Gabriel Pietroluongo Nascimento

Relatório do Segundo Trabalho de Algoritmos Numéricos

Universidade Federal do Espírito Santo

2019

Sumário

1 Introdução

1.1 Objetivo

Este trabalho tem como objetivo a solução da equação de Poisson pelo método das diferenças finitas centrais, com o sistema resultante sendo resolvido pelo método SOR. Com isso, é possível testar o sistema resultante com uma solução conhecida ou propor uma modelagem aproximada de um problema com comportamento conhecido sem solução exata conhecida. Para isso, foi proposta uma especificação técnica para a implementação de um sistema que resolva problemas que possam ser modelados dessa forma, dado um conjunto de dados em forma de uma função aproximante e um determinado número de casos de contorno, aproximando o resultado gerado do resultado real. Com isso, é possível gerar mapas de potencial e de campo elétrico de determinado sistema, possibilitando uma análise aprofundada das características do fenômeno observado.

1.2 Método de Poisson

Para a aplicação do método de Poisson, é considerado um domínio $\Omega = [a, b] \times [c, d]$, com fronteira descrita por $\partial\Omega$ de forma que

$$f(x, y) = -\left(\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2}\right) \tag{1.1}$$

$$g(x, y) = V \tag{1.2}$$

Com isso, temos que a equação 1.1 é a definição para x, y dentro do domínio Ω do sistema e a equação 1.2 é a definição para x, y na fronteira do domínio. Além disso, $f(x, y)$ e $g(x, y)$ são funções conhecidas. O objetivo do método é determinar uma função $V(x, y)$ no interior do domínio Ω, utilizando subdivisões do domínio em células retangulares pelo método das diferenças finitas. Após essa segmentação o resultado será um sistema linear, que será resolvido neste projeto por meio do método SOR (e testado posteriormente com o método de Gauss-Seidel).

A discretização da equação de Poisson por diferenças finitas centrais pode ser escrita em termos de constantes a, b, c, d, e e de um vetor solução f, de forma que a expressão no ponto (i, j), ou seja, f_{ij}, seja dada pela equação

$$f_{ij} = aV_{(i+1),j} + bV_{(i-1),j} + cV_{i,(j+1)} + dV_{i,(j-1)} + eV_{ij} \tag{1.3}$$

onde as constantes a, b, c, d, e são dadas por

$$a = b = \frac{-1}{h_x^2} \tag{1.4}$$

$$c = d = \frac{-1}{h_y^2} \tag{1.5}$$

$$e = 2 \cdot \left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) \tag{1.6}$$

Com o cálculo dessas constantes e a matriz f mencionada anteriormente, que consiste no cálculo de $f(x, y)$ para todo x, y no domínio de análise, é possível calcular pelo método SOR uma matriz-solução V, de $i = n_x \cdot n_y$ elementos, no qual cada elemento V_i é dado pela equação

$$V_i = \frac{\omega}{e} \cdot (f_i - dV_{i-n_x} - bV_{i-1} - aV_{i+1} - cV_{i+n_x}) + (1 - \omega)V_i \tag{1.7}$$

Com isso, é possível iterar nos termos do vetor V e aproximar os valores do valor exato da função real. Por fim, é necessário considerar casos de contorno, caso existam, nas iterações do SOR.

Por fim, para o cálculo do campo elétrico, que é descrito pela equação

$$\vec{E} = -\nabla V = -\left(\frac{\partial V}{\partial x}, \frac{\partial V}{\partial y}\right)^T \tag{1.8}$$

foi utilizada a definição matemática de derivada, representando cada elemento da matriz que representa o campo elétrico $\vec{E}$ sendo descrito na forma

$$E_{i,j} = \frac{E_{i,(j+1)} - E_{i,(j-1)}}{2h_x} + \frac{E_{(i+1),j} - E_{(i-1),j}}{2h_y}, 0 < i < n_x, 0 < j < n_y \tag{1.9}$$

desconsiderando a borda do domínio.

2 Desenvolvimento do projeto

2.1 Implementação do Projeto

Para o desenvolvimento do projeto, a linguagem escolhida foi C++, por conta da robustez no processamento e das características de orientação a objetos que possibilitam uma generalização das aplicações do sistema, não limitando ele a apenas os problemas apresentados na especificação.

2.1.1 Implementação do algoritmo

Conforme descrito na seção anterior, o algoritmo de Poisson foi implementado para solucionar o sistema proposto. Para isso, seguindo a especificação do trabalho, primeiramente foram definidos h_x e h_y para analisar os resultados em diferentes malhas. Assim, o algoritmo foi testado com $h_x = h_y = 0.5, 0.25, 0.125$, e os resultados obtidos comparados. Com esses valores, é possível definir a quantidade de subintervalos n_x e n_y seguindo as seguintes equações:

$$n_x = 1 + \frac{(b-a)}{h_x} \tag{2.1}$$

e

$$n_y = 1 + \frac{(d-c)}{h_y} \tag{2.2}$$

com a, b, c, d sendo os limites do domínio tal que $\Omega = [a, b] \times [c, d]$. Após a definição desses valores, foi definido um ω seguindo a equação

$$\omega = \frac{8 - \sqrt{64 - 16t^2}}{t^2} \tag{2.3}$$

onde

$$t = \cos\left(\frac{\pi}{n_x}\right) + \cos\left(\frac{\pi}{n_y}\right) \tag{2.4}$$

conforme proposto no material de apoio. Com isso, temos o domínio discretizado na forma $\Omega = \{(x, y) | a < x < b, c < y < d\}$.

Com esses valores, é possível calcular os valores das constantes a, b, c, d, e por meio das equações 1.4, 1.5 e 1.6. Seguindo o modelo proposto, também foi preenchida a matriz f com os valores de $f(x, y)$, conforme descrito na seção 1.2. Nessa etapa, também são aplicadas as condições de contorno da função descritiva no vetor f, antes da aplicação do método SOR. Caso tenha sido providenciada uma função de validação (solução exata), esse ponto do código seria o ideal para o preenchimento da matriz de soluções exatas, chamada daqui para a frente de *matriz ground-truth* e representada por q. Com todos esses elementos preparados, é possível aplicar a solução via SOR seguindo a equação 1.7 até um máximo de iterações predefinido, completando o algoritmo e produzindo o resultado aproximado da função original. Posteriormente, é possível comparar o erro do método utilizando a equação sugerida pela especificação do projeto,

$$\epsilon = max|q_i - v_i|, 0 < v < (n_x \cdot n_y) - 1 \tag{2.5}$$

Analisando o erro, é possível definir a qualidade da aproximação do sistema, dependendo de quanta precisão é necessária pelo usuário.

2.1.2 Escrita inicial do algoritmo

Para o desenvolvimento projeto, inicialmente foi implementado um algoritmo "cru"em C++, seguindo o processo descrito na seção 2.1. De posse desse algoritmo inicial, foi possível testar a implementação e obter um resultado de acordo com o esperado pela especificação do projeto (erro menor que 10^{-6}). Além disso, foram testados vários valores de iterações e de passos, a fim de verificar se a solução seria aplicável em mais de um caso e garantir que não haviam *bugs* no projeto. Uma mudança em relação ao material de apoio foi que os "casos de contorno"do método SOR não foram calculados seguindo a especificação, mas sim analisando quais seriam os problemas enfrentados a cada iteração[1]. Com isso, a cada iteração do SOR, o cálculo dos termos utilizando a equação 1.7 é considerada separadamente, e cada um dos termos que poderia ter problemas de memória é analisado individualmente. Com uma solução armazenada em memória, é possível calcular o campo elétrico trivialmente por meio da equação 1.9. Considerando que o algoritmo inicial foi uma versão mais direta, o conteúdo dele é praticamente uma aplicação direta do algoritmo mencionado anteriormente, com a exceção mencionada acima, e com o uso de variáveis globais e outros artefatos da linguagem que facilitam a prototipagem de ideias mas em longo prazo atrapalham o desenvolvimento. Assim, esse código de qualidade menor foi analisado, otimizado e reescrito para a aplicação final no projeto.

2.1.3 Implementação da classe

De posse do algoritmo já escrito na linguagem-alvo, o código teria que ser reescrito para melhorar sua legibilidade e expansibilidade (já que deve ser utilizado para resolver mais de um único problema). Com isso, primeiramente todos os valores que antes eram *"hardcoded"* em seu interior foram substituídos por variáveis apropriadas. Os métodos foram analisados e uma classe em C++, *poissonSOR* foi criada para resolver os problemas modelados da forma proposta. Com isso, é possível apenas passando parâmetros e reimplementando pequenos pedaços de código (casos de contorno) utilizar a implementação base para qualquer problema do tipo sugerido. De fato, todos os problemas do projeto tiveram sua solução decorrente da aplicação direta da classe escrita. Isso viabiliza um fluxo de trabalho mais elegante, à medida que uma análise de um fenômeno diferente modelado seguindo a equação de Poisson pode ser implementado rapidamente utilizando o código mais complexo, já escrito, como base. As minúcias dos métodos e atributos da classe implementada estão descritos em profundidade na seção 3.2.

2.1.4 Geração de saídas

O código em C++ faz o processamento e salva o resultado em arquivos no formato *.txt* na pasta do programa. Com esses dados já processados, um *script* em *Python* foi desenvolvido para a geração dos dados de saída apresentados no capítulo 3, utilizando algumas bibliotecas próprias da linguagem, como *matplotlib* e *numpy*. Assim, dado um conjunto de dados de determinado problema e as informações sobre esse conjunto, ou seja: se $f_{[i\times j]}$ é o conjunto, dados todos os elementos f_{ij} e i, j, é possível gerar o mapa de potenciais (*heatmap*) bidimensional e tridimensional e de linhas equipotenciais do conjunto de dados.

[1] Isso é necessário pois o algoritmo apresentado originalmente considera que apenas multiplicar o resultado por 0 já seria o suficiente para evitar problemas de *overflow* / *underflow*. Contudo, como em C++ o acesso a memória em regiões impróprias não tem comportamento definido, é mais prudente analisar cada caso separadamente e evitar problemas de manipulação de memória indevida

2.2 Dificuldades encontradas

As principais dificuldades encontradas no projeto foram determinar a abordagem adequada e como organizar os códigos escritos em termos de paradigmas de programação, ou seja, se seriam utilizadas classes puras de C++ ou apenas *structs* simples de C, ou até mesmo apenas uma única função que resolveria um sistema nesse formato. Com isso, para possibilitar a entrega do trabalho no prazo especificado, uma abordagem *top-down* foi adotada, isto é, ao invés de focar na base do código, o foco foi alterado para obter um resultado por meio do algoritmo "cru"desenvolvido, conforme descrito anteriormente. Isso viabilizou um desenvolvimento mais rápido por não ser necessário implementar políticas de acesso (como membros privados/públicos) inicialmente, apenas na etapa de "tradução"do código original para o paradigma orientado a objetos. Considerando isso tudo, a escolha por classes no paradigma orientado a objetos foi feita pela necessidade de analisar mais de um problema do mesmo tipo, conforme descrito na especificação, e pela praticidade de reutilização de código decorrente desse paradigma. Com isso, é possível utilizar de forma eficiente todos os scripts escritos, mesclando a parte estática de cada problema individual (como domínio e função aproximante) com o componente comum (método SOR e cálculos de erros, por exemplo). Dessa forma, mesmo que fosse necessário resolver mais um problema diferente, com um domínio diferente e com uma função e casos de contornos diferentes, seria possível implementar apenas uma "expansão"ao código já existente e obter um resultado.

Outra dificuldade foi a implementação dos casos de contorno no algoritmo SOR, o que tomou grande parte do tempo de desenvolvimento. Com a abordagem tomada, isso não deve ser mais um problema futuro, considerando que a classe implementada deixa bem claro ao cliente como implementar os casos de contorno de forma eficaz. Os casos de borda do algoritmo SOR também tomaram grande parte do tempo de desenvolvimento do projeto, tendo em vista a dificuldade para analisar resultados que são decorrentes de *lixo* na memória do computador, conforme descrito na seção 2.1.2.

3 Conclusão

3.1 Resultados

Primeiramente, é necessário definir um número máximo de iterações realizadas. Para isso, foi realizado um estudo do erro em função da quantidade de iterações realizadas. O erro V_p é dado pela equação (2.5), e se refere ao resultado dos potenciais. O erro do campo elétrico é calculado de forma similar. Para cada teste, os valores de passo foram definidos como $h_x = h_y = p$, que será a nomenclatura utilizada na seção de conclusão do projeto. Com isso, as informações sobre o estudo feito estão dispostas abaixo:

Número de iterações	Erro V_p	Erro Campo Elétrico
1	12.538790426611	2.907351752175
10	0.191925842523	0.077129816371
20	0.006512396150	0.004120976576
30	0.000286799515	0.000160785244
40	0.000001920159	0.000000852291
50	0.000000014172	0.000000006622
60	0.000000000092	0.000000000064
70	0.000000000003	0.000000000001
80	0.000000000000	0.000000000000
90	0.000000000000	0.000000000000
100	0.000000000000	0.000000000000

Tabela 1 – Análise da validação com $p = 0.5$ e 12 casas decimais de precisão

Número de iterações	Erro V_p	Erro Campo Elétrico
1	14.167268340547	1.738923212435
15	0.460664261978	0.104090120849
30	0.029494914215	0.006383550807
45	0.002814599292	0.000791168413
60	0.000243688702	0.000048514634
75	0.000008360406	0.000001642513
90	0.000000049337	0.000000010677
105	0.000000001328	0.000000000288
120	0.000000000080	0.000000000015
135	0.000000000003	0.000000000001
150	0.000000000000	0.000000000000

Tabela 2 – Análise da validação com $p = 0.25$ e 12 casas decimais de precisão

Número de iterações	Erro V_p	Erro Campo Elétrico
1	14.921764813058	0.928441841153
30	0.504155358122	0.059665023973
60	0.028215890019	0.002967654161
90	0.003017518043	0.000355706314
120	0.000220905358	0.000018882376
150	0.000006437956	0.000000483484
180	0.000000035639	0.000000003380
210	0.000000001213	0.000000000083
240	0.000000000052	0.000000000003
270	0.000000000001	0.000000000000
300	0.000000000000	0.000000000000

Tabela 3 – Análise da validação com $p = 0.125$ e 12 casas decimais de precisão

Além dos dados expostos acima, conforme requisitado na especificação, temos as seguintes informações sobre o método de Gauss-Seidel:

Número de iterações	Erro V_p	Erro Campo Elétrico
1	14.250549	3.266255
9	2.826925	0.722804
18	0.442854	0.113686
27	0.068209	0.017551
36	0.010366	0.002674
45	0.001557	0.000402
54	0.000231	0.000060
63	0.000034	0.000009
72	0.000005	0.000001
81	0.000001	0.000000
90	0.000000	0.000000

Tabela 4 – Análise do método de Gauss-Seidel com $p = 0.5$ e 6 casas decimais de precisão

Número de iterações	Erro V_p	Erro Campo Elétrico
1	15.275007	1.870706
33	2.819086	0.222521
66	0.505988	0.040343
99	0.090314	0.007203
132	0.016044	0.001280
165	0.002842	0.000227
198	0.000502	0.000040
231	0.000089	0.000007
264	0.000016	0.000001
297	0.000003	0.000000
330	0.000000	0.000000

Tabela 5 – Análise do método de Gauss-Seidel com $p = 0.25$ e 6 casas decimais de precisão

Analisando os resultados obtidos, é possível perceber a clara diferença entre o método SOR e o método de Gauss-Seidel, comparados diretamente na tabela 7. Apesar da diferença ser menor em uma malha de resolução menor, como quando $p = 0.5$, é notória a diferença à medida de a quantidade de cálculos aumenta junto com a resolução da malha.

Na tabela 7, os valores do método SOR foram ajustados para considerar apenas as primeiras 6 casas decimais, e não as 12 mostradas nas tabelas 1, 2 e 3.

Número de iterações	Erro V_p	Erro Campo Elétrico
1	15.537207	0.966205
100	4.012718	0.079665
200	1.091962	0.022221
300	0.296028	0.006029
400	0.080054	0.001631
500	0.021616	0.000440
600	0.005832	0.000119
700	0.001572	0.000032
800	0.000424	0.000009
900	0.000114	0.000002
1000	0.000031	0.000001
1100	0.000008	0.000000
1200	0.000002	0.000000
1300	0.000001	0.000000
1315	0.000000	0.000000

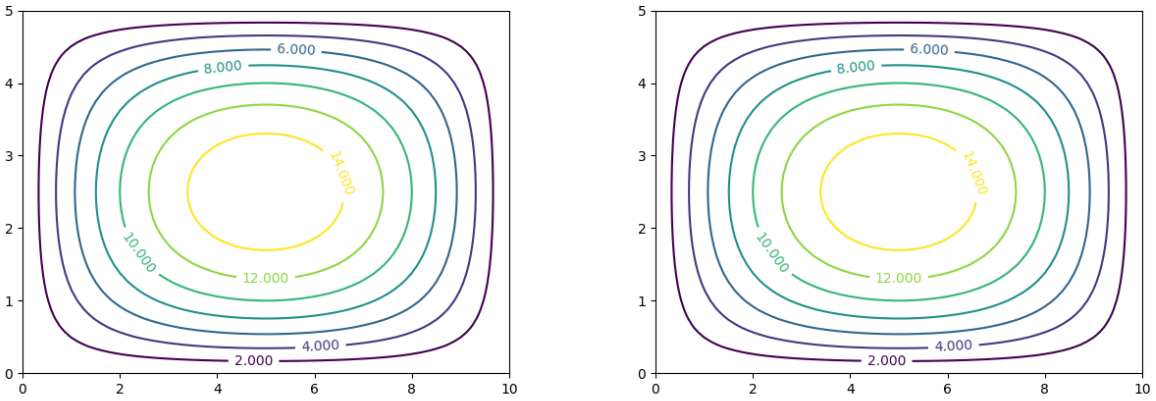
Tabela 6 – Análise do método de Gauss-Seidel com $p = 0.125$ e 6 casas decimais de precisão

Método	$p = 0.5$	$p = 0.25$	$p = 0.125$
Gauss-Seidel	81	330	1315
SOR	50	90	180

Tabela 7 – Análise do método de Gauss-Seidel com $p = 0.125$ e 6 casas decimais de precisão

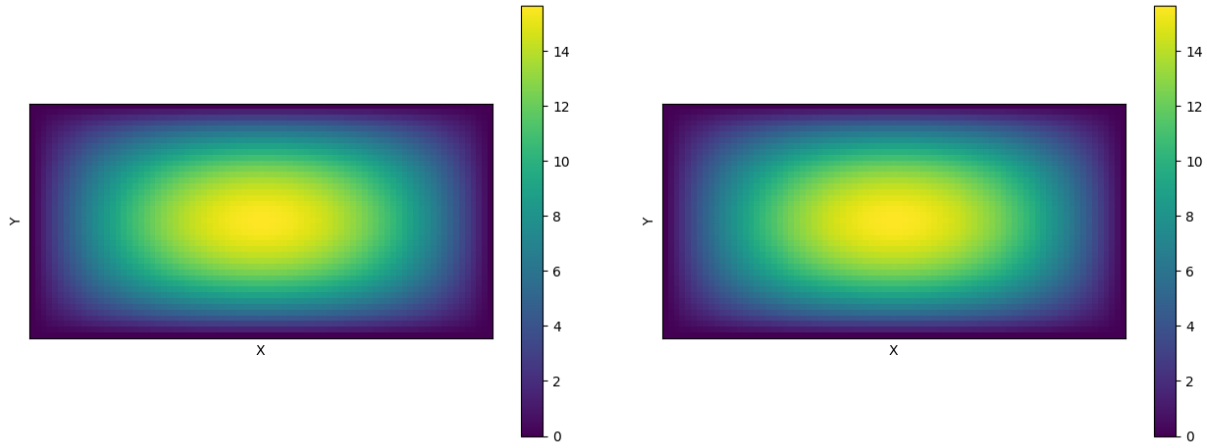
Analisando as tabelas apresentadas, é possível perceber que a partir de 300 iterações, o erro cometido é mínimo, mesmo com uma malha de alta resolução, com $p = 0.125$. Portanto, os próximos resultados apresentados foram gerados utilizando esse número de iterações.

Para gerar os resultados demonstrados abaixo, todos os casos de teste foram rodados com o número máximo de iterações definido em 300, conforme exposto acima. Nessas condições, os seguintes resultados foram obtidos:



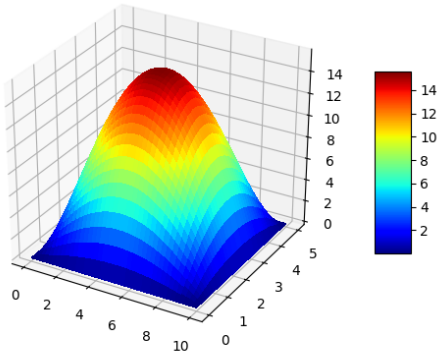
(a) Aproximação do sistema com passo $p = 0.125$ (b) Resultado exato com passo $p = 0.125$

Figura 1 – Mapa de linhas equipotenciais do sistema de validação

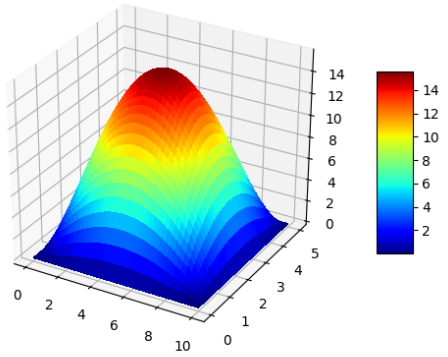


(a) Aproximação do sistema com passo $p = 0.125$ (b) Resultado exato com passo $p = 0.125$

Figura 2 – Mapa bidimensional dos potenciais do sistema de validação



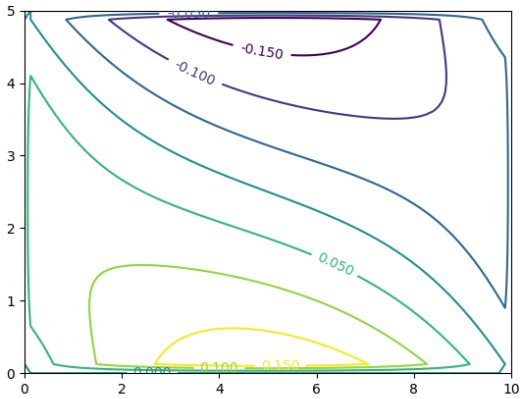
(a) Aproximação do sistema com passo $p = 0.125$



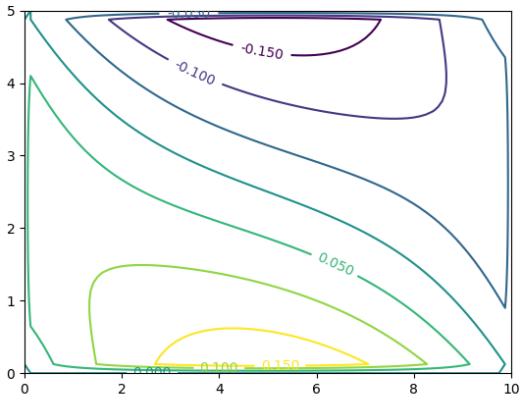
(b) Potenciais exatos com passo $p = 0.125$

Figura 3 – Modelo 3D dos potenciais do sistema de validação

Além da análise dos potenciais do sistema, também foi analisado o campo elétrico:

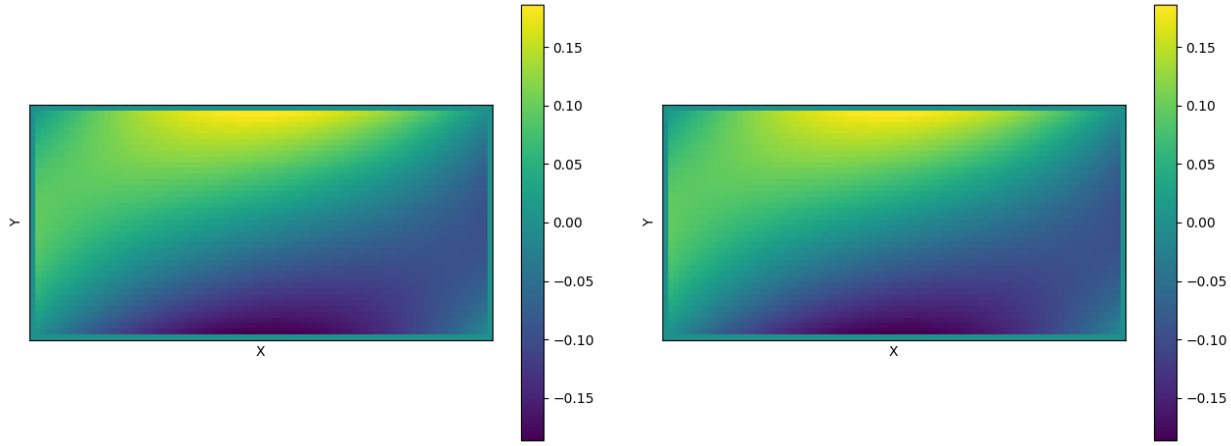


(a) Aproximação do sistema com passo $p = 0.125$



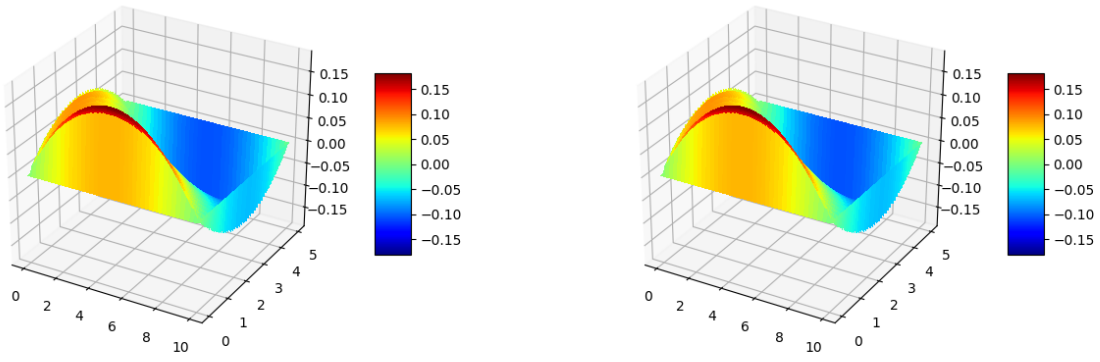
(b) Campo elétrico exato com passo $p = 0.125$

Figura 4 – Modelo 3D dos campos elétrico do sistema de validação



(a) Aproximação do sistema com passo $p = 0.125$ (b) Campo elétrico exato com passo $p = 0.125$

Figura 5 – Modelo 3D dos campos elétrico do sistema de validação



(a) Aproximação do sistema com passo $p = 0.125$ (b) Campo elétrico exato com passo $p = 0.125$

Figura 6 – Modelo 3D do campo elétrico do sistema de validação

Além dos casos de validação expostos acima, também foi analisado o sistema de capacitores:

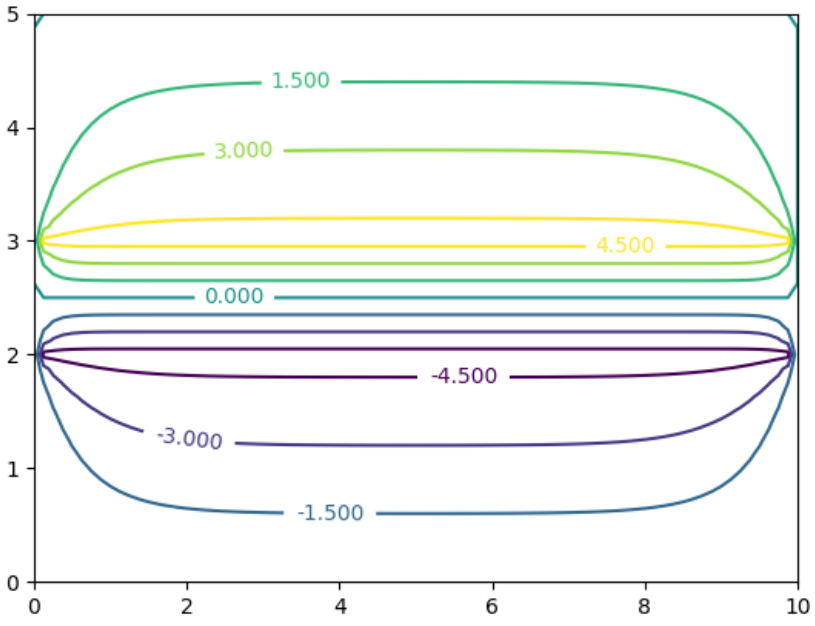


Figura 7 – Mapa de linhas equipotenciais do sistema de capacitores

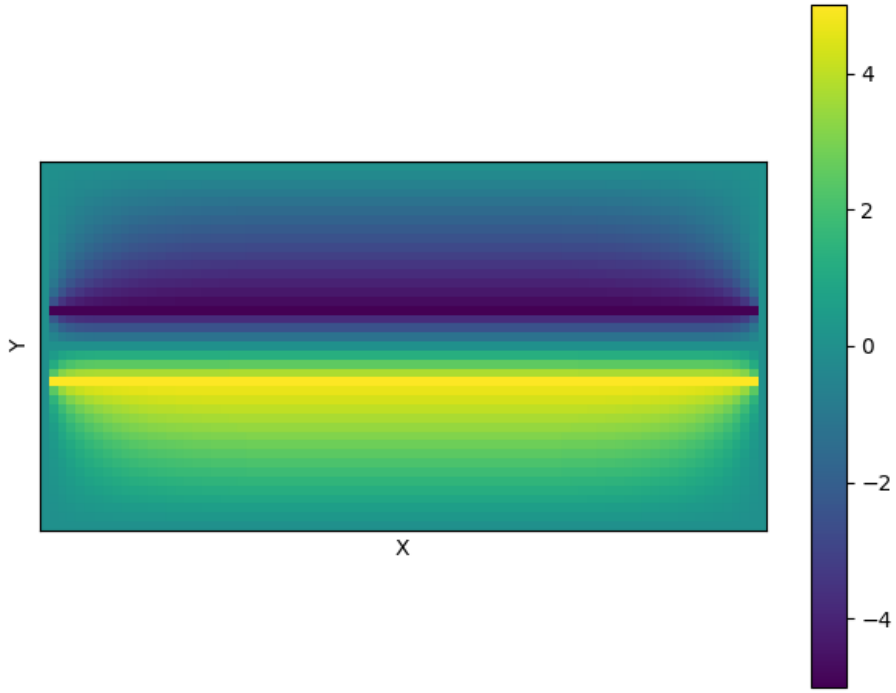


Figura 8 – Mapa bidimensional de potencial do sistema de capacitores

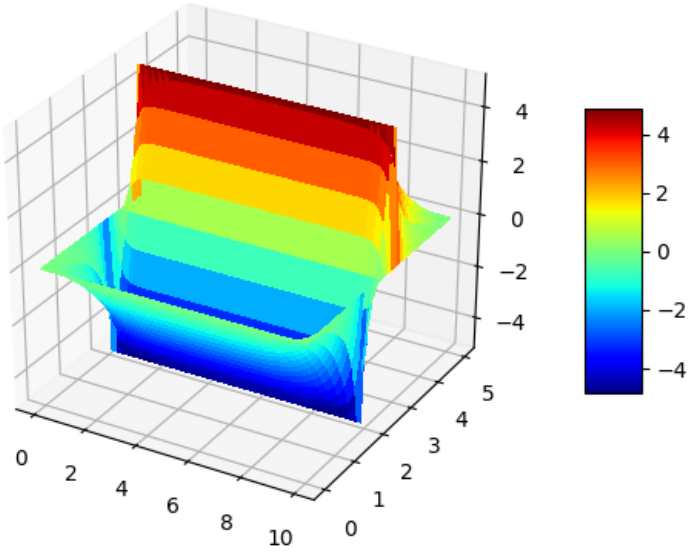


Figura 9 – Mapa tridimensional dos potenciais do sistema de capacitores

Por fim, a análise do campo elétrico do capacitor:

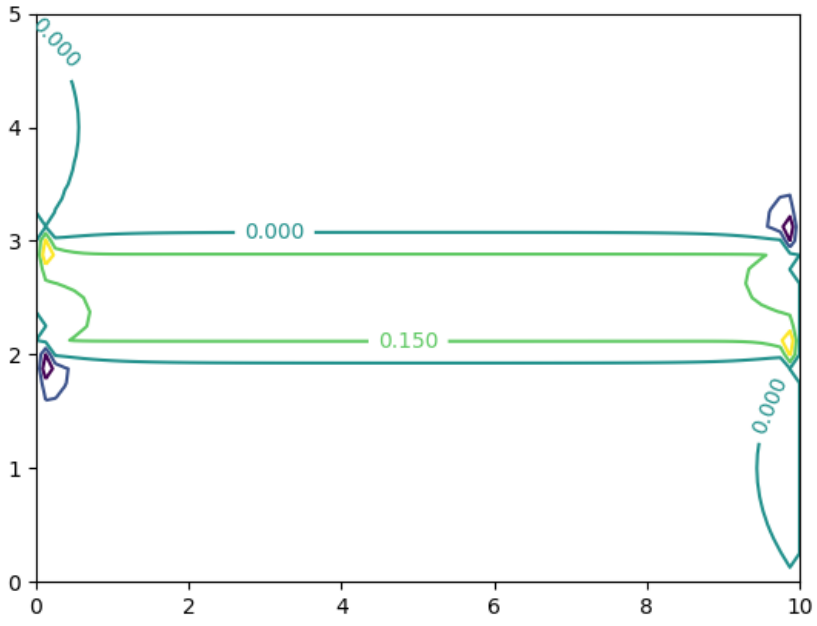


Figura 10 – Curvas de nível do campo elétrico do sistema de capacitores

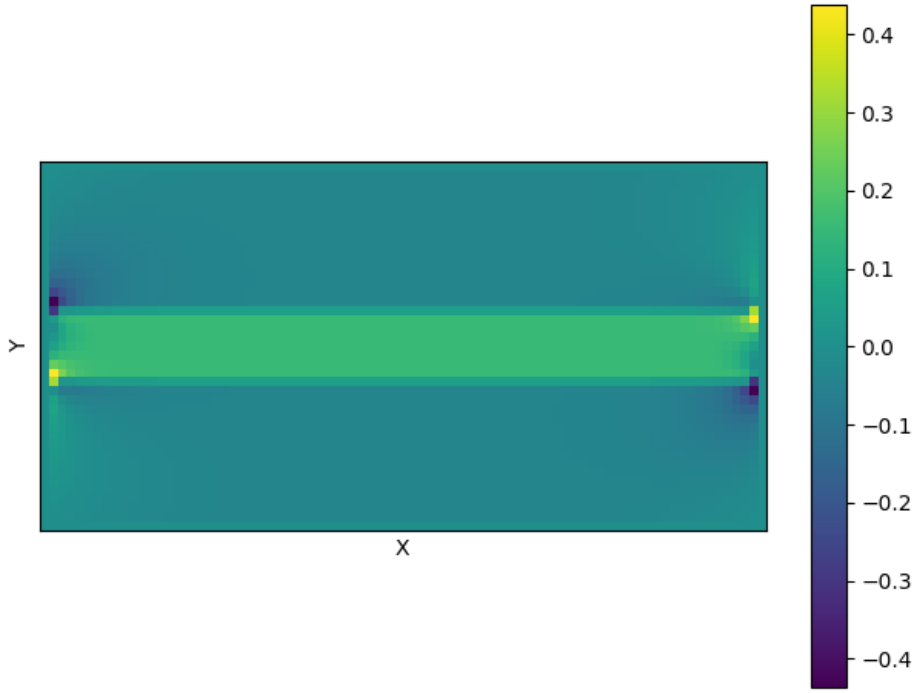


Figura 11 – Representação bidimensional do campo elétrico do sistema de capacitores

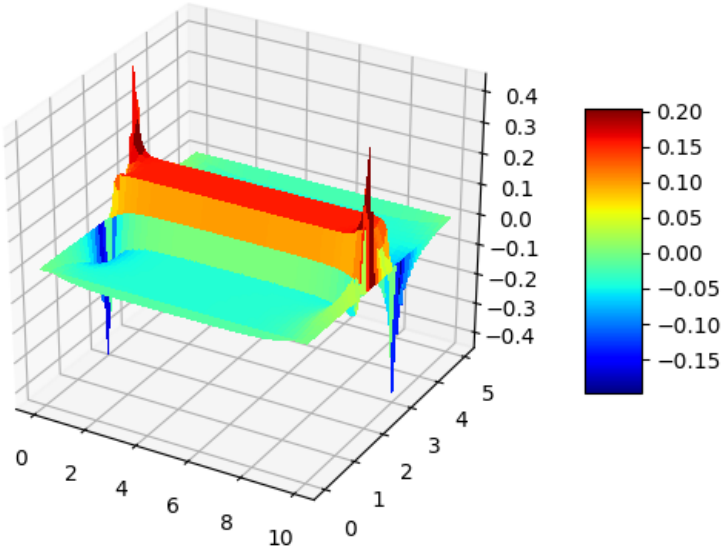


Figura 12 – Mapa tridimensional do campo elétrico do sistema de capacitores

Imagens adicionais demonstrando as outras combinações de resolução de malhas e de tipos diferentes de gráficos estão na pasta `output/imgs`, dentro da pasta na qual este relatório se encontra. Adicionalmente, os gráficos tridimensionais podem ser analisados em um ambiente tridimensional rodando o script `output/plot.py` e escolhendo o nome de arquivo adequado dentro do script.[1] A visualização tridimensional permite um melhor entendimento da distribuição dos potenciais e do campo elétrico no gráfico, dado que é possível acessar mais informações do que apenas com uma imagem bidimensional estática.

3.2 Documentação da classe

Abaixo constam as instruções para fazer uso do código desenvolvido ao longo do projeto. Informações sobre algoritmos específicos se encontram dentro do próprio código, então o conteúdo abaixo é uma visão geral das propriedades da classe.

3.2.1 Extendendo a funcionalidade da classe

Para extender a funcionalidade da classe, isto é, implementar novos tipos de problemas a serem resolvidos, primeiramente o cliente deve criar uma nova instância da classe `poissonSOR`, com o domínio e o passo desejados. Após isso, é necessário fornecer uma função de avaliação ao problema, isto é, a função $f(x, y)$ a ser calculada no domínio. Também é possível especificar um número máximo de iterações, por meio do método `setMaxIter()`. Caso não haja nenhum caso de contorno, isso já é o suficiente para utilizar a classe, chamando o método `process()` e analisando os resultados. Caso seja necessário utilizar casos de contorno, é necessário, primeiramente, definir um novo tipo de problema no `enum type`, no cabeçalho da classe. Após isso, o cliente deve adicionar as funções de contorno por meio do método `addContorno()`. Com isso, o vetor de funções de contorno é preenchido apropriadamente. Após isso, o cliente deve alterar, na implementação da classe, as funções `checkContornos()`, `typeToString()` e `doSOR()`, adicionando uma cláusula ao sistema de `switch` já implementado. Com isso, os casos de contorno serão considerados nos cálculos. Assim, é possível implementar novos casos de teste rapidamente e de forma modularizada.

[1] Tenha em mente que para possibilitar a visualização dos gráficos, é necessário ter os pacotes *numpy* e *matplotlib* instalados no sistema. Adicionalmente, é necessário configurar os valores das variáveis h_x e h_y dentro do script de acordo com o arquivo de entrada escolhido.