MAP2320 - Métodos Numéricos em EDPs
Relatório do Exercício Programa

Jeferson Barbosa - N.USP: 7991253

1. Introdução

Este relatório tem o objetivo de mostrar os resultados obtidos da equação de Poisson pelo método das diferenças finitas, considerando-se três métodos computacionais: Jacobi, Gauss-Seidel e SOR. Tal modelo computacional foi usado também na modelagem do campo eletrostático entre duas placas paralelas com diferença de potencial de 110V.

Por razões de performance computacional, a implementação foi escrita na linguagem C, utilizando o sistema operacional Ubuntu 18.04.3 LTS, e compilado com o compilador de código aberto gcc versão 7.4.0 .

No final deste relatório, há uma seção contendo comentários sobre a implementação do código e como executar com diferentes parâmetros.

2. Resultados e discussão

Conforme especificado no enunciado do EP, foi usado inicialmente uma tolerância de 10^{-5} *h* , porém o erro absoluto máximo em relação à matriz de solução analítica começou a aumentar conforme a malha ficava mais refinada.

Em um segundo momento, coloquei a Tolerância como 10^{-8} *h* , e o erro absoluto máximo caiu com uma razão de 4, conforme o N dobrava. Porém a partir de N=64 ou 128, os métodos começaram a apresentar um aumento no erro máximo, o que nos leva a crer que o método não estava convergindo com N grande, mesmo com uma tolerância baixa de 10^{-8} *h.*

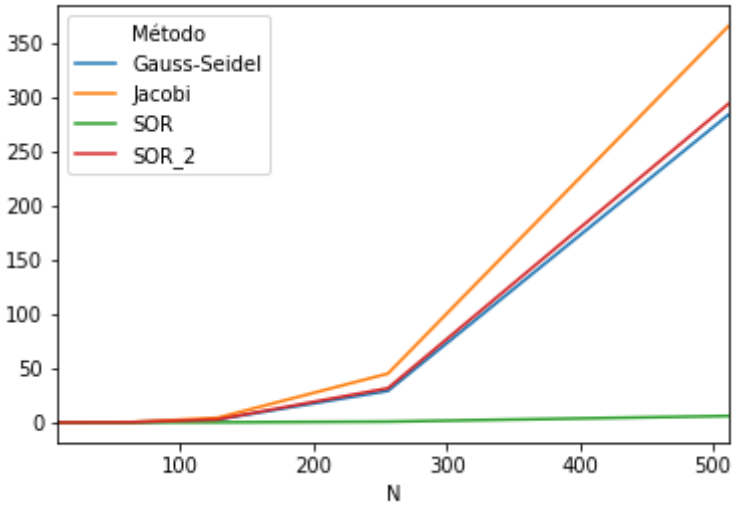
O método SOR com o parâmetro ótimo, se mostrou muito mais resistente com relação ao aumento do erro. Para TOL= 10^{-8} *h* e N=512, não apresentou este problema, mas para TOL = 10^{-5} *h,* até mesmo o SOR apresentou um aumento no erro máximo.

O parâmetro SOR_2 nas tabelas e nos gráficos, se refere ao método SOR com w = 2 / (1 + cos(π *h)). Podemos observar pelos resultados das tabelas, e nos gráficos abaixo , que dependendo do valor do parâmetro o

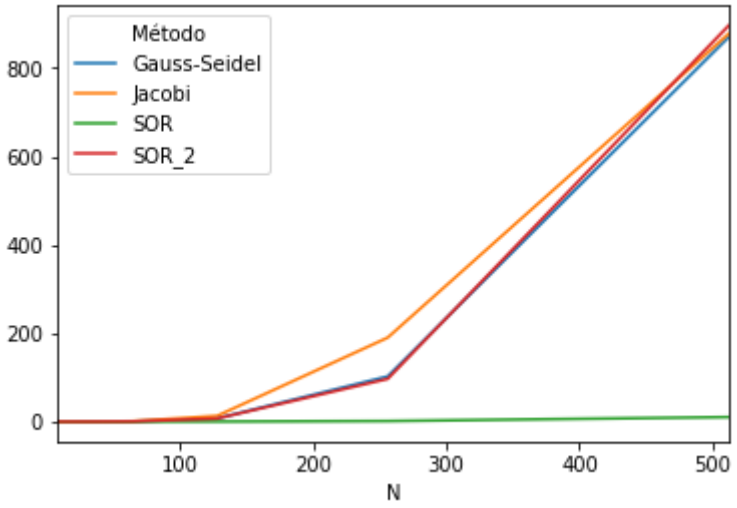
algoritmo demora muito para convergir, de modo que em alguns casos se atinge a interação máxima de 150 mil antes mesmo de convergir. Isso nos mostra como a chave para a eficiência do SOR depende do seu parâmetro ótimo.

Obs: No exercício 2, a função F do lado direito da equação de poisson fica com sinal negativo, já que o laplaciano de V é multiplicado por -1, originalmente.

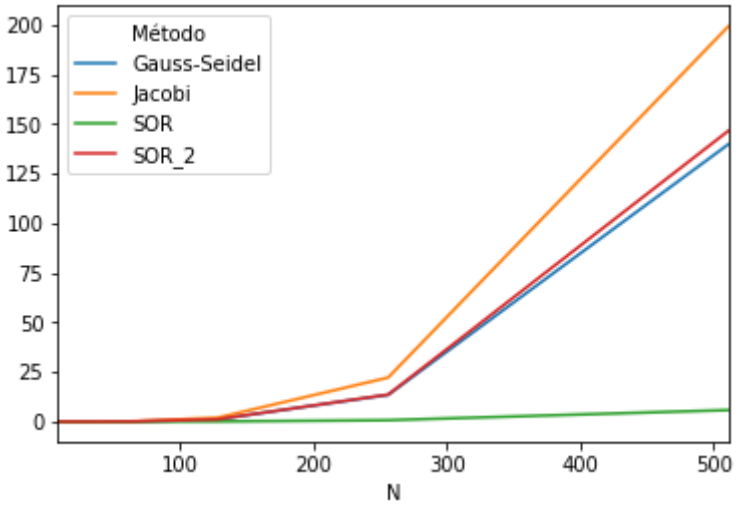
2.1 - Exercício 1



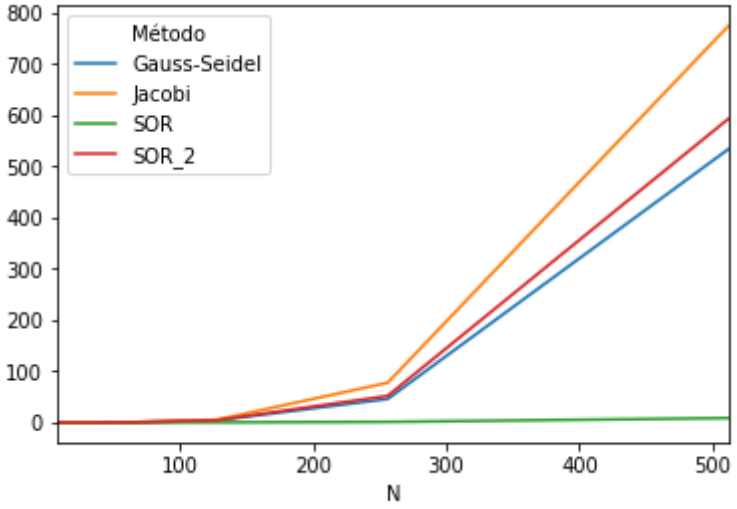
[Figura 1] - Tempo(seg), para o exercício 1-A com 10^{-5} *h*



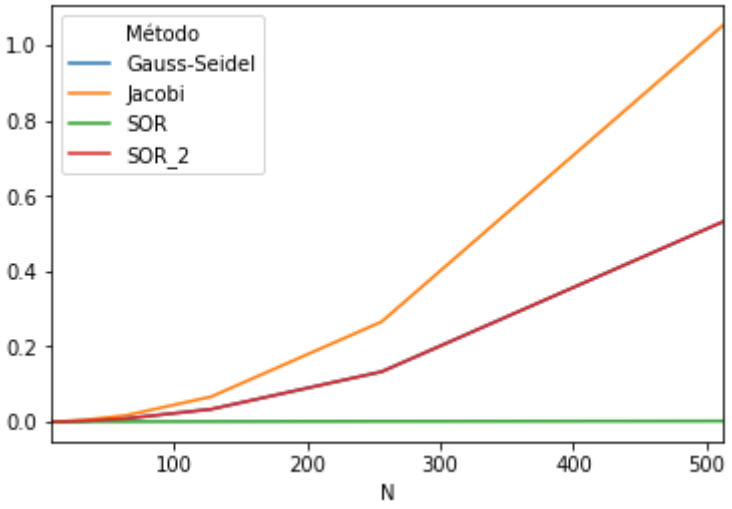
[Figura 2] - Tempo(seg), para o exercício 1-A com 10^{-8} *h*



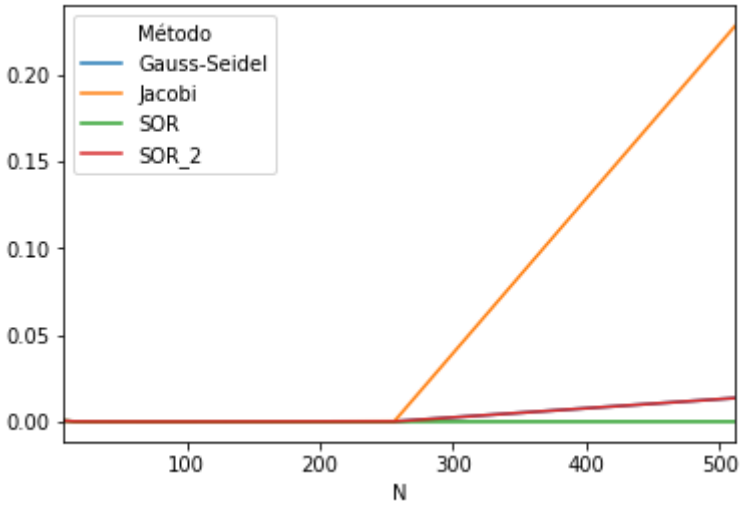
[Figura 3] - Tempo(seg), para o exercício 1-B com 10^{-5} *h*



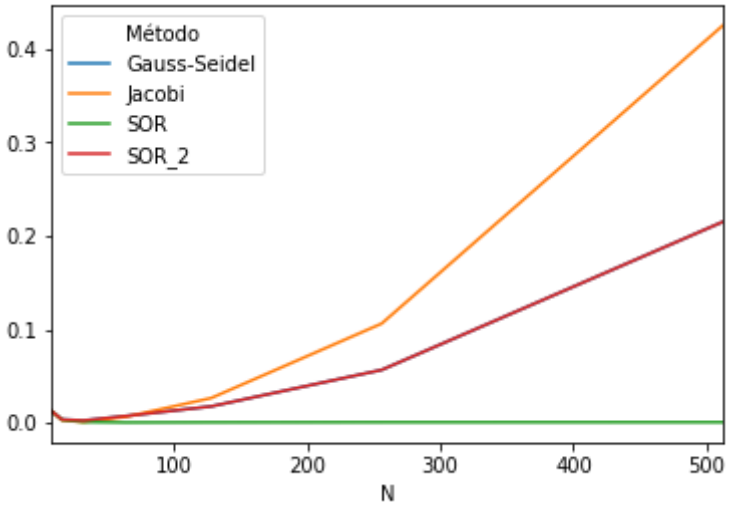
[Figura 4] - Tempo(seg), para o exercício 1-B com 10^{-8} *h*



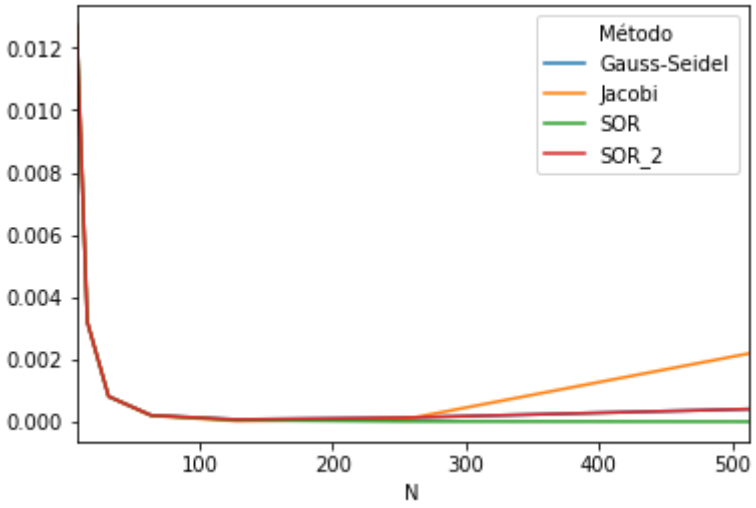
[Figura 5] - Erro máximo para o exercício 1-A com 10^{-5} *h*



[Figura 6] - Erro máximo para o exercício 1-A com 10^{-8} *h*

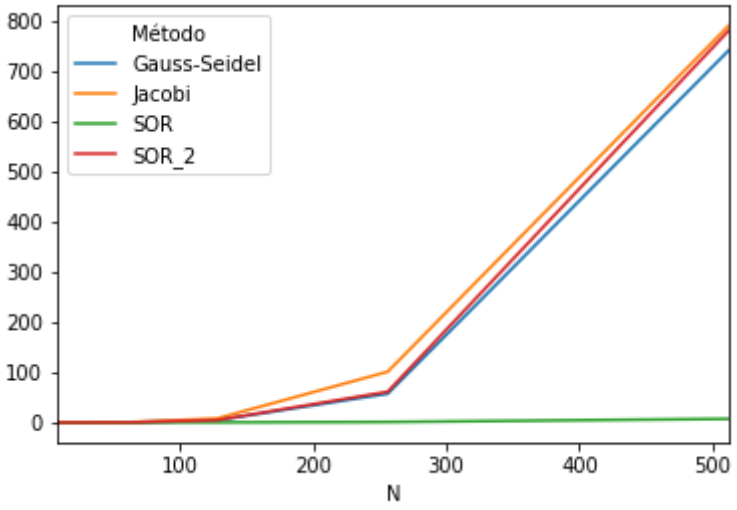


[Figura 7] - Erro máximo para o exercício 1-B com 10^{-5} *h*

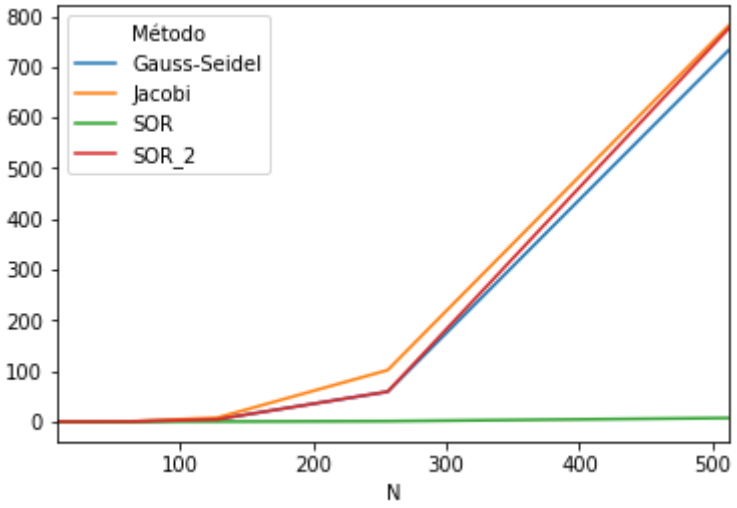


[Figura 8] - Erro máximo para o exercício 1-B com 10^{-8} *h*

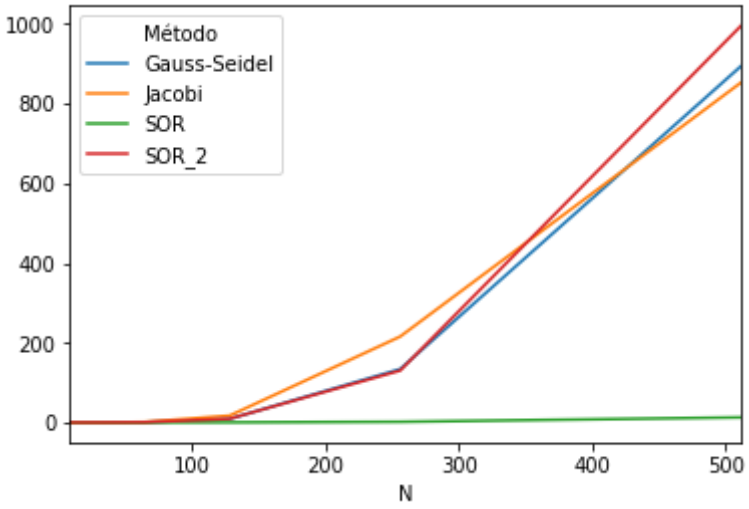
2.1 - Exercício 2



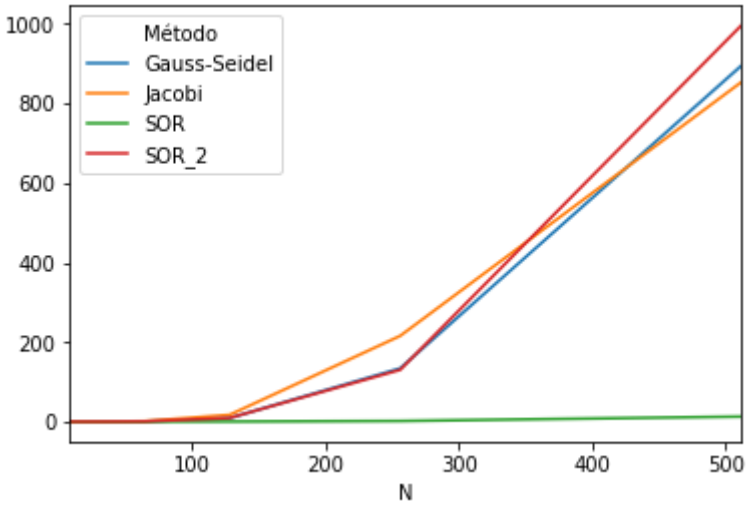
[Figura 9] - Tempo(seg) para o exercício 2-A com 10^{-5} *h*



[Figura 10] - Tempo(seg) para o exercício 2-B com 10^{-5} *h*



[Figura 11] - Tempo(seg) para o exercício 2-A com 10^{-8} *h*



[Figura 12] - Tempo(seg) para o exercício 2-B com 10^{-8} *h*

Através das tabelas e dos gráficos, podemos perceber que o erro absoluto máximo diminui quando diminuímos a tolerância, porém o custo computacional é alto. Vemos um aumento dramático no número de iterações e no tempo gasto para atingir a convergência, e em alguns casos, a convergência não é atingida mesmo após 10 minutos de processamento. Mesmo utilizando-se uma linguagem de alta performance como C, o tempo para cada iteração com N=512 é relativamente alto.

Podemos concluir que o algoritmo SOR com parâmetro ótimo é extremamente eficiente, com ordens de magnitude em relação aos métodos de Jacobi e Gauss-Seidel. O menos eficiente é o método de Jacobi, onde a convergência demora muitas iterações, ou então, nem é atingida antes do limite de 150 mil iterações. O algoritmo de Gauss-Seidel, em média, consegue convergir com metade do número de iterações de Jacobi(com o mesmo N).

A Tabela 3 nos mostra que o erro máximo tende a diminnuir quando diminuímos a tolerância para 10^{-8} *h,* a Figura 6 não demonstra isso com tanta nitidez pois os números têm ordens de magnitude diferentes.

3. Código-fonte

3.1 Arquivos e compilação

O código fonte do método iterativo foi escrito primariamente em Python, porém devido à problemas de performance, o código foi reescrito em linguagem C.

Há 3 arquivos com extensão ".c", um arquivo ".h" e um arquivo ".ipynb"(python notebook) que foi usado para auxiliar na manipulação do output gerado pelo programa em C, construindo as tabelas e os gráficos presentes neste relatório.

Para compilar o arquivo executável, executamos o seguinte comando no terminal bash do linux: *gcc main.c functions.c specific_functions.c -o main -lm*
Ou seja, temos o arquivo principal chamado main.c, e dois arquivos separados(*specific_functions.c, functions.c*) contendo funções relativas aos métodos iterativos e funções auxiliares.

specific_functions.c contém as funções de cada exercício, já que as funções f e g são diferentes em cada um deles. O header function.h é onde declaramos as assinaturas das funções que estão contidas em arquivos ".c", basicamente utiliza-se esse header para evitar que se tenha um número muito grande de linhas no arquivo principal(*main.c*).

A flag "-lm" durante a compilação tem a finalidade de indicar ao gcc que estamos usando funções específicas da biblioteca "math.h", caso contrário, o compilador não irá reconhecer funções de seno ou cosseno, por exemplo.

3.2 Notas sobre a implementação e os outputs

Para testarmos a implementação, simplesmente executamos no terminal o comando "./main", que é o nome do executável gerado com o comando acima(gcc). Note, porém, que a tolerância *TOL* pode ser diminuída dentro da função *"iterative_method"* no arquivo *"functions.c"*; consequentemente, isso irá implicar em um aumento no tempo de execução do algoritmo, como foi demonstrado na seção acima.

Uma sugestão para uma rápida inspeção: podemos substituir a linha *int array_N[7] = {8, 16, 32, 64, 128, 256, 512};* por algo como *int array_N[5] = {8, 16, 32, 64, 128};* já que para N = 512 o tempo pode ser alto, cerca de 10-14 minutos.

Outro ponto a se observar é que a variável *int exercise,* pertencente à função *void exercises (main.c),* é utilizada para executar os 4 exercícios propostos: 1-A, 1-B, 2-A, 2-B, ou seja, executando o código uma vez, os outputs dos quatro exercícios serão anexados em dois arquivos separados: *results_1.csv* e *results_2.csv.* Caso tenha necessidade de se executar apenas um método iterativo para um exercício específico, recomendo comentar os loops dentro da função *void exercises,* e também a chamada dos outros exercícios na função principal *int main.*

3.3 Notebook Python

Está incluso o notebook usado para gerar os gráficos e tabelas presentes nos resultados acima. Para facilitar a manipulação dos dados pelo pandas, foi escrito um “cabeçalho” para identificar cada bloco de dados nos arquivos de saída do C (results).

Usei as bibliotecas matplotlib e pandas, juntamente com o python 3.5