Calculo Numérico
Trabalho_4

Prof. Dra. Larissa de Freitas [1],
Guilherme de Souza[1]

1

1. Métodos implementados

- Trapezio;
- 1/3 de Simpsom;
- 3/8 de Simpsom ;
- Euler ???;
- Runge-Kutta 2a Ordem;
- Runge-Kutta 4a Ordem;
- Adams;

Para execução de tais métodos como, **Trapezio, 1/3 de Simpsom e 3/8 de Simpsom** foram aplicados na lista de exercícios 11[1]. Os métodos de **Runge-Kutta 2a ordem**, **4a ordem** e **Adams** utilizou-se a lista de exercícios 12[2].

2. Método do Trapezio

- **(A)** Trapezio: 31.365285650063754

-1.08268227	-0.73575888	0.0	5.43656366	59.11244879

Table 1. Função utlizando a regra do trapezio no exercício A da lista 11

- **(B)** Trapezio: 0.7842407666178157

0.5	0.97297297	0.9	0.8	0.69230769	0.59016393	0.25

Table 2. Função utlizando a regra do trapezio no exercício B da lista 11

- **(C)** Trapezio: 0.10486282062502501

0.0	0.20999043	0.69856645	1.08060461	0.83640026	-0.53179749	-1.66458735

Table 3. Função utlizando a regra do trapezio no exercício C da lista 11

- **(D)** Trapezio: -13.575979391799388

- **(E)** Trapezio: 0.5196110146984233

[1]Disponivel no AVA:https://ava.ufpel.edu.br/pre/pluginfile.php/318455/mod_resource/content/1/ListaDeExercicios11.pdf

[2]Disponivel no AVA:https://ava.ufpel.edu.br/pre/pluginfile.php/320366/mod_resource/content/2/ListaDeExercicios12.pdf

0.0	2.24766465	5.42294499	6.97416428	2.08548731	-13.92608463	-39.26843454	-56.88800

Table 4. Função utlizando a regra do trapezio no exercício D da lista 11

0.4	0.71910112	0.64	0.56637168	0.5	0.44137931	0.3902439	0.34594595	0.15384615

Table 5. Função utlizando a regra do trapezio no exercício D da lista 11

3. Fatoração LU

Neste métodos realizamos uma operação um pouco diferente da multiplicação de matrizes. Antes tinhamos $AB=C$ agora realização a fatoração da mesma, ou seja, $C=AB$, partiremos de da matriz C e encontraremos as matrizes A e B.

Esse método é interessante pelo fato de que usaremos um computador para computar essa quantidade de matrizes, dado um computador qualquer ele realiza X operações por segundo e, queremos que ele realize menos operações possiveis, dessa forma, vou estar gastando menos tempo. Seguindo o pessamento se temos $Ax=b$ para solucionar teremos que $x=inversa(A)b$.E realizar isso um milhão de vezes é muito custoso, até mesmo para maquinas.

Com isso realizando a fatoração LU, teremos ao invés de $Ax=b$, $LUx=b$. Dessa forma estaremos realizando por partes nosso problema, de forma a reduzir o número de operações ates realizados e, assim reduzimos o tempo.

Para aplicação desse métodos basta seguir um simples algoritimo[3]

i Escalone A até achar U;
ii As colunas de L serão encontradas durante o escalonamento (apenas colunas LI);
iii Se A possui mais colunas que linhas, as colunas de L são iguais a I.

Para visualização do método, o mesmo foi aplicado a sua lista destinada, os resultados podem ser vistos na Tabela 13 à tabela 21.

1	0	0	0
0.25	1	0	0
1.25	1	1	0
0.5	0.6667	1	1

Table 6. Matriz LU 1

4	-1	3	8
0	6	1	-5
0	0	-3	-5
0	0	0	5

Table 7. Matriz U 1

-3
4
4
6

Table 8. Resolução Matriz 1

Apresentando os resultados como a matriz L, matriz U e a solução para seu respectivo sistema, sendo o resultado apresentado da mesma forma que na seção anterior. Em cada aplicação dos métodos fez-se uso de tamanho de mantissas diferentes seguindo 4, 5 e 8 respectivamente em cada matriz, podendo analizar que temos uma aproximação maior dado um maior numero de cassas decimais.

[3]Créditos do algoritimo ao MeSalva:`https://www.youtube.com/watch?v=6VBdlqrS2yU`

1	0	0	0	0
-0.33333	1	0	0	0
2.66667	27.99946	1	0	0
0.66667	-4.99988	-0.21978	1	0
-0.33333	6.99988	0.26027	0.17222	1

Table 9. Matriz LU 2

-68
28
280
-72
362

Table 10. Resolução Matriz 2

3	-2	-1	7	3
0	0.33334	2.66667	0.33331	-2.00001
0	0	-72.99865	-27.99919	49.99919
0	0	0	-4.13704	-1.041
0	0	0	0	0.16581

Table 11. Matriz U 2

1	0	0	0	0	0
0.25	1	0	0	0	0
-0.5	-1.66666667	1	0	0	0
1.75	4	-2.03743315	1	0	0
2.5	3.83333333	-2.133636363	0.27928898	1	0
2	2.33333333	-1.53475936	0.6587665	-1.9408528	1

Table 12. Matriz L 3

4	12	14	5	2	-1
0	-6	8.5	0.75	2.5	-1.75
0	0	31.1666667	4.75	3.166666666	-3.41666667
0	0	0	9.92780746	0.95187167	11.78877006
0	0	0	0	3.91597093	-9.38338265
0	0	0	0	0	-10.1382391

Table 13. Matriz U 3

23
11
14
-80
66
41

Table 14. Resolução Matriz 3

4. Cholesky

Seguindo a mesma idéia em ganho de eficiência como no método anterior apresentado, o método de cholesky tem por objetivo decompor A em GG^T desde que sua matriz seja positiva definida. Assim teremos uma matriz triangular inferior e sua matriz adjunta. Esse método é comumente usado para simulações de Monte Carlo, ja demonstrado ser duas vezes mais eficiênte que a fatoração LU. Os resultados podem ser visto a seguir, obervando as Tabelas 22 a 30.

3	0	0
-2	5	0
1	-1	4

Table 15. Matriz G 1

-1
0
2

Table 16. Resolução Matriz 1

3	-2	1
0	5	-1
0	0	4

Table 17. Matriz GT 1

2	0	0	0
-1	1.41421	0	0
2	0.70711	3.08221	0
5	-1.41422	0	1.8918

Table 18. Matriz G 2

3
1
-0
-1

Table 19. Resolução Matriz 2

2	-1	2	5
0	1.41421	0.70711	-1.41422
0	0	3.08221	0.64889
0	0	0	1.8918

Table 20. Matriz GT 2

1	0	0	0	0
2	1	0	0	0
-3	5	4	0	0
0	1	-1	2	0
3	-2	0	1	5

Table 21. Matriz G 3

-2
8
-1
4
0

Table 22. Resolução Matriz 3

1	2	-3	0	3
0	1	5	1	-2
0	0	4	-1	0
0	0	0	2	1
0	0	0	0	5

Table 23. Matriz GT 3

Como na seção anterior, apresentamos os resultados igualmente, apresentando G e a G^T. Junto a solução do sistema, para esse método fez-se uso de 5 digitos para mantissa.

5. Fatoração Gauss-Jacobi

Aqui iniciamos a primeira seção de metodos iterativos, apresentando Gauss-Jacobi. Tais métodos podem ser mais rapidos e exigir menos memória do computador, fornecendo sequências que convergem para soluções sob certar condições.

Dito isso, seja $Ax=B$ um sistema linear de ordem N. A ideia é generalizar o métodos do ponto fixo, escrevendo o sistema linear $x=Cx+g$, onde C é uma matriz de ordem N e g é um vetor coluna Nx1. Dado um vetor de iteração inicial $x^{(0)}$ podemos ir

construindo iterativamente.

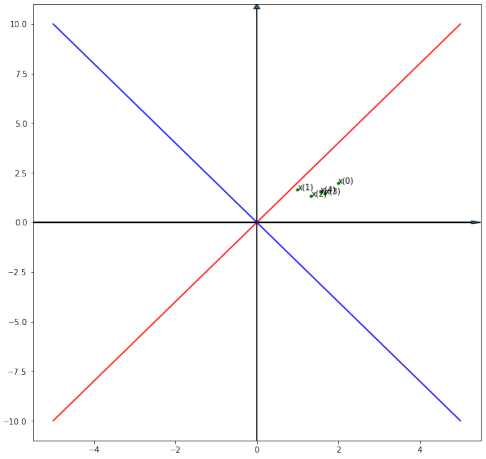

Figure 1. Gráfico da função um da lista cinco executada pelo método Gauss-Jacobi.

Na Figura 1, podemos observar o resultado obtido pela execução do código. Para a primeira execução passamos o erro como 0.05 e um total máximo de iterações 4. Podemos ver que o método se apresentou um tanto quanto eficiente (olhando singularmente), porém o mesmo só irá convergir em 7 iterações.

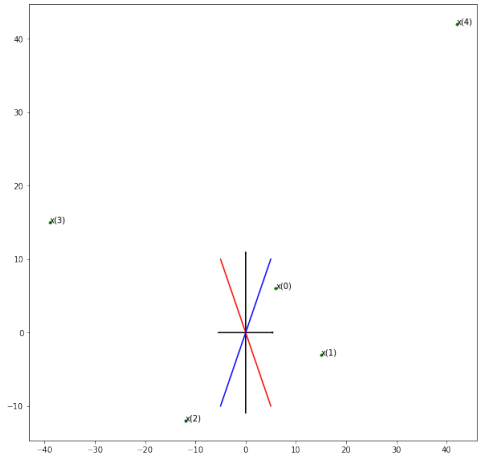

Figure 2. Gráfico da função dois da lista cinco executada pelo Gauss-Jacobi.

Na Figura 2, segui fazendo uso do 0.05 como erro e, para quatro para numero de máximo de iterações, porém testes com numeros mariores foram executados para comprovação de uma convergência a longo prazo, porém o sistema não converge.

Na Figura 3, segue os mesmo parâmetros, testes com maior numero de iterações foram executados, para comprovação de uma convergência em 750 iterações, porém por padronização seguimos com 4 iterações.

Seguindo na Figura 4, os mesmo parametros se sucedem, erro em 0.05 e numero de iterações máxima em 4. Neste sistema voltamos a convergir com um numero ainda menor que anteriormente, levando somente 4 iterações.

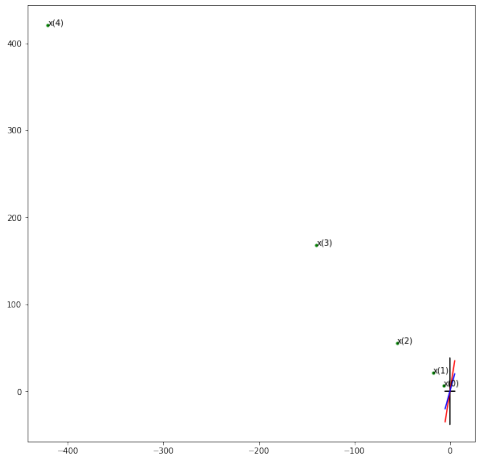

Figure 3. Gráfico da função três da lista cinco executada pelo método Gauss-Jacobi.

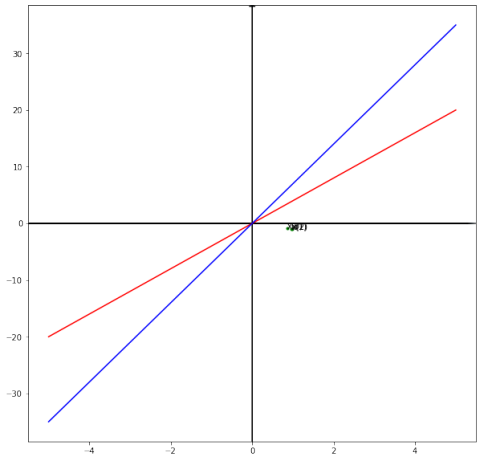

Figure 4. Gráfico da função um da lista cinco executada pelo método Gauss-Jacobi.

6. Fatoração Gauss-Seidel

Assim como na seção anterior, seguimos com métodos iterativos, o Gauss-Seidel é semelhante ao de Jacobi, seguindo os mesmos critérios para convergência. Neste método é condição suficiente que a matriz seja extritamente diagonal dominante, com isso fica garantido a convergência da sucessão de valores gerados para solução exata do sistema.

Com isso esse método acaba por convergir mais rapido que o Jacobi, isso pode ver visto de acordo com os resultados apresentados a seguir.

Na Figura 5 temos como parâmetro de entrada valores parecidos com os do método anterior, erro em 0.05 e numero máximo de iterações em 4. Podemos observar que apenas em quatro iterações o método ja convergiu, enquanto no método anterior precisou-se de 7, ou seja, o método realmente se demonstra mais "rapido" que o anterior.

Na Figura 6 ja podemos começar a notar a distância entre os pontos, levando as quatro iterações e ainda sim não convergindo como o anterior. Um maior numero de iterações foi passados para fim de teste de convergência, porém o mesmo segue a não convergir mesmo em 1000 iterações.

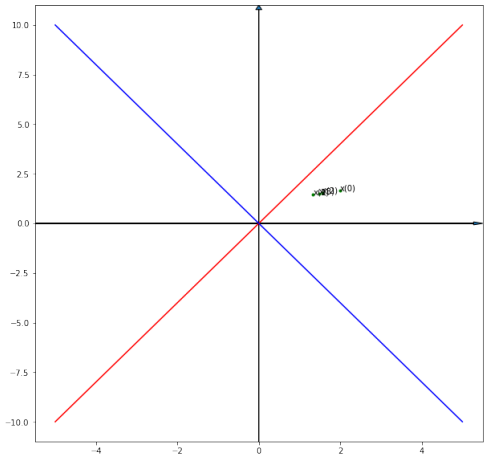

Figure 5. Gráfico da função um da lista cinco executada pelo método Gauss-Seidel.

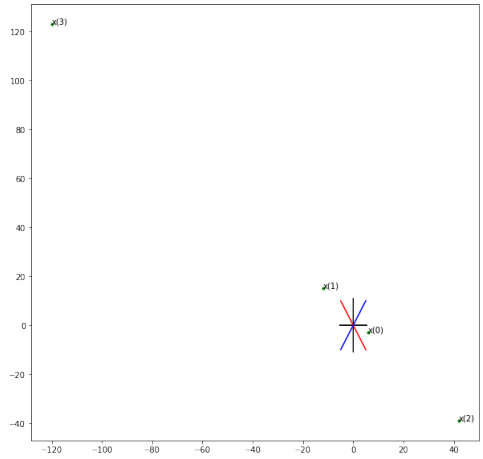

Figure 6. Gráfico da função dois da lista cinco executada pelo método Gauss-Seidel.

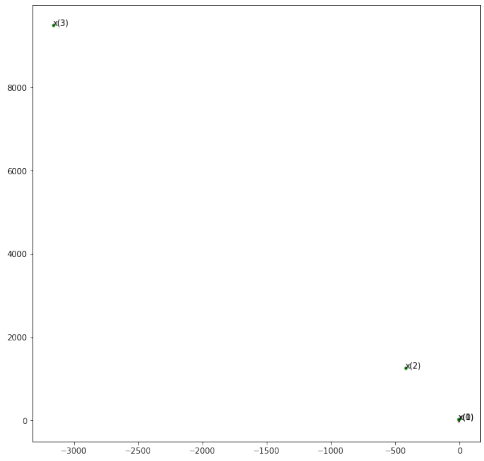

Figure 7. Gráfico da função três da lista cinco executada pelo método Gauss-Seidel.

Na Figura 7, encontramos um resultado semelhante com a 6, não convergindo ao final das iterações, e com uma grafico com pontos extremamente distântes.

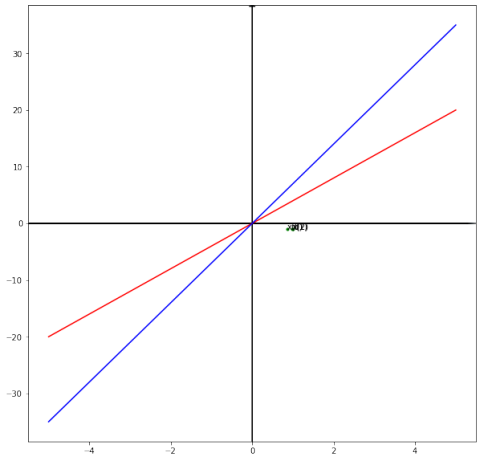

Figure 8. Gráfico da função um da lista cinco executada pelo método Gauss-Seidel.

Analisando a Figura 8, notamos que voltamos a converir, só que agora com um numero ainda menor que o anterior, convergindo em somente 3 iterações, podemos notar que gráficos que convergem, tendem seus pontos ficarem proximos, quanto aos que não convergem, mantem uma certa distância entre eles, isso é notavel na aplicação deste método, bastando analisar os gráficos.

7. Newton

O método de Newton tem por objetivo resolver equações não lineares (observe que estou falando desse método em específico, dado que equação de Newton com algumas alterações tem outros objetivos), assim, consiste em dada uma aproximação inicial x^0 da solução, calcular a aproximação (formula ocultada), a cada iteração k $\geq$ 0, até que o critério de convergência seja satisfeito, que na minha minha implementação é dado pelo erro e numero iterações.

Como aproximação inicial foi passado para o primeiro sistema [1, 2, 3], erro 0.001 e numero de iteração máxima 4. Para as demais fez-se uso dos mesmo parâmetros, somente alterando a aproximação inicial para [1, 2]. Para os sistemas passado em lista fora encontrado as respectivas soluções, sera aprensentado em cada iteração, observando as Tabelas 31 a 34.

-2.629	8.041	3.041
-2.372	4.068	1.799
-2.993	1.75	0.554
-1.188	1.381	0.451

Table 24. Solução do exercício A aplicando Newton.

No arquivo .ipynb é encontrado esse método com o numero maximo de iterações em 100 e gráficos plotados para os mesmos, a fim de averiguar quantas mais iterações seria possivel até se aproximar do erro. Com isso podemos ver um maior aproximação, ou seja, uma maior precição de fato, sendo restringindo somente pelo erro e não mais pelo número de iterações.

0.456	1.053
0.153	0.541
-0.04	0.274
0.128	0.137

Table 25. Solução do exercício B aplicando Newton.

-0.001	-0.002
-0.001	-0.001
-1.0000e-03	1.1111e+02
-1.0000e-03	7.4073e+01

Table 26. Solução do exercício C aplicando Newton.

0.28169014	1.30985915
-0.42463288	0.95573739
0.03303406	0.62510336
0.63138713	-0.12186995

Table 27. Solução do exercício D aplicando Newton.

Porém as entrada são valores "chutes", logo o método de Newton sofre com isso, pois a mesma contém alguns inconvenientes. A aproximação inicial deve estar proxima da solução para que o método venha a convergir. Quando N é grande, pode ser muito custoso calcular e fatorar $J(x_k)$. Entre outros, ou seja, o conhecimento do método e sua aplicabilidade, deve serem conhecidas antes.