado. Para cada entrada existe um resultado e os resultados são separados por um enter ('\n').

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

tan(x)

0.001

pi/2

(3*pi)/2

Resultado:

F(x) = tan(x)

Resultado: 6.28318530717959

Problema 2:

sqrt(x) - 5*exp(-x)

0.01

1.4

1.5

Resultado:

F(x) = sqrt(x) - 5*exp(-x)

Resultado: 1.43041940822215

Problema 3:

pow(x,5) - 500

0.2

4

5

Resultado:

F(x) = pow(x,5) - 500

Resultado: 3.6236188905493765

Dificuldades enfrentadas

Como este método é quase a mesma implementação do método anterior, então não houve nenhuma dificuldade ao implementá-lo.

Método da Eliminação de Gauss

Estratégia de Implementação:

Este método recebe como entrada as equações do sistema, o vetor b, que são os valores após a igualdade e um vetor de todas variáveis usadas no sistema. Primeiramente uma matriz é gerada com todos os coeficientes de cada equação do sistema em que o tamanho da matriz é a quantidade de equações. Para retirar os coeficientes de cada equação, esta equação é tratada como equação polinomial a partir da função Poly do sympy e então é possível receber uma lista contendo todos os coeficientes da equação com o método .coeffs(). Um fato importante é que o programa resolve equações com até cinco variáveis. Isto porque os símbolos precisam estar definidos, então foi definido no início quais são as variáveis que podem ser usadas, e as variáveis são: x, y, z, w e g.

Para transformar a matriz em uma matriz triangular, foi feito exatamente a ideia do método. Definindo o pivô e se este for diferente de zero, então é encontrado o multiplicador e depois este é usado para zerar os elementos abaixo dos elementos da diagonal, se o pivô é zero, então é feita uma troca com a linha de baixo. Lembrando que o vetor b também é atualizado, porém separadamente da matriz, ou seja, existe um vetor que guarda todos os elementos do vetor b.

No final, somente é feita a substituição de cada linha da matriz como sendo os coeficientes de cada equação, lembrando que as iterações começaram do fim da matriz, pois assim temos somente um coeficiente não zerado. As equações são na verdade reconstruídas de acordo com as linhas da matriz multiplicadas pelo vetor de variáveis. O resultado foi guardado em um vetor chamado results. Para cada resultado encontrado, o vetor de variáveis é atualizado com o valor da variável, assim, na próxima iteração o valor da variável é usado na reconstrução da equação.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada recebe o sistema a ser resolvido. Exemplo de uma entrada:

x + y + z = 1

x - y - z = 1

2*x + 3*y - 4*z = 9

Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa os valores de cada variável no fim. Para cada entrada existe um resultado e os resultados são separados por um enter (‘\n’).

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

6*x + 2*y -z = 7

2*x + 4*y + z = 7

3*x + 2*y + 8*z = 13

Resultado:

(z, 1)

(y, 1)

(x, 1)

Problema 2:

3*x + 3*y + z = 7

2*x + 2*y - z = 3

x - y + 5*z = 5

Resultado:

(z, 1)
(y, 1)
(x, 1)

Problema 3:

x + y + z = 1

x - y - z = 1

2*x + 3*y - 4*z = 9

Resultado:

(z, -1)

(y, 1)

(x, 1)

Dificuldades enfrentadas

Uma dificuldade enfrentada foi saber como usar o valor encontrado de uma variável e usar em uma outra iteração. Mas isto foi resolvido atualizando o vetor de variáveis mudando uma variável por seu valor encontrado, e usando o vetor de variáveis para a reconstrução das outras equações.

Método Fatoração LU

Estratégia de Implementação:

Neste método foi feito a mesma ideia do método anterior para criar a matriz dos coeficientes de cada equação do sistema. Posteriormente foi criada a matriz LU. A matriz LU é criada inicialmente zerada. Esta matriz vai conter tanto a matriz L quanto a matriz U. A ideia para fazer as duas matrizes em uma só foi a seguinte: Foram usados dois contadores (‘i’ e ‘j’) que vão até o fim do tamanho da matriz. Para calcular a parte U, foi usada a fórmula do próprio método usada para obter os elementos da matriz U. Para calcular a parte L, como na parte U o ‘i’ é a linha e o ‘j’ é a coluna, na parte L foi feito o contrário, porque assim atingimos a parte “inversa” da parte U, ou seja, usando o ‘j’ como linha e o ‘i’ como coluna. Lembrando que na parte U o ‘j’ andou sempre do ‘i’ até o fim e na parte L de ‘i+1’ até o fim. Isso porque na parte U vamos da diagonal até a última coluna. Já na parte L vamos sempre começar da linha ‘i+1’, pois na matriz L a diagonal é sempre 1, então não é preciso começar desde a diagonal, até porque iria alterar o valor de U na diagonal.

Seguindo a lógica, para a multiplicação das matrizes, foi criado um vetor multiplicador inicialmente zerado. Esse vetor vai pegar cada linha da matriz L e multiplicar pelo vetor auxiliar que contém a mesma quantidade de variáveis da equação, porém com outros símbolos (a, b, c, d, e). No momento da multiplicação, o vetor multiplicador recebe uma linha da matriz e essa linha é multiplicada pelo vetor auxiliar de variáveis, isso vai resultar em uma equação, que tem coeficiente, variável e um valor de igualdade que é um elemento do vetor b. Nesta equação é utilizada uma função do sympy chamada solve_linear, que é usada nesse caso para isolar a variável que está na equação. Depois esse resultado é guardado para usar na próxima iteração e também armazenado no vetor b. No fim, teremos um resultado no vetor b que será a multiplicação da matriz L por um vetor auxiliar.

Para multiplicar a matriz U pelo vetor de variáveis original, foi usada a mesma ideia, uma diferença é que a leitura da matriz U foi da última linha, que contém somente um coeficiente diferente de zero, para a primeira. Esse resultado foi guardado no vetor b e este foi retornado. Importante lembrar que assim como o método anterior, o máximo de variáveis que um sistema pode ter neste programa é 5. E as variáveis que podem ser usadas são: ‘x’, ‘y’, ‘z’,’ w’ e ‘g’

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada recebe o sistema a ser resolvido. Exemplo de uma entrada:

x + y + z = 1

x - y - z = 1

2*x + 3*y - 4*z = 9

Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa os valores de cada variável no fim. Para cada entrada existe um resultado e os resultados são separados por um enter (‘\n’).

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

x + 4*y - z = 6

2*x - y + 2*z = 3

-x + 3*y + z = 5

Resultado:

(x, 7/6)

(y, 7/3)

(z, 9/2)

Problema 2:

x + y = 5

x - y = -7

Resultado:

(x, -1)

(y, 6)

Problema 3:

x + y + z = 1

x - y - z = 1

2*x + 3*y - 4*z = 9

Resultado:

(x, 1)

(y, 1)

(z, -1)

Dificuldades enfrentadas

Uma dificuldade enfrentada foi tratar as duas matrizes L e U como uma matriz só. Mas foi resolvido da forma que foi explicada no primeiro tópico desse método.

Método de Jacobi

Estratégia de Implementação:

Na implementação deste método, a precisão é a quantidade de iterações que a entrada precisa. Primeiramente é criado o vetor resultado e iniciado com zero. Logo depois é investigado se o sistema realmente converge a partir da soma dos valores fora da diagonal principal da matriz em relação aos valores da diagonal principal. Se a soma dos valores absolutos dos elementos fora da diagonal principal for maior do que o valor absoluto do elemento da diagonal principal, o sistema não converge. Se o sistema converge, então primeiramente, para cada equação é isolada uma variável usando o solve_linear do sympy. Depois que estão todas as variáveis isoladas, é usada uma cópia do vetor resultado para substituir os valores iniciais (o primeiro valor é 0,0,0…,0) e então atribuir o resultado ao vetor resultado. Então esse processo se repete, fazendo uma outra cópia do vetor resultado com os novos valores e fazendo o cálculo com estes novos valores, assim por diante. No final, o vetor resultado vai estar com os resultados finais do sistema. Lembrando que esse loop vai de 0 até a precisão, como foi explicado no início. Lembrando que o limite de cálculo vai até sistemas com 5 variáveis (x, y, z, w, g).

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada recebe primeiro a precisão, e depois o sistema a ser resolvido. Exemplo de uma entrada:

5

10*x + 2*y + z = 7

x + 5*y + z = -8

2*x + 3*y + 10*z = 6

Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa os valores do sistema no fim. Para cada entrada existe um resultado e os resultados são separados por um enter (‘\n’).

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

3

10*x + y - z = 10

x + 10*y + z = 12

2*x - y + 10*z = 11

Resultado:

1.0

1.0

1.0

Problema 2:

5

10*x + 2*y + z = 7

x + 5*y + z = -8

2*x + 3*y + 10*z = 6

Resultado:

1.0

-2.0

1.0

Problema 3:

4

x + y + z = 1

x - y - z = 1

2*x + 3*y - 4*z = 9

Resultado:

Não converge.

Dificuldades enfrentadas

Uma dificuldade foi saber como tratar uma equação lida como uma equação mesmo, ou seja, com um valor antes do igual e com outro valor depois. Mas isso foi resolvido usando a função Eq() do sympy. O ‘Eq’ é definido assim: Eq(Valor antes do igual, Valor depois do igual). Tratando a equação como um ‘Eq’, a função solve_linear funciona da mesma forma.

Método de Gauss Seidel

Estratégia de Implementação:

A implementação deste método é quase a mesma que o método anterior, a única mudança é que ao invés de fazer uma cópia do vetor resultado, usamos o próprio vetor resultado para os cálculos. Assim, mesmo que estiver em uma mesma iteração, quando o valor de x1, por exemplo, for alterado, então este valor já será usado no cálculo de x2 e x3.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada recebe primeiro a precisão, e depois o sistema a ser resolvido. Exemplo de uma entrada:

5

10*x + 2*y + z = 7

x + 5*y + z = -8

2*x + 3*y + 10*z = 6

Lembrando que podem ser inseridas outras entradas abaixo desta, seguindo a mesma ordem, somente separadas por um Enter. O arquivo de saída informa os valores do sistema no fim. Para cada entrada existe um resultado e os resultados são separados por um enter (‘\n’).

Problema teste 1, 2, 3...

Abaixo estão três entradas que foram testadas neste método e seus respectivos resultados:

Problema 1:

3

10*x + y - z = 10

x + 10*y + z = 12

2*x - y + 10*z = 11

Resultado:

1.0

1.0

1.0

Problema 2:

5

10*x + 2*y + z = 7

x + 5*y + z = -8

2*x + 3*y + 10*z = 6

Resultado:

1.0

-2.0

1.0

Problema 3:

4

x + y + z = 1

x - y - z = 1

2*x + 3*y - 4*z = 9

Resultado:

Não converge.

Dificuldades enfrentadas

Como este método é quase a mesma implementação do método anterior, então não houve nenhuma dificuldade ao implementá-lo.

Considerações Finais

É importante lembrar que as implementações destes métodos não estão completamente revisadas e testadas, por isso, podem haver alguns casos que gerem algum problema. Mas por fim, grande parte dos problemas podem ser resolvidos com estas implementações.