UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ - UTFPR
CURSO DE LICENCIATURA EM MATEMÁTICA

JOÃO PEDRO SANTOS BRITO MICHELETTI

SOLUÇÃO NUMÉRICA DA EQUAÇÃO DE POISSON EM MALHAS ESTRUTURADAS BIDIMENSIONAIS E TRIDIMENSIONAIS

CURITIBA

2021

JOÃO PEDRO SANTOS BRITO MICHELETTI

SOLUÇÃO NUMÉRICA DA EQUAÇÃO DE POISSON EM MALHAS ESTRUTURADAS BIDIMENSIONAIS E TRIDIMENSIONAIS

NUMERICAL SOLUTION OF POISSON EQUATION IN BIDIMESIONAL AND THREE-DIMENSIONAL STRUCTURED MESHES

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Matemática da Universidade Tecnológica Federal do Paraná (UTFPR), Campus Curitiba, para obtenção do título de Licenciado em Matemática.
Orientador: Rudimar Luiz Nós

CURITIBA

2021

JOÃO PEDRO SANTOS BRITO MICHELETTI

SOLUÇÃO NUMÉRICA DA EQUAÇÃO DE POISSON EM MALHAS ESTRUTURADAS BIDIMENSIONAIS E TRIDIMENSIONAIS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Licenciado em Matemática da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 23/agosto/2021

_____

Rudimar Luiz Nós
Doutor
Universidade Tecnológica Federal do Paraná

_____

Nara Bobko
Doutora
Universidade Tecnológica Federal do Paraná

_____

Júlio César Santos Sampaio
Doutor
Universidade Tecnológica Federal do Paraná

CURITIBA

2021

In memoriam
Mario Micheletti (1933-2021)

AGRADECIMENTOS

Ao professor Rudimar, por toda ajuda, paciência e companheirismo na elaboração deste trabalho.

Aos meus pais e ao meu irmão, que mesmo em momentos de dificuldades nunca deixaram de buscar um futuro melhor.

À Lúcia e à Rosângela, por todo apoio cultural que me propiciaram durante a minha infância.

À professora Neusa, que sempre me ajudou e incentivou nos estudos.

À professora Nara, pelas várias conversas sobre (quase) tudo.

Ao Cleber e à Elisandra, pelas longas conversas acerca da profissão de professor.

Ao professor Scott MacKenzie, da York University, que me ajudou com o Matlab mesmo não me conhecendo.

Aos meus professores da UTFPR, por todo o aprendizado que me proporcionaram.

Aos meus professores do Colégio Estadual Emílio de Menezes. Em especial: Cleber Dias de Araújo, Eliza Mendes Proeliza, Fabio Luiz de Souza, João Maurício, Juliano Santos, Patricia Baptista e Valeska Alves.

Aos meus amigos, que me fizeram viver ótimos momentos.

O peso da evidência de uma afirmação extraordinária deve ser proporcional à sua estranheza.
Pierre-Simon Laplace (1749-1827): matemático, astrônomo e físico francês.

RESUMO

MICHELETTI, J. P. S. B. **Solução numérica da equação de Poisson em malhas estruturadas bidimensionais e tridimensionais**. 111 f. Trabalho de Conclusão de Curso - Curso de Licenciatura em Matemática, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Neste trabalho, solucionamos numericamente equações diferenciais parciais elípticas de segunda ordem, como as equações de Laplace e de Poisson, empregando o método de diferenças finitas em malhas estruturadas bidimensionais e tridimensionais. Para solucionar o sistema de equações lineares proveniente da discretização por diferenças finitas, usamos os métodos iterativos de Gauss-Seidel e SOR. Além disso, construímos soluções manufaturadas para algumas equações de Poisson, comparamos as soluções exata e numérica e testamos valores ótimos para o parâmetro de relaxação no método SOR. Também aplicamos a teoria estudada na solução numérica de problemas estacionários ou de equilíbrio e utilizamos o Matlab e o Tecplot 360 para visualizar a solução numérica. Concluímos que a convergência do método SOR é lenta em problemas com condições de contorno de Neumann e em problemas com singularidades.

Palavras-chave: Equação de Laplace; Método de diferenças finitas; Problemas estacionários; Matlab; Tecplot 360.

ABSTRACT

MICHELETTI, J. P. S. B. **Numerical solution of Poisson equation in bidimensional and three-dimensional structured meshes**. 111 pg. Monograph - Curso de Licenciatura em Matemática, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

In this work, we numerically solve second-order elliptic partial differential equations such as the Laplace and Poisson equations, using the finite difference method in two-dimensional and three-dimensional structured meshes. To solve the system of linear equations arising from the finite difference discretization, we use the iterative methods of Gauss-Seidel and SOR. Furthermore, we build manufactured solutions for some Poisson equations and compare the exact and numerical solutions, and test optimal values for the relaxation parameter in the SOR method. We also apply the theory studied in the numerical solution of stationary or equilibrium problems and employ Matlab and Tecplot 360 to visualize the numerical solution. We conclude that the convergence of the SOR method is slow in problems with Neumann boundary conditions and in problems with singularities.

Keywords: Laplace's equation; Finite difference method; Steady state problems; Matlab; Tecplot 360.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A teoria de equações diferenciais fundamenta modelos matemáticos que descrevem diversos fenômenos naturais, físicos e biológicos, tais como meteorologia, desintegração radioativa, escoamento de fluidos, resfriamento de um corpo, propagação de doenças infecciosas e dispersão de poluentes. As equações diferenciais presentes nesses modelos permitem fazer previsões sobre como os fenômenos se comportam em diversas circunstâncias através do estudo da variação de parâmetros, enquanto que em um experimento a observação dessas variações pode levar muito tempo, ser muito cara ou até mesmo impossível (BOYCE; DIPRIMA, 2011).

1.1 EQUAÇÕES DIFERENCIAIS

Uma equação diferencial é uma igualdade que relaciona uma função (ou funções) com sua(s) derivada(s), onde a incógnita é uma função (ou funções). Pode-se classificar as equações diferenciais quanto à ordem, à linearidade e à homogeneidade:

1. a ordem é dada pela derivada de maior ordem presente na equação diferencial;
2. a linearidade é definida pela dependência linear (grau um) das funções incógnitas e das derivadas dessas funções;
3. a homogeneidade é estabelecida quando o termo que independe das funções incógnitas e de suas derivadas é identicamente nulo.

Há dois tipos de equações diferenciais: as ordinárias e as parciais.

Uma equação diferencial ordinária (EDO) de ordem n é uma igualdade da forma

$$F[t, u(t), u'(t), ..., u^{(n)}(t)] = 0, \tag{1.1}$$

onde F é uma função que relaciona a variável independente t, a função incógnita u(t) e suas n primeiras derivadas (BOYCE; DIPRIMA, 2011). Assim, em uma EDO a função incógnita depende de uma única variável independente e as derivadas são simples, isto é, relativas à única variável independente. Toda função $u(t)$ que satisfaz a igualdade (1.1) é uma solução da EDO.

Exemplo 1.1. *Uma partícula se desloca sobre o eixo x de modo que, em cada instante de tempo t, a velocidade é o dobro da posição. Qual a função que descreve a posição da partícula no instante t? Da Física, sabemos que a derivada de uma função $x(t)$, que descreve a posição de uma partícula em relação ao tempo, é a função que descreve a velocidade $v(t)$ da partícula em relação ao tempo. Dessa forma, temos que*

$$v(t) = \frac{d}{dt}x(t) = 2x(t), \;\; t > 0. \tag{1.2}$$

A solução da EDO (1.2)*, de primeira ordem, linear e homogênea, é dada pela família de funções* $x(t) = ke^{2t}$*,* $k \in \mathbb{R}$*, obtida separando-se as variáveis (GUIDORIZZI, 2018).*

Em contrapartida, uma equação diferencial parcial (EDP) em n variáveis independentes $x_1, ..., x_n$ é uma igualdade da forma

$$F\left(x_1, ..., x_n, u, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1^2}, ..., \frac{\partial^2 u}{\partial x_1 \partial x_n}, ..., \frac{\partial^k u}{\partial x_n^k}\right) = 0, \tag{1.3}$$

onde $x = (x_1, ..., x_n) \in \Omega$, Ω é um subconjunto aberto de $\mathbb{R}^n$, F é uma função dada e $u = u(x)$ é a função incógnita (IÓRIO, 1989). Portanto, em uma EDP a função incógnita (ou funções incógnitas) depende de duas ou mais variáveis independentes e as derivadas são parciais, ou seja, relativas a pelo menos duas variáveis independentes. Toda função $u\left(x_1, x_2, \ldots, x_n\right)$ que satisfaz a igualdade (1.3) é uma solução da EDP.

Exemplo 1.2. *A equação unidimensional do calor*

$$\frac{\partial}{\partial t}u(x,t) = K\frac{\partial^2}{\partial x^2}u(x,t), \;\; x \in [a,b], \;\; t > 0, \tag{1.4}$$

onde $u(x,t)$ *é a temperatura em um ponto* x *de uma barra no instante de tempo* t*, sendo a barra constituída de um material condutor uniforme de calor e cuja superfície lateral está termicamente isolada, ou seja, a superfície lateral da barra não troca calor com o meio ambiente, e* K *é a constante de difusibilidade térmica do material do qual a barra é constituída. Na estruturação da equação* (1.4)*, considera-se a lei do resfriamento de Fourier, a quantidade de calor em função do calor específico e a função* $u(x,t)$*, que descreve a temperatura, contínua com derivadas parciais contínuas até a segunda ordem (FIGUEIREDO, 1997).*

A EDP (1.4) é uma equação de segunda ordem, parabólica, linear e homogênea. Dada uma equação diferencial parcial de segunda ordem, como a equação (1.4), em duas variáveis x e y do tipo

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu = G, \tag{1.5}$$

os valores de A, B e C definem as três categorias de EDPs de segunda ordem. Assim, a EDP de segunda ordem é:

1. elíptica, se $B^2 - 4AC < 0$;
2. parabólica, se $B^2 - 4AC = 0$;
3. hiperbólica, se $B^2 - 4AC > 0$.

A solução da equação do calor (1.4) pode ser obtida empregando-se o método de separação de variáveis e a expansão de $u(x,t)$ em série de Fourier (NÓS, 2019). Para tanto, é preciso impor à equação (1.4) condições auxiliares para garantir a unicidade de solução, isto é,

para determinar uma solução particular da família de soluções da EDP. Existem dois tipos de condições auxiliares.

1. Condições iniciais: uma das variáveis independentes é fixada e são impostos o valor da solução e das derivadas parciais em relação a essa variável fixa como função das outras variáveis. Por exemplo:

$$\begin{aligned} u(x,0) =& f(x); \\ \frac{\partial}{\partial t}u(x,0) =& g(x), \end{aligned}$$

   onde $f(x)$ e $g(x)$ são funções dadas.

2. Condições de contorno: condições sobre o valor da solução e suas derivadas são impostas nas fronteiras do domínio espacial. São condições do tipo

$$\alpha u(x) + \beta \frac{\partial}{\partial n}u(x) = f(x), x \in \partial\Omega, \tag{1.6}$$

   onde α e β são constantes dadas, $f(x)$ é uma função dada no bordo $\partial\Omega$ da região Ω e $\frac{\partial}{\partial n}u(x)$ é a derivada de $u(x)$ na direção normal à $\partial\Omega$. Quando $\beta = 0$, a condição (1.6) é denominada *condição de Dirichlet*[1]; quando $\alpha = 0$, *condição de Neumann*[2]; quando $\alpha, \beta \neq 0$, *condição de Robin*[3].

As condições auxiliares definem se os fenômenos modelados por EDPs são problemas de valor inicial, problemas de contorno ou problemas mistos. Esses fenômenos podem ser classificados em dois tipos elementares de fenômenos físicos: os transientes, que evoluem com o tempo; os estacionários, que estão em um estado de equilíbrio e não evoluem mais com o passar do tempo (FORTUNA, 2000).

Problemas de equilíbrio são aqueles nos quais a propriedade de interesse não se altera com o passar do tempo. Geralmente, esses problemas são modelados por EDPs elípticas da forma

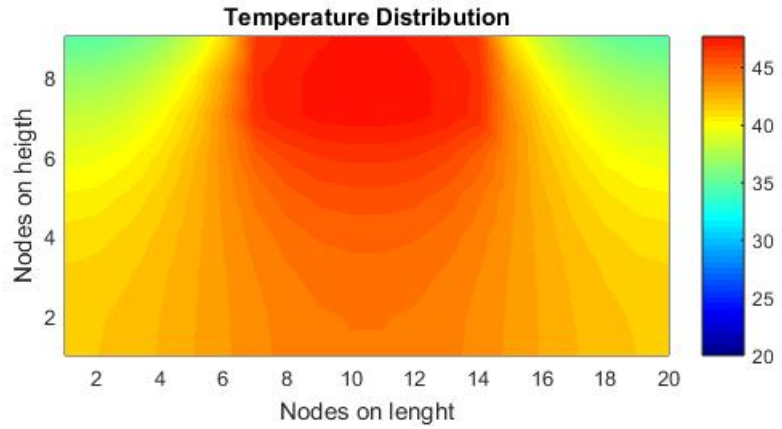
$$\nabla u = \frac{\partial^2 u}{\partial x_1^2} + ... + \frac{\partial^2 u}{\partial x_n^2} = f, \tag{1.7}$$

onde $u = u(x_1, ..., x_n)$, $f = f(x_1, ..., x_n)$ e ∇ denota o operador Laplaciano. Se $f = 0$, a equação (1.7) é denominada equação de Laplace[4] e as funções u que a satisfazem são *funções harmônicas*; se $f \neq 0$, a equação (1.7) é denominada equação de Poisson[5].

Segundo Iório (1989), a equação de Laplace modela vários problemas da Física Matemática, entre eles o estudo de campos eletrostáticos, a energia de uma partícula sobre a qual agem apenas forças gravitacionais e a distribuição de temperatura em estado estacionário. A Figura 1.1 ilustra a distribuição de temperatura fornecida por uma EDP elíptica.

[1] Peter Gustav Lejeune Dirichlet (1805-1859): matemático alemão.
[2] John von Neumann (1903-1957): matemático, físico e engenheiro húngaro-americano.
[3] Victor Gustave Robin (1855-1897): matemático aplicado francês.
[4] Pierre-Simon Laplace (1749-1827): matemático, astrônomo e físico francês.
[5] Siméon Denis Poisson (1781-1840): matemático e engenheiro francês.

Figura 1.1 – Distribuição de temperatura em um canal retangular preenchido por um substrato que circunda um chip



Fonte: Gpsgui (2019).

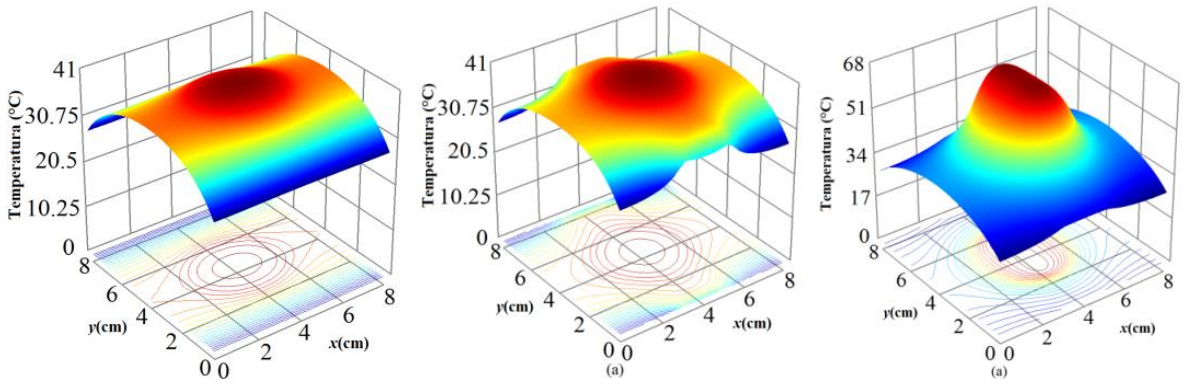
Os problemas transientes "envolvem a variação temporal das grandezas físicas de interesse" (FORTUNA, 2000, p. 52), e podem evoluir ou não para um estado estacionário. Esses problemas são modelados por EDPs parabólicas ou hiperbólicas, como a equação transiente de difusão de calor (1.8) e a equação da onda (1.9), respectivamente.

$$\frac{\partial}{\partial t}T(x,t) = \alpha \frac{\partial^2}{\partial x^2}T(x,t); \tag{1.8}$$

$$\frac{\partial^2}{\partial t^2}\phi(x,t) = v^2 \frac{\partial^2}{\partial x^2}\phi(x,t). \tag{1.9}$$

Na equação (1.8), $T(t,x)$ é a função que descreve a temperatura e α é o coeficiente de difusibilidade térmica do material; na equação (1.9), $\phi(x,t)$ descreve a posição de um ponto de uma corda que vibra e v é uma constante relacionada à velocidade de propagação de ondas na corda esticada. A Figura 1.2 ilustra a difusão de temperatura modelada por uma EDP parabólica.

Figura 1.2 – Distribuição de temperatura em tecido biológico com um tumor: (a) sem campo eletromagnético e sem nanopartículas; (b) com indução de campo eletromagnético; (c) com indução de campo eletromagnético e nanopartículas

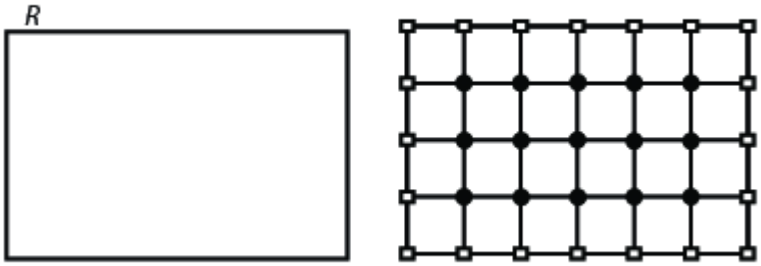


(a) (b) (c)

Fonte: Varón (2019) .

A solução exata ou analítica das EDPs que modelam fenômenos estacionários ou fenômenos transientes permite determinar o valor da propriedade física de interesse em qualquer ponto do domínio espacial de observação do fenômeno. Contudo, não há técnicas de solução analítica para grande parte das EDPs, como por exemplo, para as equações de Navier-Stokes, que são equações diferenciais parciais não lineares que simulam os campos de velocidade e de pressão em um escoamento. Assim, é necessário empregar métodos numéricos para aproximar a solução dessas EDPs com o auxílio de computadores. Na solução numérica, não é mais possível determinar o valor da propriedade física de interesse em qualquer ponto do domínio espacial, ou seja, em uma região contínua, mas somente em um conjunto de pontos do domínio previamente escolhidos, isto é, em uma região discreta. Essa região discreta, ou domínio espacial discreto, é denominada malha. A Figura 1.3 ilustra a discretização de uma região contínua R.

Figura 1.3 – Discretização da região *R*: • pontos interiores da malha; □ pontos de fronteira da malha



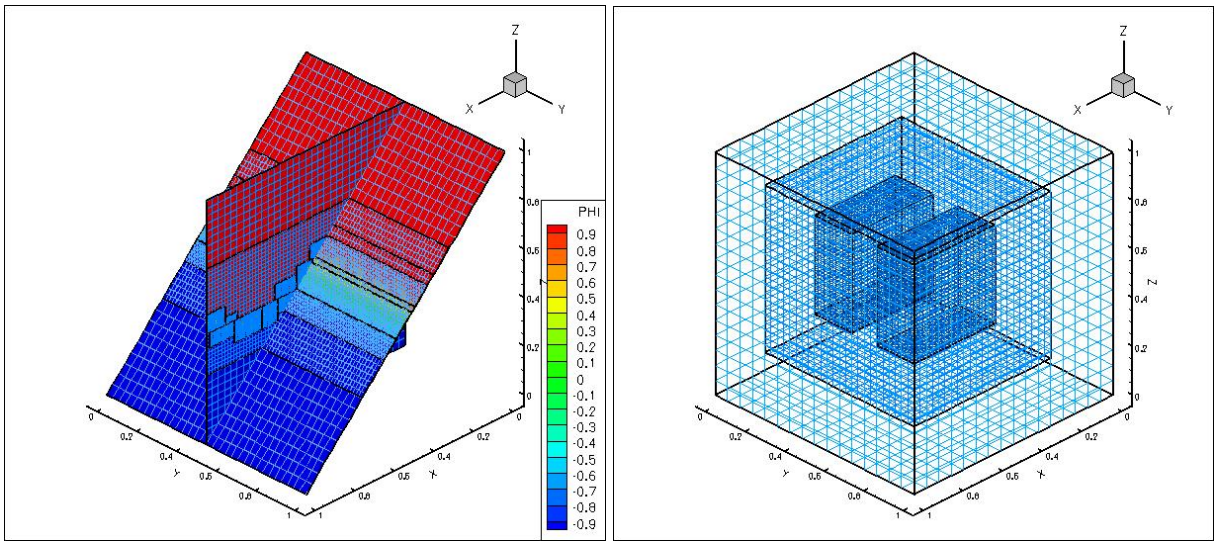
Fonte: Almeida (2017) .

Dessa maneira, para solucionar numericamente uma EDP precisamos inicialmente escrever seus termos em função dos pontos da malha. Neste processo, podemos utilizar malhas estruturadas ou malhas não estruturadas.

As malhas estruturadas apresentam uma estrutura regular na distribuição dos pontos (FORTUNA, 2000), sendo que cada ponto interno possui o mesmo número de pontos vizinhos (MALISKA, 2004). Esse tipo de malha facilita a programação numérica devido à distribuição matricial dos pontos (MALISKA, 2004). As células computacionais são quadriláteros em malhas estruturadas bidimensionais - Figura 1.4(a), e hexaedros em malhas estruturadas tridimensionais - Figura 1.4(b). Nessas malhas, cada célula tem o mesmo número de células vizinhas, exceto quando a célula pertence à fronteira (ou contorno). Os nós de cada célula são identificados pelos índices i e j, em duas dimensões, e pelos índices i, j e k, em três dimensões.

Entretanto, devido à complexidade da geometria relativa ao domínio espacial do fenômeno em observação, nem sempre é possível empregar uma malha estruturada. As malhas não estruturadas são mais versáteis, possuem adaptabilidade e são mais apropriadas para discretizar geometrias irregulares, sendo que em alguns problemas apenas malhas não estruturadas conseguem discretizar devidamente o domínio espacial (MALISKA, 2004). Malhas não estruturadas

Figura 1.4 – Malhas estruturadas: (a) bidimensional; (b) tridimensional



(a) (b)

Fonte: Nós (2007).

não obedecem a nenhuma lei de construção e os pontos internos não possuem o mesmo número de pontos vizinhos. Nessas malhas, as células computacionais são geralmente triangulares em duas dimensões - Figura 1.5(a), e tetraédricas em três dimensões - Figura 1.5(b). Diferentemente das malhas estruturadas, os nós não podem ser identificados de forma única pelos índices i e j, em duas dimensões, e pelos índices i, j e k, em três dimensões. Malhas não estruturadas têm, geralmente, mais células computacionais do que malhas estruturadas.

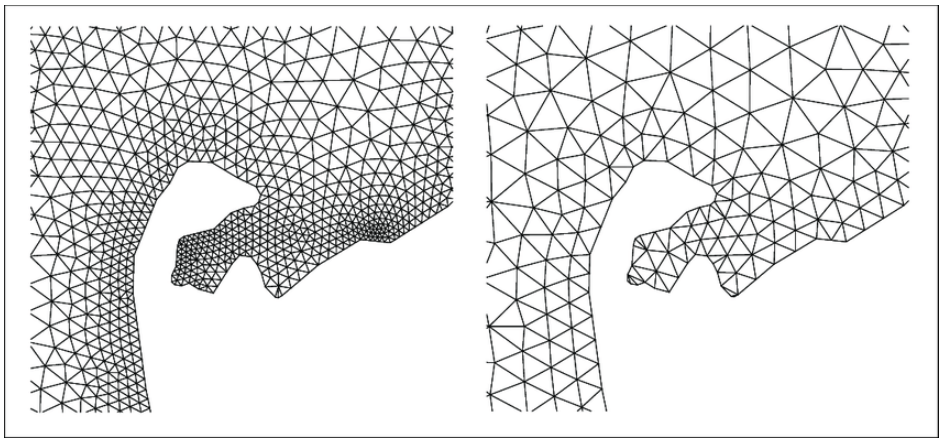
Dessa forma, malhas não estruturadas apresentam duas desvantagens: a discretização do domínio espacial, geralmente complicada devido à falta de regularidade na distribuição dos pontos; o tempo computacional para a solução das equações associadas à discretização. Para contornar essas desvantagens, podemos empregar malhas híbridas - Figura 1.5(b), que são uma junção de malhas estruturadas com malhas não estruturadas. O emprego de malhas híbridas se justifica pelo fato de que, dependendo da propriedade física observada, a solução numérica deve ser mais precisa ou pode ser menos precisa em diferentes regiões do domínio computacional.

A escolha da malha está atrelada à natureza do fenômeno a ser solucionado numericamente (MALISKA, 2004), e consequentemente à estratégia adotada para discretizar a EDP. Na discretização da EDP, podemos empregar diferenças finitas, volumes finitos e elementos finitos.

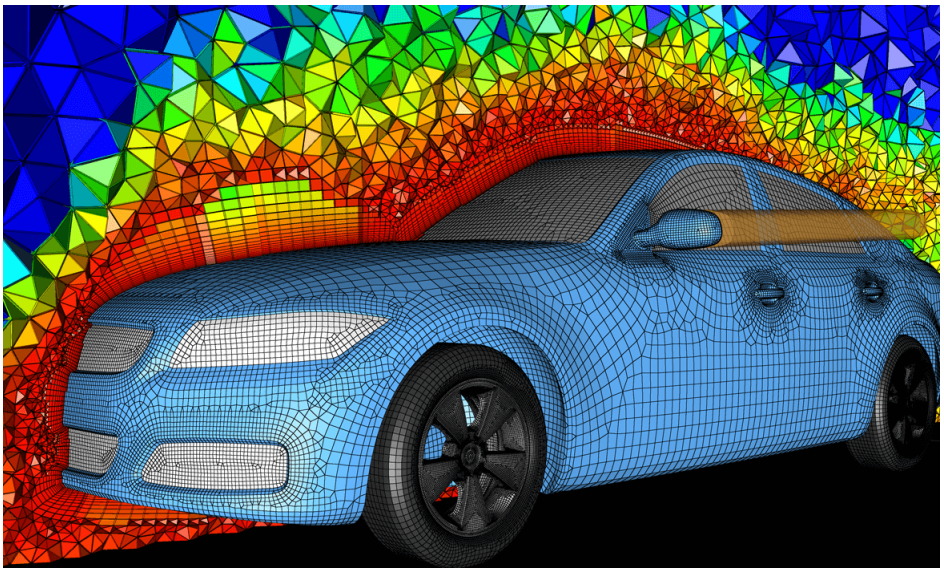
1. No método de diferenças finitas, as derivadas da EDP são substituídas por diferenças algébricas obtidas através da expansão em série de Taylor das variáveis da solução em vários pontos vizinhos ao ponto de avaliação (THOMPSON; SONI; WEATHERILL, 1999).

2. No método de volumes finitos, as variáveis de solução são consideradas constantes dentro da célula de controle e os fluxos através dos lados (ou faces) da célula (que separam valores

Figura 1.5 – Malhas não estruturadas: (a) bidimensional; (b) tridimensional híbrida



(a)



(b)

Fonte: (a) Researchgate (2019); (b) Pointwise (2019).

descontínuos da solução) são calculados mais precisamente com um procedimento que representa a dissolução de tal descontinuidade durante o intervalo de tempo (THOMPSON; SONI; WEATHERILL, 1999).

3. No método de elementos finitos, há duas formas básicas de aproximação (THOMPSON; SONI; WEATHERILL, 1999):

   a) a variacional, onde as EDPs são substituídas por um princípio variacional integral mais fundamental (a partir do qual elas surgem através do cálculo de variações);

   b) a residual ponderada, na qual as EDPs são multiplicadas por certas funções e depois integradas sobre a célula computacional.

O método de diferenças finitas é inerente a malhas estruturadas; o método de volumes

finitos pode ser aplicado em malhas estruturadas e não estruturadas; o método de elementos finitos é intrínseco a malhas não estruturadas.

Neste trabalho, solucionamos numericamente EDPs elípticas, ou seja, EDPs que não apresentam termos do tipo

$$\frac{\partial^n \phi}{\partial t^n}, \ n \geq 1,$$

onde t é a variável temporal e ϕ é uma propriedade física de interesse, como temperatura por exemplo, usando o método de diferenças finitas em malhas blocoestruturadas bidimensionais e tridimensionais.

Exemplo 1.3. *A discretização por diferenças finitas centradas de segunda ordem (FORTUNA, 2000) da equação de Laplace bidimensional*

$$\frac{\partial^2}{\partial x^2} u(x,y) + \frac{\partial^2}{\partial y^2} u(x,y) = 0, \tag{1.10}$$

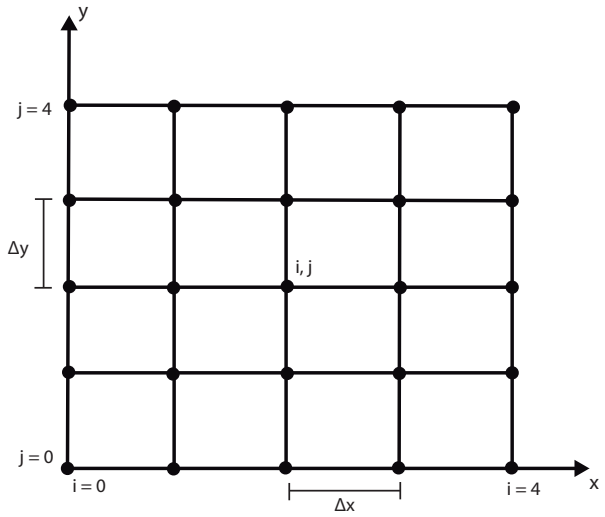
é dada por

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + \left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0, \tag{1.11}$$

onde i, j representa o ponto (x_i, y_j).

Para solucionar numericamente a equação (1.10) *no domínio discreto uniforme $(\Delta x = \Delta y)$ dado pela Figura 1.6, devemos aplicar* (1.11) *aos nove pontos interiores da malha uniforme.*

Figura 1.6 – Domínio para a discretização da equação de Laplace bidimensional



Fonte: Almeida (2017) .

Dessa forma, denotando $\Delta x = \Delta y = \beta$*, obtemos:*

$$\textit{Em } (1,1): u_{2,1} + u_{0,1} - 2\left(1+\beta^2\right)u_{1,1} + \beta^2\left(u_{1,2} + u_{1,0}\right) = 0; \tag{1.12}$$

$$\textit{Em } (2,1): u_{3,1} + u_{1,1} - 2\left(1+\beta^2\right)u_{2,1} + \beta^2\left(u_{2,2} + u_{2,0}\right) = 0; \tag{1.13}$$

$$\textit{Em } (3,1): u_{4,1} + u_{2,1} - 2\left(1+\beta^2\right)u_{3,1} + \beta^2\left(u_{3,2} + u_{3,0}\right) = 0; \tag{1.14}$$

$$\textit{Em } (1,2): u_{2,2} + u_{0,2} - 2\left(1+\beta^2\right)u_{1,2} + \beta^2\left(u_{1,3} + u_{1,1}\right) = 0; \tag{1.15}$$

$$\textit{Em } (2,2): u_{3,2} + u_{1,2} - 2\left(1+\beta^2\right)u_{2,2} + \beta^2\left(u_{2,3} + u_{2,1}\right) = 0; \tag{1.16}$$

$$\textit{Em } (3,2): u_{4,2} + u_{2,2} - 2\left(1+\beta^2\right)u_{3,2} + \beta^2\left(u_{3,3} + u_{3,1}\right) = 0; \tag{1.17}$$

$$\textit{Em } (1,3): u_{2,3} + u_{0,3} - 2\left(1+\beta^2\right)u_{1,3} + \beta^2\left(u_{1,4} + u_{1,2}\right) = 0; \tag{1.18}$$

$$\textit{Em } (2,3): u_{3,3} + u_{1,3} - 2\left(1+\beta^2\right)u_{2,3} + \beta^2\left(u_{2,4} + u_{2,2}\right) = 0; \tag{1.19}$$

$$\textit{Em } (3,3): u_{4,3} + u_{2,3} - 2\left(1+\beta^2\right)u_{3,3} + \beta^2\left(u_{3,4} + u_{3,2}\right) = 0. \tag{1.20}$$

Nas equações (1.12)-(1.20)*, os termos em azul correspondem aos pontos de fronteira. Definidas as condições de contorno, as equações* (1.12)-(1.20) *caracterizam um sistema linear cuja solução fornece o valor de* $u(x, y)$ *em cada um dos nove pontos interiores da malha.*

Para aproximar a solução de sistemas lineares como o definido pelas equações (1.12)-(1.20), empregamos dois métodos numéricos iterativos: o método de Gauss[6]-Seidel[7] e o método SOR (Successive Over-Relaxation) (BURDEN; FAIRES; BURDEN, 2016; FERZIGER, 1981; GERALD; WHEATLEY, 1994; HUMES et al., 1984; SCHWARZ, 1989).

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

Solucionar numericamente, em duas e três dimensões, problemas de equilíbrio modelados por equações diferenciais parciais elípticas empregando o método de diferenças finitas em malhas estruturadas.

1.2.2 OBJETIVOS ESPECÍFICOS

- Construir soluções manufaturadas para equações diferenciais parciais.
- Discretizar as equações de Laplace e de Poisson empregando diferenças finitas centradas de segunda ordem.

[6] Carl Friedrich Gauss (1777-1855): matemático, astrônomo e físico alemão; um dos mais influentes matemáticos na história da matemática.

[7] Philipp Ludwig von Seidel (1821-1896): matemático alemão.

- Solucionar o sistema linear proveniente da discretização das equações de Laplace e de Poisson através de dois métodos numéricos iterativos: o método de Gauss-Seidel e o método SOR.
- Testar valores ótimos de relaxação para o método SOR.
- Construir códigos em Linguagem C para solucionar numericamente, em duas e em três dimensões, as equações de Laplace e de Poisson.
- Utilizar um aplicativo gráfico, como o matlab por exemplo, para visualizar as soluções exata e numérica das equações de Laplace e de Poisson.

1.3 PROCEDIMENTOS METODOLÓGICOS

Os procedimentos metodológicos que adotamos são os seguintes:

- Estudo de equações diferenciais parciais (FORTUNA, 2000; IÓRIO, 1989; JOHN, 1982; NÓS, 2019);
- Estudo de métodos de discretização, particularmente o método de diferenças finitas (FORTUNA, 2000; STRIKWERDA, 1989);
- Estudo de métodos iterativos para a solução de sistemas de equações lineares, particularmente os métodos de Gauss-Seidel e SOR (BURDEN; FAIRES; BURDEN, 2016; FERZIGER, 1981; GERALD; WHEATLEY, 1994; HUMES et al., 1984; SCHWARZ, 1989);
- Estudo da Linguagem C (SCHILDT, 1997);
- Uso do matlab para visualição de curvas e de superfícies (MATLAB, 2021);
- Uso do overleaf para a escrita do texto em latex (OVERLEAF, 2019).

1.4 ESTRUTURA DO TRABALHO

O trabalho está organizado em cinco capítulos e dois apêndices: no primeiro capítulo, fazemos uma breve introdução ao tema do TCC, descrevemos os objetivos do trabalho e os procedimentos metológicos adotados, definimos uma equação diferencial parcial, classificando-a quanto à linearidade, ordem e homogeneidade, e apresentamos as equações de Laplace e de Poisson, bem como problemas modelados por essas equações; no segundo capítulo, introduzimos o método de diferenças finitas, empregando-o para discretizar as equações de Laplace e de Poisson, assim como os métodos numéricos iterativos de Gauss-Seidel e SOR para solucionar os sistemas de equações lineares proveniente das discretizações; no terceiro capítulo, construímos soluções

manufaturadas para as equações de Laplace e de Poisson; no quarto capítulo, solucionamos numericamente problemas de equilíbrio; no quinto capítulo, fazemos as considerações finais; nos apêndices, elencamos conceitos pertinentes ao trabalho, assim como os códigos computacionais em linguagem C utilizados nas simulações do terceiro e do quarto capítulos.

2 DISCRETIZAÇÃO DA EDP

A solução numérica de uma EDP inicia com o processo de discretização.

2.1 POLINÔMIO DE TAYLOR

O método das diferenças finitas, empregado na solução numérica de EDPs, tem como base a fórmula de Taylor[1] com resto de Lagrange[2]. Apresentam-se a seguir alguns teoremas relativos à fórmula de Taylor, cujas demonstrações podem ser encontradas em Guidorizzi (2018).

Teorema 2.1. *Se u é uma função diferenciável até a segunda ordem no intervalo I, então existe pelo menos um $\overline{x}$ entre x_0 e x, com $x_0, x \in I$, tal que*

$$u(x) = u(x_0) + u'(x_0)(x - x_0) + \frac{u''(\overline{x})}{2}(x - x_0)^2. \tag{2.1}$$

Na igualdade (2.1), denomina-se

$$P(x) = u(x_0) + u'(x_0)(x - x_0)$$

de polinômio de Taylor de ordem um da função u em torno de x_0, sendo

$$E(x) = \frac{u''(\overline{x})}{2}(x - x_0)^2$$

o erro cometido ao se aproximar $u(x)$ por $P(x)$.

Se não é possível calcular exatamente o valor de $u(x)$, também não é possível calcular qual o valor de $\overline{x}$, por isso a motivação da aproximação. Também têm-se que, quanto mais próximo x está de x_0, menor será $(x - x_0)^2$ e, por consequência, $E(x)$.

Para melhorar a aproximação, podemos utilizar o polinômio de Taylor de ordem dois.

Teorema 2.2. *Se u é uma função diferenciável até a terceira ordem no intervalo I, então existe pelo menos um $\overline{x}$ entre x_0 e x, com $x_0, x \in I$, tal que*

$$u(x) = u(x_0) + u'(x_0)(x - x_0) + \frac{u''(x_0)}{2}(x - x_0)^2 + \frac{u'''(\overline{x})}{3!}(x - x_0)^3. \tag{2.2}$$

Em (2.2),

$$u(x) = u(x_0) + u'(x_0)(x - x_0) + \frac{u''(x_0)}{2}(x - x_0)^2$$

[1] Brook Taylor (1685-1731): matemático britânico, membro da Royal Society.
[2] Joseph-Louis Lagrange (1736-1813): matemático e astrônomo italiano, posteriormente naturalizado francês.

é o polinômio de Taylor de ordem dois da função u em torno de x_0 e

$$E(x) = \frac{u'''(\overline{x})}{3!}(x - x_0)^3$$

é o erro cometido ao se aproximar $u(x)$ por $P(x)$.

Para x na vizinhança de x_0, temos que

$$(x - x_0)^3 < (x - x_0)^2.$$

Portanto, o erro cometido ao se aproximar uma função pelo polinômio de Taylor de ordem dois é geralmente menor do que o erro cometido ao se aproximar pelo polinômio de Taylor de ordem um.

Exemplo 2.1. *Aproximar o valor de $ln(1,05)$ utilizando os polinômios de Taylor de ordem um e de ordem dois da função $u(x) = ln(x)$.*

Consideremos $x_0 = 1$. Assim, pelo polinômio de Taylor de ordem um, temos que:

$$\begin{aligned} P(x) =& u(1) + u'(1)(x - 1); \\ P(x) =& x - 1; \\ ln(1,05) \approx& P(1,05) = 1,05 - 1 = 0,05. \end{aligned} \tag{2.3}$$

Empregando o polinômio de Taylor de ordem 2, obtemos que:

$$\begin{aligned} P(x) =& u(1) + u'(1)(x - 1) + \frac{u''(1)}{2}(x - 1)^2; \\ P(x) =& (x - 1) - \frac{1}{2}(x - 1)^2; \\ ln(1,05) \approx& P(1,05) = 0,05 - 0,00125 = 0,04875. \end{aligned} \tag{2.4}$$

Comparando as aproximações com o valor de $ln(1,05)$, temos em (2.3) *um erro menor do que 10^{-2} e, em* (2.4)*, um erro menor do que 10^{-4}.*

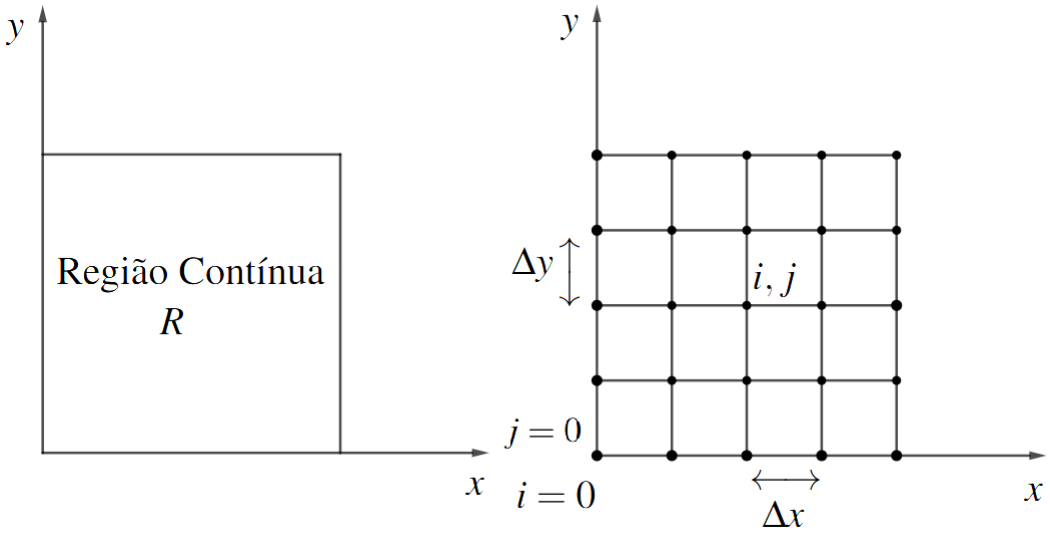
2.2 MÉTODO DE DIFERENÇAS FINITAS

A solução de uma EDP em uma região contínua *R* implica em se obter valores para a função incógnita em todos os pontos de *R*. Porém, numericamente, o computador pode calcular somente a solução em uma quantidade finita de pontos dessa região. Este conjunto de pontos onde a solução é calculada denomina-se malha (FORTUNA, 2000). A Figura 2.1 ilustra a discretização de uma região contínua *R*.

Os pontos da malha na Figura 2.1 estão localizados na intersecção das retas paralelas ao eixo x com as retas paralelas ao eixo y, separados respectivamente por uma distância Δx e Δy, não necessariamente iguais. Assim, um determinado ponto (i, j) têm coordenadas

$$(x_0 + i\Delta x, y_0 + j\Delta y),$$

Figura 2.1 – Domínio contínuo discretizado



Fonte: O Autor baseado em Fortuna (2000).

sendo que o ponto (x_0, y_0) representa a origem do sistema de coordenadas (do problema), que na Figura 2.1 é $(0, 0)$ (FORTUNA, 2000).

Para que possamos solucionar EDPs numericamente, estas devem ser expressas na forma de operações aritméticas que o computador possa executar (FORTUNA, 2000). Segundo Fortuna (2000, p. 74): "devemos representar os diferenciais da EDP por expressões algébricas, ou seja, *discretizar* a EDP". Consequentemente, antes de se resolver a EDP numericamente, precisamos determinar as expressões algébricas relativas aos termos da EDP, escritas em função dos pontos da malha (FORTUNA, 2000). Essas expressões algébricas são aproximações por diferenças finitas (LOGAN, 2015). A conclusão desse processo é uma equação algébrica, denominada equação de diferenças finitas ou EDF (FORTUNA, 2000). A EDF é então aplicada em cada ponto da malha em que se deseja calcular a solução do problema (FORTUNA, 2000), originando um sistema de equações lineares. Conforme Fortuna (2000), solucionando-se o sistema de EDFs, determina-se uma solução aproximada da EDP, que não é exata devido a erros:

- inerentes ao processo de discretização das equações;
- de arredondamento nos cálculos feitos pelo computador;
- na aproximação numérica das condições auxiliares.

2.3 EXPANSÕES EM SÉRIE DE TAYLOR

A ferramenta básica para aproximar as derivadas da EDP é a série de Taylor, que estabelece uma relação entre o valor de uma função u e de suas derivadas em um ponto x com os valores de u em pontos vizinhos de x (CUMINATO; JUNIOR, 2013). Se a função u possui

derivadas até a ordem n em um intervalo $[a, b]$, com $x, x_0 \in [a, b]$, a expansão de u em série de Taylor é definida por:

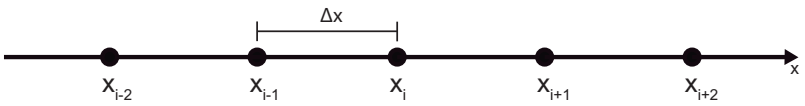
$$u(x) = u\left(x_0\right) + \Delta x \frac{du}{dx}\bigg|_{x_0} + \frac{\left(\Delta x\right)^2}{2!}\frac{d^2u}{dx^2}\bigg|_{x_0} + \frac{\left(\Delta x\right)^3}{3!}\frac{d^3u}{dx^3}\bigg|_{x_0} + \cdots + R_n,$$

onde $\Delta x = x - x_0$ e R_n é o resto dado por

$$R_n = \frac{\left(\Delta x\right)^n}{n!}\frac{d^nu}{dx^n}\bigg|_{\xi}, \xi \in [a, b].$$

Dessa forma, seja uma malha unidimensional com pontos uniformemente espaçados ($\Delta x = x_i - x_{i-1}$), como representado na Figura 2.2.

Figura 2.2 – Malha unidimensional com pontos uniformemente espaçados



Fonte: Almeida (2017).

Para discretizar, pelo método de diferenças finitas, a primeira derivada de uma função u no ponto $x_i = i\Delta x$, denotada por $\left.\frac{du}{dx}\right|_i$, devemos expandir a série de Taylor de u em torno de x_i, obtendo:

$$u\left(x_i + \Delta x\right) = u\left(x_i\right) + \Delta x \left.\frac{du}{dx}\right|_i + \frac{\left(\Delta x\right)^2}{2!}\left.\frac{d^2u}{dx^2}\right|_i + \frac{\left(\Delta x\right)^3}{3!}\left.\frac{d^3u}{dx^3}\right|_i + \cdots + R_n. \tag{2.5}$$

Ao isolar a primeira derivada em (2.5), temos que:

$$\left.\frac{du}{dx}\right|_i = \frac{u\left(x_i + \Delta x\right) - u(x_i)}{\Delta x} + \left[-\frac{\Delta x}{2!}\left.\frac{d^2u}{dx^2}\right|_i - \frac{\left(\Delta x\right)^2}{3!}\left.\frac{d^3u}{dx^3}\right|_i - \cdots\right]. \tag{2.6}$$

A expressão (2.6) mostra que a primeira derivada é igual a

$$\frac{u\left(x_i + \Delta x\right) - u\left(x_i\right)}{\Delta x},$$

mais os termos da série de Taylor até R_n. Estes termos constituem o *erro local de truncamento* (ELT):

$$ELT = -\frac{\Delta x}{2!}\left.\frac{d^2u}{dx^2}\right|_i - \frac{\left(\Delta x\right)^2}{3!}\left.\frac{d^3u}{dx^3}\right|_i - \cdots. \tag{2.7}$$

O ELT (2.7) surge devido à utilização de um número finito de termos na série de Taylor. Esse erro estabelece a diferença entre o valor exato da derivada e sua aproximação numérica, indicando como essa diferença varia com a diminuição do espaçamento Δx, ou seja, com o refinamento da malha (FORTUNA, 2000).

2.3.1 DISCRETIZAÇÃO DAS DERIVADAS PRIMEIRA E SEGUNDA

A primeira derivada de uma função u em um ponto $x_i \in \mathbb{R}$ é dada por:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \lim_{h\to 0} \frac{u\left(x_i+h\right)-u\left(x_i\right)}{h}. \tag{2.8}$$

Uma forma de aproximar (2.8) é considerar

$$\left.\frac{du}{dx}\right|_{x=x_i} \approx \frac{u\left(x_i+h\right)-u\left(x_i\right)}{h}, \tag{2.9}$$

para algum $h \in \mathbb{R}$ suficientemente pequeno.

A aproximação (2.9) utiliza o ponto $x_i + h$ à frente de x_i, sendo denominada *diferença progressiva (ou ascendente) de primeira ordem* (CUMINATO; JUNIOR, 2013).

A aproximação (2.9) também pode ser deduzida a partir da série de Taylor. A expansão em série de Taylor da função u em $x = x_i + h$, em torno de $x = x_i$, é igual a:

$$u\left(x_i+h\right) = u\left(x_i\right) + h\left.\frac{du}{dx}\right|_{x=x_i} + \frac{h^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} + \frac{h^3}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} + \cdots . \tag{2.10}$$

Isolando a primeira derivada na equação (2.10), obtemos que:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \frac{u\left(x_i+h\right)-u\left(x_i\right)}{h} - \frac{h}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} - \frac{h^2}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} - \cdots . \tag{2.11}$$

Em (2.11), definimos um h suficiente pequeno, truncamos a série no primeiro termo e ignoramos os termos relativos às derivadas de ordem maior do que ou igual a dois. Desta forma, podemos reescrever (2.11) como:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \frac{u\left(x_i+h\right)-u\left(x_i\right)}{h} + O(h).$$

Assim,

$$\frac{u\left(x_i+h\right)-u\left(x_i\right)}{h}$$

é uma aproximação progressiva de primeira ordem para a primeira derivada da função u. A aproximação é de primeira ordem porque o menor expoente de h nos termos de $O(h)$ é igual a 1.

Analogamente, ao expandir a função u em série de Taylor em $x = x_i - h$, temos que:

$$u\left(x_i-h\right) = u\left(x_i\right) - h\left.\frac{du}{dx}\right|_{x=x_i} + \frac{h^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} - \frac{h^3}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} + \cdots . \tag{2.12}$$

Isolando a primeira derivada em (2.12), obtemos que:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \frac{u\left(x_i\right)-u\left(x_i-h\right)}{h} + \frac{h}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} - \frac{h^2}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} + \cdots . \tag{2.13}$$

Ignorando em (2.13) os termos relativos às derivadas de ordem maior do que ou igual a dois, obtemos uma aproximação por *diferença regressiva de primeira ordem* para a primeira derivada:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \frac{u\left(x_i\right) - u\left(x_i - h\right)}{h} + O(h).$$

Além das aproximações de primeira ordem, podemos obter uma aproximação de segunda ordem para a primeira derivada da função u. Subtraindo a equação (2.12) da equação (2.10), obtemos que:

$$u\left(x_i + h\right) - u\left(x_i - h\right) = 2h\left.\frac{du}{dx}\right|_{x=x_i} + 2\frac{h^3}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} + \cdots . \tag{2.14}$$

Isolando a primeira derivada em (2.14), temos que:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \frac{u\left(x_i + h\right) - u\left(x_i - h\right)}{2h} - \frac{h^2}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} - \cdots . \tag{2.15}$$

Descartando em (2.15) os termos relativos às derivadas de ordem maior do que ou igual a três, obtemos uma aproximação por *diferença centrada de segunda ordem* para a derivada primeira:

$$\left.\frac{du}{dx}\right|_{x=x_i} = \frac{u\left(x_i + h\right) - u\left(x_i - h\right)}{2h} + O(h^2), \tag{2.16}$$

onde $O(h^2)$ indica ordem 2 para a aproximação, pois o menor expoente de h nos termos descartados é 2.

Já uma aproximação para a segunda derivada da função u pode ser obtida somando-se as equações (2.10) e (2.12):

$$u\left(x_i + h\right) + u\left(x_i - h\right) = 2u\left(x_i\right) + 2\frac{h^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} + 2\frac{h^4}{4!}\left.\frac{d^4u}{dx^4}\right|_{x=x_i} + \cdots . \tag{2.17}$$

Isolando a segunda derivada da função u em (2.17), temos que:

$$\left.\frac{d^2u}{dx^2}\right|_{x=x_i} = \frac{u\left(x_i + h\right) - 2u\left(x_i\right) + u\left(x_i - h\right)}{h^2} - 2\frac{h^2}{4!}\left.\frac{d^4u}{dx^4}\right|_{x=x_i} - \cdots . \tag{2.18}$$

Ignorando em (2.18) os termos relativos às derivadas de ordem maior do que ou igual a quatro, obtemos a aproximação de segunda ordem para a derivada segunda:

$$\left.\frac{d^2u}{dx^2}\right|_{x=x_i} = \frac{u\left(x_i + h\right) - 2u\left(x_i\right) + u\left(x_i - h\right)}{h^2} + O\left(h^2\right), \tag{2.19}$$

denominada aproximação por *diferença centrada de segunda ordem*.

Em algumas situações, como na discretização da condição de Neumann para pontos da fronteira, para não precisarmos utilizar pontos fora da malha, denominados *pontos fantasmas*, usamos apenas diferenças progressivas ou regressivas. Já apresentamos as aproximações por

diferenças regressivas e progressivas para a derivada primeira, mas apenas as de primeira ordem. Podemos empregar também diferenças progressivas/regressivas de segunda ordem para a derivada primeira, o que melhora a precisão da aproximação numérica.

Uma forma de obter uma expressão de diferenças regressivas de segunda ordem para a derivada primeira (FERNANDO, 2004), é considerarmos a equação :

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{au(x_i) + bu(x_{i-1}) + cu(x_{i-2})}{h} + O(h^2), \tag{2.20}$$

onde $x_{i-1} = x_i - h$ e $x_{i-2} = x_i - 2h$, que pode ser reescrita como

$$au(x_i) + bu(x_{i-1}) + cu(x_{i-2}) = h\left.\frac{du}{dx}\right|_{x_i} + O(h^3). \tag{2.21}$$

Em (2.20), determinamos os coeficientes a, b e c calculando $u(x_{i-1})$ e $u(x_{i-2})$ através da expansão em Série de Taylor da função u em torno $x_i = x_{i-1} + h$. Assim, temos que:

$$u(x_{i-2}) = u(x_i) - 2h\left.\frac{du}{dx}\right|_{x_i} + \frac{(2h)^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x_i} + O(h^3), \tag{2.22}$$

$$u(x_{i-1}) = u(x_i) - h\left.\frac{du}{dx}\right|_{x_i} + \frac{h^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x_i} + O(h^3). \tag{2.23}$$

Multiplicando (2.22) por c, (2.23) por b e somando $au(x_i)$ a ambos os lados desta última, da adição dessas equações obtemos:

$$au(x_i)+bu(x_{i-1})+cu(x_{i-2}) = (a+b+c)u(x_i) - h(2c+b)\left.\frac{du}{dx}\right|_{x_i} + \frac{h^2}{2}(4c+b)\left.\frac{d^2u}{dx^2}\right|_{x_i} + O(h^3). \tag{2.24}$$

Comparando (2.21) e (2.24), obtemos que:

$$\begin{cases} a+b+c &= 0 \\ 2c+b &= -1 \\ 4c+b &= 0 \end{cases}. \tag{2.25}$$

A solução do sistema (2.25) é $a = \dfrac{3}{2}$, $b = -2$ e $c = \dfrac{1}{2}$. Substituindo esses valores em (2.20), concluímos que:

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{3u(x_i) - 4u(x_{i-1}) + u(x_{i-2})}{2h} + O(h^2). \tag{2.26}$$

A expressão (2.26) é uma aproximação regressiva de segunda ordem para a derivada primeira. Analogamente, podemos determinar que:

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{-3u(x_i) + 4u(x_{i+1}) - u(x_{i+2})}{2h} + O(h^2). \tag{2.27}$$

A expressão (2.27) é uma aproximação progressiva de segunda ordem para a derivada primeira.

2.4 DISCRETIZAÇÃO DE EQUAÇÕES ESTACIONÁRIAS

Equações estacionárias são equações diferenciais que não possuem termos do tipo

$$\frac{\partial^n \psi}{\partial t^n}, n \geq 1.$$

Este tipo de equação pode representar fenômenos que estão em equilíbrio, isto é, que não dependem do tempo (FORTUNA, 2000). Geralmente, esses fenômenos são modelados pela equação de Laplace, que em coordenadas cartesianas bidimensionais é dada por:

$$\nabla \psi = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0. \tag{2.28}$$

Usualmente, a equação (2.28) é discretizada por meio de diferenças centrais de segunda ordem, definidas em (2.19), ou seja,

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta y)^2} = 0. \tag{2.29}$$

A equação (2.29) pode ser reescrita como:

$$\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j} + \left(\frac{\Delta x}{\Delta y}\right)^2 (\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}) = 0. \tag{2.30}$$

Adotando em (2.30) $\epsilon = \dfrac{\Delta x}{\Delta y}$, temos que:

$$\psi_{i+1,j} + \psi_{i-1,j} + \epsilon^2 \psi_{i,j+1} + \epsilon^2 \psi_{i,j-1} - 2(1 + \epsilon^2)\psi_{i,j} = 0. \tag{2.31}$$

Ao aplicar a equação (2.31) em uma malha, obtemos um sistema linear que pode ser solucionado numericamente pelos métodos iterativos descritos a seguir.

2.5 MÉTODO DE GAUSS-SEIDEL

O método de Gauss-Seidel consiste em um processo iterativo que resolve um sistema linear $Ax = b$ calculando uma sequência $\{x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots\}$ de aproximações da solução exata do sistema, a partir de uma aproximação inicial $x^{(0)}$ (HUMES et al., 1984).

Seja o sistema linear $Ax = b$, de ordem *n*:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{cases}. \tag{2.32}$$

Fixada a ordem das equações, com $a_{ii} \neq 0, i = 1, 2, \cdots, n$, podemos reescrever as equações do sistema (2.32) como:

$$\begin{aligned}
x_1 &= \frac{b_1 - a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n}{a_{11}};\\
x_2 &= \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \ldots - a_{2n}x_n}{a_{22}};\\
x_3 &= \frac{b_3 - a_{31}x_1 - a_{32}x_2 - \ldots - a_{3n}x_n}{a_{33}};\\
&\vdots\\
x_n &= \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{n,n-1}x_{n-1}}{a_{nn}}.
\end{aligned}$$

Utilizando a aproximação inicial $x^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$, o método de Gauss-Seidel calcula a sequência de aproximações $\{x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots\}$ por intermédio das equações

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \ldots - a_{1n}x_n^{(k)}}{a_{11}}, \tag{2.33}$$

$$x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \ldots - a_{2n}x_n^{(k)}}{a_{22}}, \tag{2.34}$$

$$x_3^{(k+1)} = \frac{b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - \ldots - a_{3n}x_n^{(k)}}{a_{33}}, \tag{2.35}$$

$$\vdots$$

$$x_n^{(k+1)} = \frac{b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \ldots - a_{n,n-1}x_{n-1}^{(k+1)}}{a_{nn}}, \tag{2.36}$$

onde $k+1$ indica a iteração atual e k, a iteração anterior.

A sequência de aproximações gerada pelas equações (2.33)-(2.36) converge se, dado $\delta > 0$, existe $\overline{k}$ tal que para todo $k > \overline{k}$ e $i = 1, 2, \ldots, n$,

$$\left|x_i^{(k)} - \overline{x_i}\right| \leq \delta,$$

sendo $\overline{x} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$ a solução exata.

Contudo, como a solução exata $\overline{x}$ não é conhecida, precisamos adotar um critério de parada para o processo iterativo. Um critério possível consiste na comparação de duas aproximações consecutivas

$$x^{(k-1)} = \left(x_1^{(k-1)}, x_2^{(k-1)}, \ldots, x_n^{(k-1)}\right)$$

e

$$x^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right),$$

utilizando a variação relativa

$$Var^{(k)} = max\left\{v_1^{(k)}, v_2^{(k)}, \ldots, v_n^{(k)}\right\},$$

onde

$$v_i^{(k)} = \begin{cases} \left|\dfrac{x_i^{(k)} - x_i^{(k-1)}}{x_i^{(k)}}\right|, & \text{se} \quad x_i^{(k)} \neq 0, \\ 0, & \text{se} \quad x_i^{(k)} = x_i^{(k-1)} = 0, \\ 1, & \text{se} \quad x_i^{(k)} = 0 \quad \text{e} \quad x_i^{(k-1)} \neq 0. \end{cases}$$

O processo iterativo é interrompido quando $Var^{(k)} \leq \varepsilon$, onde ε é uma precisão prefixada. A solução do sistema será dada então pela k-ésima aproximação obtida. Entretanto, o processo pode não convergir, o que faz com que seja necessário estipular um número máximo de iterações (ITMAX) a serem realizadas (HUMES et al., 1984).

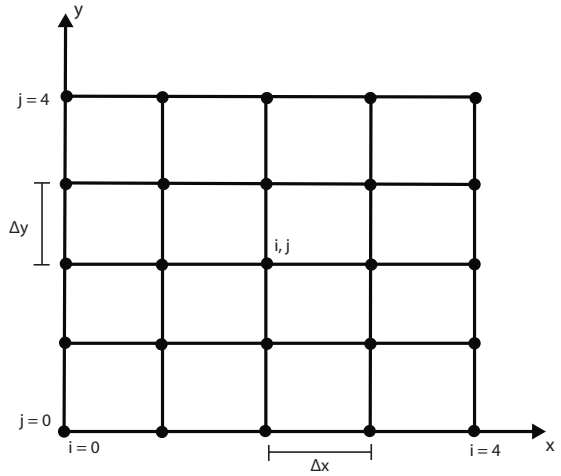
Pode-se mostrar que o método de Gauss-Seidel converge se a matriz dos coeficientes A no sistema linear $Ax = b$ é estritamente diagonal dominante, ou seja,

$$\mid a_{ii} \mid > \sum_{j=1,\, j\neq i}^{n} \mid a_{ij} \mid \ \forall i = 1, 2, \ldots, n. \tag{2.37}$$

A condição dada por (2.37) é suficiente à convergência do método de Gauss-Seidel, sendo denominada *critério das linhas* (BURDEN; FAIRES; BURDEN, 2016; HUMES et al., 1984). Há outros critérios de convergência para o método de Gauss-Seidel (GARCIA; HUMES; STERN, 2003; KALABA; SPINGARN, 1978).

Exemplo 2.2. *Utilizar o Método de Gauss-Seidel para solucionar o sistema linear obtido pela aplicação da equação* (2.31) *no domínio espacial bidimensional ilustrado na Figura 2.3.*

Figura 2.3 – Domínio espacial bidimensional para a equação discreta (2.31)



Fonte: Almeida (2017).

Isolando $\psi_{i,j}$ na equação (2.31)*, temos que:*

$$\psi_{i,j} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{i+1,j} + \psi_{i-1,j} + \epsilon^2\psi_{i,j+1} + \epsilon^2\psi_{i,j-1}\right). \tag{2.38}$$

A equação (2.38) *deve ser aplicada em cada ponto do domínio ilustrado na Figura 2.3. Nos nove pontos internos (i,j) $(1 \leqslant i,j \leqslant 3)$ desse domínio, não se conhece o valor de $\psi_{i,j}$. Já nos demais pontos, o valor de $\psi_{i,j}$ é definido pelas condições de contorno da EDP. As nove equações obtidas são descritas a seguir.*

- *Linha com $j = 1$:*

$$\psi_{1,1} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{2,1} + \psi_{0,1} + \epsilon^2\psi_{1,2} + \epsilon^2\psi_{1,0}\right); \tag{2.39}$$

$$\psi_{2,1} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{3,1} + \psi_{1,1} + \epsilon^2\psi_{2,2} + \epsilon^2\psi_{2,0}\right); \tag{2.40}$$

$$\psi_{3,1} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{4,1} + \psi_{2,1} + \epsilon^2\psi_{3,2} + \epsilon^2\psi_{3,0}\right). \tag{2.41}$$

- *Linha com $j = 2$:*

$$\psi_{1,2} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{2,2} + \psi_{0,2} + \epsilon^2\psi_{1,3} + \epsilon^2\psi_{1,1}\right); \tag{2.42}$$

$$\psi_{2,2} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{3,2} + \psi_{1,2} + \epsilon^2\psi_{2,3} + \epsilon^2\psi_{2,1}\right); \tag{2.43}$$

$$\psi_{3,2} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{4,2} + \psi_{2,2} + \epsilon^2\psi_{3,3} + \epsilon^2\psi_{3,1}\right). \tag{2.44}$$

- *Linha com $j = 3$:*

$$\psi_{1,3} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{2,3} + \psi_{0,3} + \epsilon^2\psi_{1,4} + \epsilon^2\psi_{1,2}\right); \tag{2.45}$$

$$\psi_{2,3} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{3,3} + \psi_{1,3} + \epsilon^2\psi_{2,4} + \epsilon^2\psi_{2,2}\right); \tag{2.46}$$

$$\psi_{3,3} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{4,3} + \psi_{2,3} + \epsilon^2\psi_{3,4} + \epsilon^2\psi_{3,2}\right). \tag{2.47}$$

Os valores de $\psi_{i,j}$, dados pelas equações (2.39)-(2.47)*, devem ser calculados em alguma ordem, isto é, devemos decidir se $\psi_{1,3}$ será calculado antes ou depois de $\psi_{3,1}$. Usualmente, calculamos os valores de $\psi_{i,j}$ de acordo com a sua posição no domínio computacional. Exemplificando, em duas dimensões, esses valores são calculados da esquerda para a direita e de baixo para cima. Assim, tal ordem resulta no cálculo de $\psi_{1,1}^{(k+1)}$, $\psi_{2,1}^{(k+1)}$, $\psi_{3,1}^{(k+1)}$, $\psi_{1,2}^{(k+1)}$, $\psi_{2,2}^{(k+1)}$, $\psi_{3,2}^{(k+1)}$ etc.*

Aplicando o método de Gauss-Seidel, temos que os valores calculados de $\psi_{i,j}$ pertencem à iteração $(k+1)$. Assim, usando a ordem de cálculo definida anteriormente, obtemos as relações recursivas para as equações (2.39)-(2.47)*, que são citadas a seguir.*

- *Linha com $j = 1$:*

$$\psi_{1,1}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{2,1}^{(k)} + \underline{\psi_{0,1}} + \epsilon^2\psi_{1,2}^{(k)} + \epsilon^2\underline{\psi_{1,0}}\right); \tag{2.48}$$

$$\psi_{2,1}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{3,1}^{(k)} + \psi_{1,1}^{(k+1)} + \epsilon^2\psi_{2,2}^{(k)} + \epsilon^2\underline{\psi_{2,0}}\right); \tag{2.49}$$

$$\psi_{3,1}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\underline{\psi_{4,1}} + \psi_{2,1}^{(k+1)} + \epsilon^2\psi_{3,2}^{(k)} + \epsilon^2\underline{\psi_{3,0}}\right). \tag{2.50}$$

- *Linha com $j = 2$:*

$$\psi_{1,2}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{2,2}^{(k)} + \underline{\psi_{0,2}} + \epsilon^2\psi_{1,3}^{(k)} + \epsilon^2\psi_{1,1}^{(k+1)}\right); \tag{2.51}$$

$$\psi_{2,2}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{3,2}^{(k)} + \psi_{1,2}^{(k+1)} + \epsilon^2\psi_{2,3}^{(k)} + \epsilon^2\psi_{2,1}^{(k+1)}\right); \tag{2.52}$$

$$\psi_{3,2}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\underline{\psi_{4,2}} + \psi_{2,2}^{(k+1)} + \epsilon^2\psi_{3,3}^{(k)} + \epsilon^2\psi_{3,1}^{(k+1)}\right). \tag{2.53}$$

- *Linha com $j = 3$:*

$$\psi_{1,3}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{2,3}^{(k)} + \underline{\psi_{0,3}} + \epsilon^2\underline{\psi_{1,4}} + \epsilon^2\psi_{1,2}^{(k+1)}\right); \tag{2.54}$$

$$\psi_{2,3}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{3,3}^{(k)} + \psi_{1,3}^{(k+1)} + \epsilon^2\underline{\psi_{2,4}} + \epsilon^2\psi_{2,2}^{(k+1)}\right); \tag{2.55}$$

$$\psi_{3,3}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\underline{\psi_{4,3}} + \psi_{2,3}^{(k+1)} + \epsilon^2\underline{\psi_{3,4}} + \epsilon^2\psi_{3,2}^{(k+1)}\right). \tag{2.56}$$

Os termos sublinhados nas equações (2.48)-(2.56) *indicam os valores dos elementos de fronteira, que são definidos pelas condições de contorno. Retirando esses elementos, podemos generalizar as equações (2.48)-(2.56) na seguinte forma:*

$$\psi_{i,j}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(\psi_{i+1,j}^{(k)} + \psi_{i-1,j}^{(k+1)} + \epsilon^2\psi_{i,j+1}^{(k)} + \epsilon^2\psi_{i,j-1}^{(k+1)}\right). \tag{2.57}$$

Essa versão do método de Gauss-Seidel é a versão por ponto (PGS), pois a equação (2.57) provê o valor de apenas uma incógnita. Como para cada ponto do domínio calculamos apenas um único valor de $\psi_{i,j}^{k+1}$, o custo computacional, a cada iteração, é baixo. Desta maneira, é suficiente calcular a equação (2.57) em cada ponto do domínio até que o critério de parada seja satisfeito (FORTUNA, 2000).

2.5.1 GAUSS-SEIDEL POR LINHA

O método de Gauss-Seidel por linha (LGS) calcula simultaneamente os valores da solução ψ^{k+1} nos pontos de uma linha $j = a, a \in \mathbb{N}$. Para tanto, reescrevemos a equação (2.31) como:

$$\psi_{i+1,j}^{(k+1)} - 2(1+\epsilon^2)\psi_{i,j}^{(k+1)} + \psi_{i-1,j}^{(k+1)} = -\epsilon^2\psi_{i,j+1}^{(k)} - \epsilon^2\psi_{i,j-1}^{(k)}. \tag{2.58}$$

Aplicando a equação (2.58) no domínio da Figura 2.3, nos pontos da linha $j = 1$, temos que:

$$\psi_{2,1}^{(k+1)} - 2(1+\epsilon^2)\psi_{1,1}^{(k+1)} + \underline{\psi_{0,1}} = -\epsilon^2\psi_{1,2}^{(k)} - \epsilon^2\underline{\psi_{1,0}}; \tag{2.59}$$

$$\psi_{3,1}^{(k+1)} - 2(1+\epsilon^2)\psi_{2,1}^{(k+1)} + \psi_{1,1}^{(k+1)} = -\epsilon^2\psi_{2,2}^{(k)} - \epsilon^2\underline{\psi_{2,0}}; \tag{2.60}$$

$$\underline{\psi_{4,1}} - 2(1+\epsilon^2)\psi_{3,1}^{(k+1)} + \psi_{2,1}^{(k+1)} = -\epsilon^2\psi_{3,2}^{(k)} - \epsilon^2\underline{\psi_{3,0}}. \tag{2.61}$$

Os termos sublinhados nas equações (2.59)-(2.61) são os elementos da fronteira da malha. Essas três equações constituem um sistema tridiagonal, que pode ser resolvido pelo algoritmo de Thomas (FORTUNA, 2000). A solução desse sistema são os valores de ψ na primeira linha da malha, isto é, ψ_{11}, ψ_{21} e ψ_{31}.

O método de Gauss-Seidel por linha calcula, ao mesmo tempo, todas as incógnitas em uma linha da malha. Por isso, e devido à necessidade de resolver um sistema tridiagonal para cada linha da malha, o custo computacional do LGS é maior que a do PGS, porém a taxa de convergência do LGS é maior (FORTUNA, 2000).

2.6 MÉTODO SOR

O método SOR é descrito nesta seção conforme Burden, Faires e Burden (2016) e Fortuna (2000).

Com o avanço das iterações do método de Gauss-Seidel, a diferença entre sucessivas aproximações $\psi^{(k+1)}$ e $\psi^{(k)}$ diminui se o método converge. Contudo, podem ser necessárias muitas iterações até que se alcance a solução do sistema de equações com a precisão prefixada.

Para diminuir a quantidade de iterações, ou seja, acelerar a convergência do método de Gauss-Seidel, podemos empregar uma extrapolação desse método: o método de sobrerrelaxações sucessivas.

Seja $\tilde{x} \in \mathbb{R}^n$ uma aproximação para a solução do sistema linear $Ax = b$. O vetor residual de $\tilde{x}$ com respeito ao sistema é $r = b - A\tilde{x}$.

No método de Gauss-Seidel, o vetor residual é associado a cada aproximação do vetor solução. O objetivo é gerar uma sequência de aproximações que faça o vetor residual convergir rapidamente para o vetor nulo.

Suponhamos que

$$r_i^{(k)} = \left(r_{1i}^{(k)}, r_{2i}^{(k)}, r_{3i}^{(k)}, \ldots, r_{ni}^{(k)}\right)$$

denote o vetor residual para o método de Gauss-Seidel, correspondendo à sequência de aproximações

$$x_i^{(k)} = \left\{x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \ldots, x_{i-1}^{(k)}, x_i^{(k-1)}, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)}\right\}.$$

O m-ésimo componente do vetor $r_i^{(k)}$ é

$$r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj} x_j^{(k)} - \sum_{j=i}^{n} a_{mj} x_j^{(k-1)},$$

ou, equivalentemente,

$$r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj} x_j^{(k)} - \sum_{j=i+1}^{n} a_{mj} x_j^{(k-1)} - a_{mi} x_i^{(k-1)}, \ \forall m = 1, 2, \cdots, n. \tag{2.62}$$

Já o i-ésimo componente do vetor $r_i^{(k)}$ é

$$r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} - a_{ii} x_i^{(k-1)},$$

o qual pode ser reescrito como

$$a_{ii} x_i^{(k-1)} + r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}. \tag{2.63}$$

No Método de Gauss-Seidel, $x_i^{(k)}$ é dado por

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}. \tag{2.64}$$

Dessa forma, empregando (2.64) podemos reescrever (2.63) como:

$$a_{ii} x_i^{(k-1)} + r_{ii}^{(k)} = a_{ii} x_i^{(k)}. \tag{2.65}$$

Assim, utilizar o método de Gauss-Seidel pode ser caracterizado como determinar $x_i^{(k)}$ que satisfaça

$$x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}}. \tag{2.66}$$

A equação (2.66) pode ser reescrita como:

$$x_i^{(k)} = x_i^{(k-1)} + \omega \frac{r_{ii}^{(k)}}{a_{ii}}, \tag{2.67}$$

com $\omega > 0$. Certas escolhas de ω reduzem a norma do vetor residual e fazem com que a convergência seja significantemente mais rápida.

Métodos que geram uma sequência de aproximações empregando a relação recursiva (2.67) são denominados *métodos de relaxação*. Quando $0 < \omega < 1$, os métodos são denominados *métodos de subrelaxação*, sendo mais úteis quando o sistema linear a ser resolvido é oriundo de EDPs não lineares (FORTUNA, 2000). Quando $\omega > 1$, os métodos são denominados *métodos*

de sobrerrelaxação ou *SOR* (Sucessive Over-Relaxation), sendo utilizados para acelerar a convergência de sistemas que são convergentes pelo método de Gauss-Seidel.

Finalmente, substituindo (2.62) em (2.67), com $m = i$, concluímos que:

$$x_i^{(k)} = x_i^{(k-1)} + \omega \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} - a_{ii} x_i^{(k-1)}}{a_{ii}};$$

$$x_i^{(k)} = (1-\omega)\, x_i^{(k-1)} + \omega \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}. \tag{2.68}$$

A equação (2.68) define a relação de recorrência do método SOR.

2.7 DISCRETIZAÇÃO DA EQUAÇÃO DE POISSON BIDIMENSIONAL

A equação de Poisson bidimensional é dada por:

$$\begin{aligned} \nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= f(x,y), \ \ (x,y) \in \Omega \subset \mathbb{R}^2 \text{ e } f(x,y) \neq 0, \\ u(x,y) &= g(x,y), \ \ \text{se } \ (x,y) \in \partial\Omega. \end{aligned} \tag{2.69}$$

A discretização de (2.69) com diferenças centradas de segunda ordem, definidas por (2.19), resulta em:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + \left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} \approx f_{i,j}, \tag{2.70}$$

onde $i, j = (x_i, y_j) \in \Omega \cup \partial\Omega$.

A expressão discreta (2.70) define um estêncil de cinco pontos para o cálculo de $u_{i,j}$. Esse estêncil é ilustrado na Figura 2.4(a).

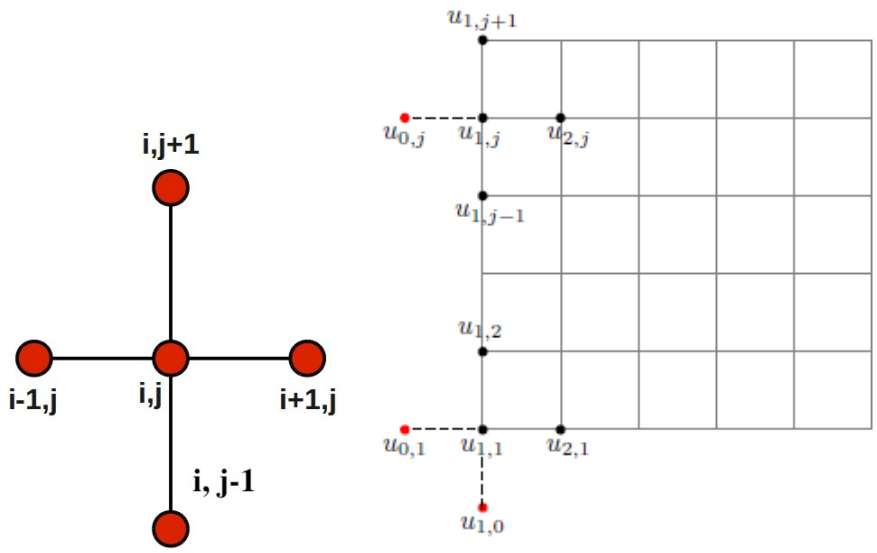
Quando empregamos condições de contorno de Dirichlet, a matriz A, no sistema linear $Au = f$ definido pela relação discreta (2.70), é simétrica $\Big(a_{ij} = a_{ji}$ ou $A^T = A\Big)$, diagonal dominante $\left(\mid a_{ii} \mid \geq \sum_{j=1,\, j\neq i}^{n} \mid a_{ij} \mid \ \forall i = 1, 2, \ldots, n \right)$ e definida positiva[3] (CHENG, 2020). Já quando utilizamos condições de contorno de Neumann, com a derivada primeira na fronteira discretizada com diferenças finitas centradas de segunda ordem dada por (2.16), a matriz A é simétrica e semidefinida positiva[4] (CHENG, 2020). Neste último caso, precisamos determinar valores para $u_{i,j}$ em pontos fantasmas, como ilustra a Figura 2.4(b).

A matriz A no sistema linear $Au = f$ é determinante à convergência do método de Gauss-Seidel.

[3] Uma matriz real A_{nxn} simétrica é definida positiva se $x^T Ax > 0$ para todo vetor x não nulo. Todos os autovalores de uma matriz simétrica definida positiva são positivos.

[4] Uma matriz real A_{nxn} simétrica é semidefinida positiva se $x^T Ax \geq 0$ para todo vetor x não nulo. Todos os autovalores de uma matriz simétrica semidefinida positiva são não negativos.

Figura 2.4 – Discretização centrada de segunda ordem da equação de Poisson bidimensional: (a) estêncil de cinco pontos; (b) pontos fantasmas na malha estruturada



(a) (b)

Fonte: (a) Williams (2011); (b) Chen (2020).

2.7.1 SOLUÇÃO DO SISTEMA LINEAR

Isolando $u_{i,j}$ na equação discreta (2.70) e considerando $\dfrac{\Delta x}{\Delta y} = \epsilon$, obtemos que:

$$u_{i,j} = \frac{1}{2(1+\epsilon^2)}\left(u_{i+1,j} + u_{i-1,j} + \epsilon^2 u_{i,j+1} + \epsilon^2 u_{i,j-1} - (\Delta x)^2 f_{i,j}\right). \tag{2.71}$$

A equação (2.71) deve ser aplicada a cada ponto do domínio discreto, gerando um sistema de equações lineares. Esse sistema pode ser solucionado por intermédio dos métodos iterativos de Gauss-Seidel e SOR, cujas equações iterativas são dadas, respectivamente, por:

$$u_{i,j}^{(k+1)} = \frac{1}{2(1+\epsilon^2)}\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k+1)} + \epsilon^2 u_{i,j+1}^{(k)} + \epsilon^2 u_{i,j-1}^{(k+1)} - (\Delta x)^2 f_{i,j}\right); \tag{2.72}$$

$$u_{i,j}^{(k+1)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{2(1+\epsilon^2)}\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k+1)} + \epsilon^2 u_{i,j+1}^{(k)} + \epsilon^2 u_{i,j-1}^{(k+1)} - (\Delta x)^2 f_{i,j}\right). \tag{2.73}$$

2.8 DISCRETIZAÇÃO DA EQUAÇÃO DE POISSON TRIDIMENSIONAL

A equação de Poisson tridimensional é dada por:

$$\nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f(x,y,z), (x,y,z) \in \Omega \subset \mathbb{R}^3 \text{ e } f(x,y,z) \neq 0, \tag{2.74}$$

$$u(x,y,z) = g(x,y,z) \text{ se } (x,y,z) \in \partial\Omega.$$

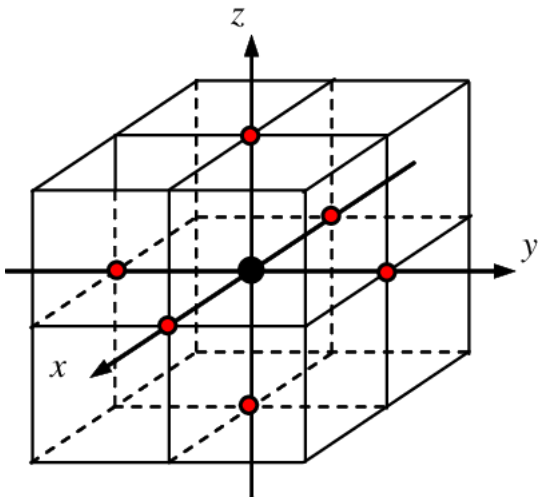
Aplicando à equação (2.74) diferenças centradas de segunda ordem, definidas por (2.19), temos que:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j,k} + \left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j,k} + \left.\frac{\partial^2 u}{\partial z^2}\right|_{i,j,k} \approx \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{(\Delta x)^2} + \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{(\Delta y)^2} + \\ + \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta z)^2} \approx f_{i,j,k}, \tag{2.75}$$

sendo $i, j, k = (x_i, y_j, z_k) \in \Omega \cup \partial\Omega$.

A expressão discreta (2.75) define um estêncil de sete pontos para o cálculo de $u_{i,j,k}$, ilustrado na Figura 2.5.

Figura 2.5 – Estêncil de sete pontos para a discretização centrada de segunda ordem da equação de Poisson tridimensional



Fonte: ResearchGate (2013).

2.8.1 SOLUÇÃO DO SISTEMA LINEAR

Isolando $u_{i,j,k}$ na equação discreta (2.75) e considerando $\dfrac{\Delta x \Delta y}{\Delta z} = \epsilon$, obtemos que:

$$u_{i,j,k} = \frac{1}{2((\Delta x)^2 + (\Delta y)^2 + \epsilon^2)} \left((\Delta y)^2 \left(u_{i+1,j,k} + u_{i-1,j,k}\right) + (\Delta x)^2 \left(u_{i,j+1,k} + u_{i,j-1,k}\right) \right) + \\ + \frac{1}{2((\Delta x)^2 + (\Delta y)^2 + \epsilon^2)} \left(\epsilon^2 u_{i,j,k+1} + \epsilon^2 u_{i,j,k-1} - (\Delta x)^2 (\Delta y)^2 f_{i,j,k} \right). \tag{2.76}$$

A equação (2.76) deve ser aplicada a cada ponto do domínio discreto, gerando um sistema de equações lineares. As equações iterativas para a solução desse sistema por intermédio dos métodos de Gauss-Seidel e SOR são dadas, respectivamente, por:

$$u_{i,j,k}^{(k+1)} = \frac{1}{2((\Delta x)^2 + (\Delta y)^2 + \epsilon^2)} \left((\Delta y)^2 \left(u_{i+1,j,k}^{(k)} + u_{i-1,j,k}^{(k+1)}\right) + (\Delta x)^2 \left(u_{i,j+1,k}^{(k)} + u_{i,j-1,k}^{(k+1)}\right) \right) + \\ + \frac{1}{2((\Delta x)^2 + (\Delta y)^2 + \epsilon^2)} \left(\epsilon^2 u_{i,j,k+1}^{(k)} + \epsilon^2 u_{i,j,k-1}^{(k+1)} - (\Delta x)^2 (\Delta y)^2 f_{i,j,k} \right); \tag{2.77}$$

$$
\begin{aligned}
u_{i,j,k}^{(k+1)} =& (1-\omega)u_{i,j,k}^{(k)} + \\
&+ \frac{\omega}{2((\Delta x)^2 + (\Delta y)^2 + \epsilon^2)} \left((\Delta y)^2 \left(u_{i+1,j,k}^{(k)} + u_{i-1,j,k}^{(k+1)} \right) + (\Delta x)^2 \left(u_{i,j+1,k}^{(k)} + u_{i,j-1,k}^{(k+1)} \right) \right) + \\
&+ \frac{\omega}{2((\Delta x)^2 + (\Delta y)^2 + \epsilon^2)} \left(\epsilon^2 u_{i,j,k+1}^{(k)} + \epsilon^2 u_{i,j,k-1}^{(k+1)} - (\Delta x)^2 (\Delta y)^2 f_{i,j,k} \right). \qquad (2.78)
\end{aligned}
$$

3 SIMULAÇÕES COMPUTACIONAIS: SOLUÇÕES MANUFATURADAS

Uma solução manufaturada para a equação de Poisson

$$\nabla u = f, \tag{3.1}$$

é uma solução construída por intermédio da determinação da função f a partir da atribuíção de uma função para u. A partir desta atribuição, deriva-se u em relação às variáveis independentes.

Exemplo 3.1. *Seja* $u(x,y) = x^3y^3$. *Derivando* $u(x,y)$ *duas vezes em relação às variáveis independentes* x *e* y, *temos que:*

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = 6xy\left(x^2+y^2\right). \tag{3.2}$$

Assim, $u(x,y) = x^3y^3$ *é uma solução manufaturada para a equação de Poisson* (3.2).

Na construção de uma solução manufaturada para a equação de Poisson (3.1), devemos considerar, além da função u, as condições de contorno.

Neste capítulo, construímos algumas soluções manufaturadas para as equações de Poisson bidimensional e tridimensional. Objetivamos com essas soluções validar os códigos computacionais, elaborados em linguagem C (Apêndice B), em malhas *isotrópicas* ($\Delta x = \Delta y = \Delta z$) e *anisotrópicas* ($\Delta x \neq \Delta y \neq \Delta z$), assim como testar valores ótimos para o parâmetro w no método SOR.

Nos testes que realizamos com as soluções manufaturadas, avaliamos os seguintes parâmetros:

1. hx, hy, hz ($\Delta x, \Delta y, \Delta z$): passo espacial nas direções, respectivamente, x, y e z;
2. *Número de iterações*: número de vezes que o computador precisa executar o código para satisfazer o critério de parada (tolerância, precisão prefixada) dos métodos iterativos de Gauss-Seidel e SOR;
3. *Erro absoluto*: diferença entre as soluções exata (manufaturada) e numérica, na norma do máximo ($\|$erro absoluto$\|_\infty$) (Apêndice A);
4. *Tempo de CPU*: quantos segundos são necessários para que o computador execute o código;
5. w: parâmetro de aceleração de convergêcia ótimo do método SOR.

Na solução numérica das soluções manufaturadas propostas, empregamos para os métodos de Gauss-Seidel e SOR uma tolerância (precisão prefixada) de 10^{-6}.

Os testes computacionais foram efetuados em um computador com as seguintes configurações: processador AMD Ryzen 7 3700X; placa de vídeo AMD Radeon RX 5700 XT; 2x8 Gb de memória RAM 3200 MHz; fonte de 600 W com PFC ativo e certificado 80 plus bronze.

Os códigos computacionais foram compilados/executados no *Dev-C++* (LAPLACE, 2020), e as soluções exata e numérica bidimensionais e tridimensionais foram visualizadas, respectivamente, com o *Matlab* (MATLAB, 2021) e com o Tecplot 360 (TECPLOT, 2021).

3.1 BIDIMENSIONAIS

Elaboramos seis soluções manufaturadas para a equação de Poisson bidimensional. Nas cinco primeiras (problemas um a cinco), testamos condições de contorno de Dirichlet; na sexta (problema seis), condições de contorno de Neumann.

Nos testes com condições de contorno de Dirichlet, aplicados o operador Laplaciano discreto (2.70) nos pontos interiores da malha, mantendo a solução exata nos pontos de fronteira. Já no teste com condições de contorno de Neumann, aplicamos o operador Laplaciano discreto (2.70) em todos os pontos da malha. Neste caso, precisamos calcular valores para pontos fantasmas. Para atribuir valores para estes pontos, empregamos a discretização centrada de segunda ordem para a derivada primeira, definida em (2.16). Isolando nesta expressão o ponto fantasma, obtemos, para cada uma das fronteiras:

$$\begin{aligned}
\text{Fronteira inferior :} \quad & u_{i,0} = u_{i,2} - 2\Delta y \frac{\partial u}{\partial y}; \\
\text{Fronteira superior :} \quad & u_{i,n+1} = u_{i,n-1} + 2\Delta y \frac{\partial u}{\partial y}; \\
\text{Fronteira esquerda :} \quad & u_{0,j} = u_{2,j} - 2\Delta x \frac{\partial u}{\partial x}; \\
\text{Fronteira direita :} \quad & u_{m+1,j} = u_{m-1,j} + 2\Delta x \frac{\partial u}{\partial x},
\end{aligned}$$

onde $m, n \in \mathbb{N}$ representam o número de pontos da malha nas direções x e y, respectivamente.

Para problemas com todas as fronteiras de Neumann, há uma condição de compatibilidade ou de integrabilidade (NÓS; ROMA; CENICEROS, 2005; STRIKWERDA, 1989) para que os problemas sejam bem postos. Para um problema bidimensional, essa condição é dada por:

$$\iint_{\Omega} f = \int_{\partial\Omega} g, \tag{3.3}$$

sendo $g = \dfrac{\partial u}{\partial \eta}$ a condição de contorno de Neumann, e η a normal à fronteira.

1. **Primeiro problema**

$$\text{Solução exata} : u(x,y) = xy.$$
$$\text{Equação de Laplace} : \frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = 0.$$

Domínio: $[0,1]\times[0,1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x,0) = 0;$$
$$\text{Fronteira superior} : u(x,1) = x;$$
$$\text{Fronteira esquerda} : u(0,y) = 0;$$
$$\text{Fronteira direita} : u(1,y) = y.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

Tabela 3.1 – Solução numérica do primeiro problema 2D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = 0.1$	
Número de iterações	136
$\|$erro absoluto$\|_\infty$	$6.20246e-007$
Tempo de CPU (s)	$0.5459\ s$
Passo espacial: $hx = hy = 0.01$	
Número de iterações	8469
$\|$erro absoluto$\|_\infty$	$9.76499e-005$
Tempo de CPU (s)	$7.186\ s$
Passo espacial: $hx = hy = 0.002$	
Número de iterações	129193
$\|$erro absoluto$\|_\infty$	$2.48451e-003$
Tempo de CPU (s)	$4561\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$	
Número de iterações	1383
$\|$erro absoluto$\|_\infty$	$1.21496e-005$
Tempo de CPU (s)	$0.5969\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$	
Número de iterações	18779
$\|$erro absoluto$\|_\infty$	$2.48601e-004$
Tempo de CPU (s)	$30.91\ s$

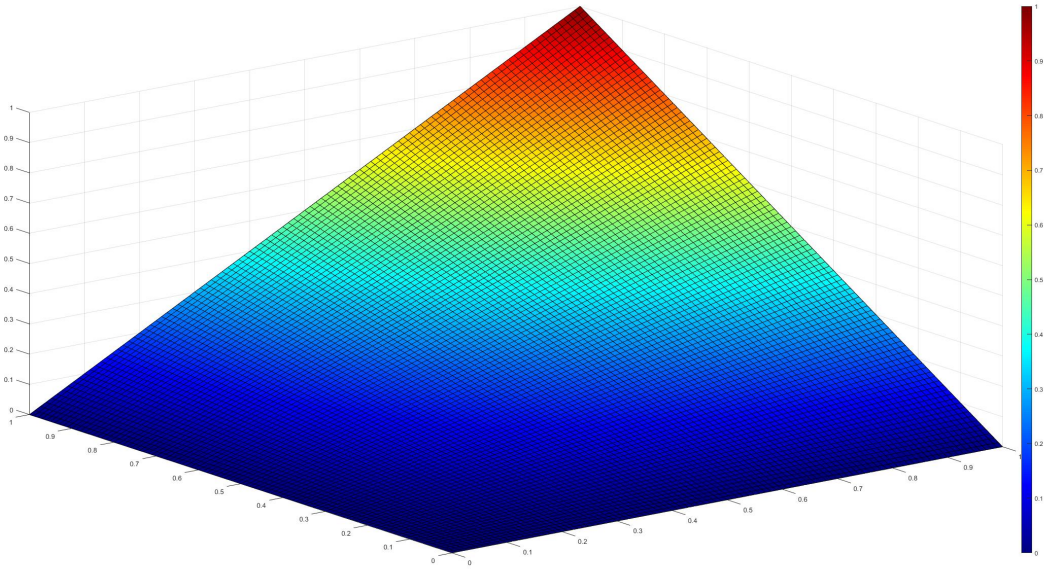
Fonte: O autor.

Tabela 3.2 – Solução numérica do primeiro problema 2D pelo método SOR

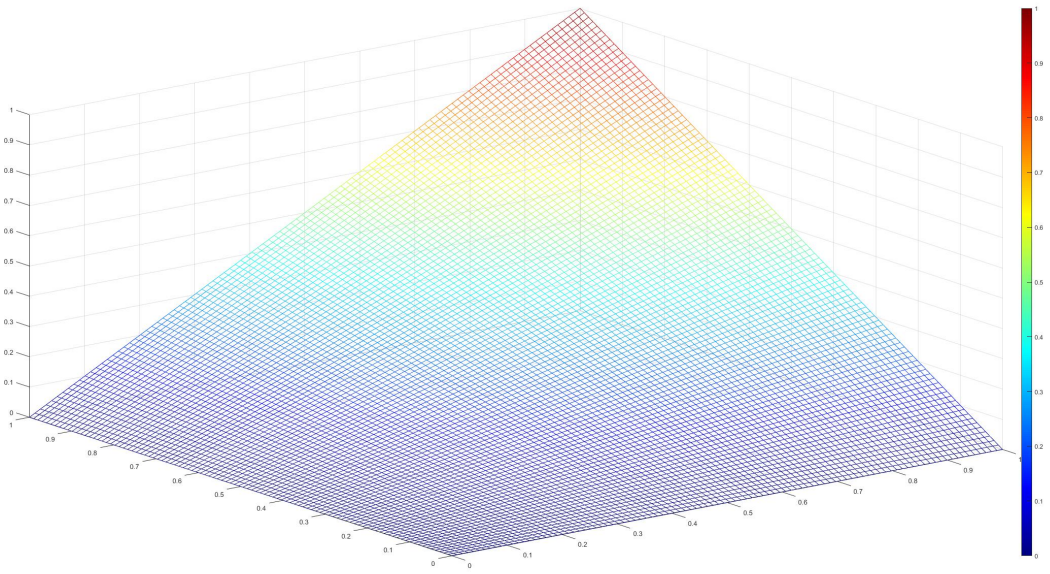
Passo espacial: $hx = hy = 0.1$; w=1.6	
Número de iterações	42
$\|$erro absoluto$\|_\infty$	$3.71856e-009$
Tempo de CPU (s)	$2.14\ s$
Passo espacial: $hx = hy = 0.01$; w=1.9	
Número de iterações	630
$\|$erro absoluto$\|_\infty$	$1.92781e-006$
Tempo de CPU (s)	$3.372\ s$
Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	11046
$\|$erro absoluto$\|_\infty$	$1.1178e-004$
Tempo de CPU (s)	$411.4\ s$
Passo espacial: $hx = hy = 0.001$; w=1.9	
Número de iterações	36106
$\|$erro absoluto$\|_\infty$	$4.89452e-004$
Tempo de CPU (s)	$1.002e+004\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$; w=1.9	
Número de iterações	179
$\|$erro absoluto$\|_\infty$	$1.51961e-008$
Tempo de CPU (s)	$2.587\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$; w=1.9	
Número de iterações	1454
$\|$erro absoluto$\|_\infty$	$7.5251e-006$
Tempo de CPU (s)	$7.978\ s$
Passo espacial: $hx = 0.002 \quad hy = 0.00\overline{3}$; w=1.9	
Número de iterações	7898
$\|$erro absoluto$\|_\infty$	$7.35057e-005$
Tempo de CPU (s)	$172.2\ s$

Fonte: O autor.

Figura 3.1 – Solução manufaturada do primeiro problema 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-2}$, $w = 1.9$ e 630 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

2. **Segundo problema**

$$\text{Solução exata} : u(x,y) = x^2 + y^2.$$
$$\text{Equação de Poisson} : \frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = 4.$$

Domínio: $[0,1]\times[0,1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x,0) = x^2;$$
$$\text{Fronteira superior} : u(x,1) = x^2 + 1;$$
$$\text{Fronteira esquerda} : u(0,y) = y^2;$$
$$\text{Fronteira direita} : u(1,y) = y^2 + 1.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

Tabela 3.3 – Solução numérica do segundo problema 2D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = 0.1$	
Número de iterações	137
$\|$erro absoluto$\|_\infty$	$1.28327e-006$
Tempo de CPU (s)	$2.801\ s$
Passo espacial: $hx = hy = 0.01$	
Número de iterações	8640
$\|$erro absoluto$\|_\infty$	$1.95323e-004$
Tempo de CPU (s)	$15.15\ s$
Passo espacial: $hx = hy = 0.002$	
Número de iterações	133585
$\|$erro absoluto$\|_\infty$	$4.96482e-003$
Tempo de CPU (s)	$1.003e+004\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$	
Número de iterações	1400
$\|$erro absoluto$\|_\infty$	$2.51059e-005$
Tempo de CPU (s)	$2.397\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$	
Número de iterações	19211
$\|$erro absoluto$\|_\infty$	$4.9728e-004$
Tempo de CPU (s)	$56.22\ s$

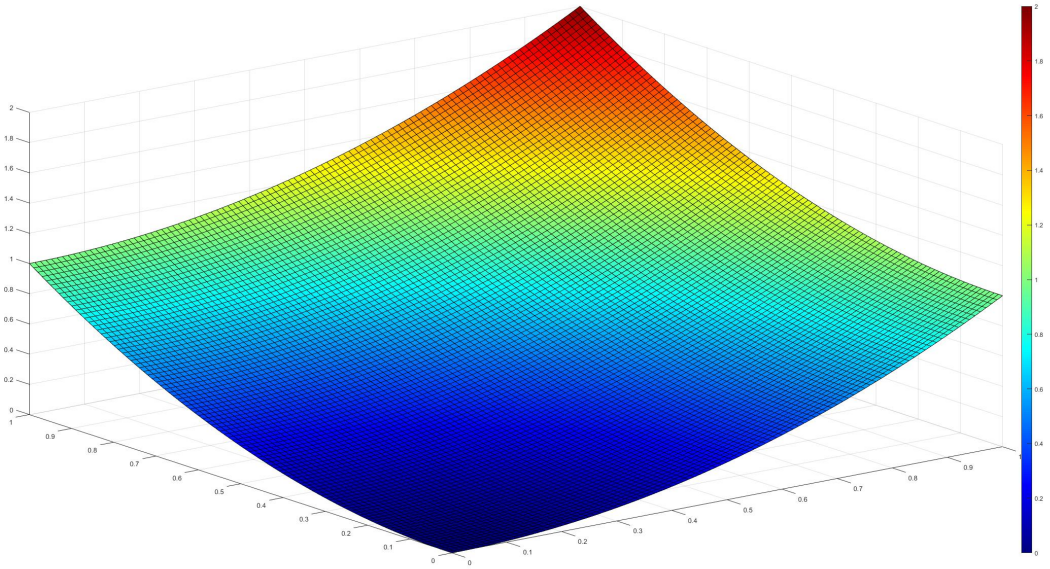
Fonte: O autor.

Tabela 3.4 – Solução numérica do segundo problema 2D pelo método SOR

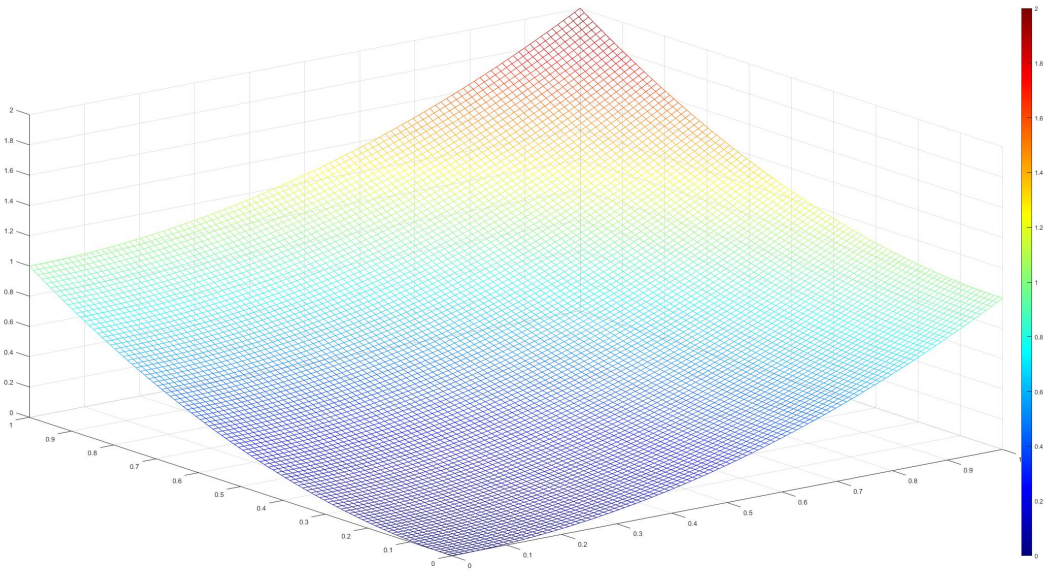
Passo espacial: $hx = hy = 0.1$; w=1.6	
Número de iterações	42
$\|$erro absoluto$\|_\infty$	$6.64642e-009$
Tempo de CPU (s)	$3.486\ s$
Passo espacial: $hx = hy = 0.01$; w=1.9	
Número de iterações	635
$\|$erro absoluto$\|_\infty$	$3.87203e-006$
Tempo de CPU (s)	$3.013\ s$
Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	11255
$\|$erro absoluto$\|_\infty$	$2.23503e-004$
Tempo de CPU (s)	$636.4\ s$
Passo espacial: $hx = hy = 0.001$; w=1.9	
Número de iterações	36986
$\|$erro absoluto$\|_\infty$	$9.78695e-004$
Tempo de CPU (s)	$1.171e+004\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$; w=1.9	
Número de iterações	173
$\|$erro absoluto$\|_\infty$	$5.61945e-008$
Tempo de CPU (s)	$3.162\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$; w=1.9	
Número de iterações	1470
$\|$erro absoluto$\|_\infty$	$1.51714e-005$
Tempo de CPU (s)	$9.82\ s$
Passo espacial: $hx = 0.002 \quad hy = 0.00\overline{3}$; w=1.9	
Número de iterações	8034
$\|$erro absoluto$\|_\infty$	$1.47579e-004$
Tempo de CPU (s)	$176.5\ s$

Fonte: O autor.

Figura 3.2 – Solução manufaturada do segundo problema 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-2}$, $w = 1.9$ e 635 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

3. **Terceiro problema**

$$\text{Solução exata} : u(x,y) = x^3y^2 + x^2y^3.$$
$$\text{Equação de Poisson} : \frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = 2x^3 + 2y^3 + 6x^2y + 6xy^2.$$

Domínio: $[0,1]\times[0,1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x,0) = 0;$$
$$\text{Fronteira superior} : u(x,1) = x^3 + x^2;$$
$$\text{Fronteira esquerda} : u(0,y) = 0;$$
$$\text{Fronteira direita} : u(1,y) = y^3 + y^2.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

Tabela 3.5 – Solução numérica do terceiro problema 2D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = 0.1$	
Número de iterações	192
$\|$erro absoluto$\|_\infty$	$1.25632e-009$
Tempo de CPU (s)	$8.552\ s$
Passo espacial: $hx = hy = 0.001$	
Número de iterações	21017
$\|$erro absoluto$\|_\infty$	$1.95313e-010$
Tempo de CPU (s)	67.39 s
Passo espacial: $hx = hy = 0.002$	
Número de iterações	564766
$\|$erro absoluto$\|_\infty$	$3.96753e-011$
Tempo de CPU (s)	$4.61e+004\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$	
Número de iterações	2389
$\|$erro absoluto$\|_\infty$	$2.91985e-009$
Tempo de CPU (s)	$2.021\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$	
Número de iterações	52611
$\|$erro absoluto$\|_\infty$	$1.86393e-010$
Tempo de CPU (s)	$1352\ s$

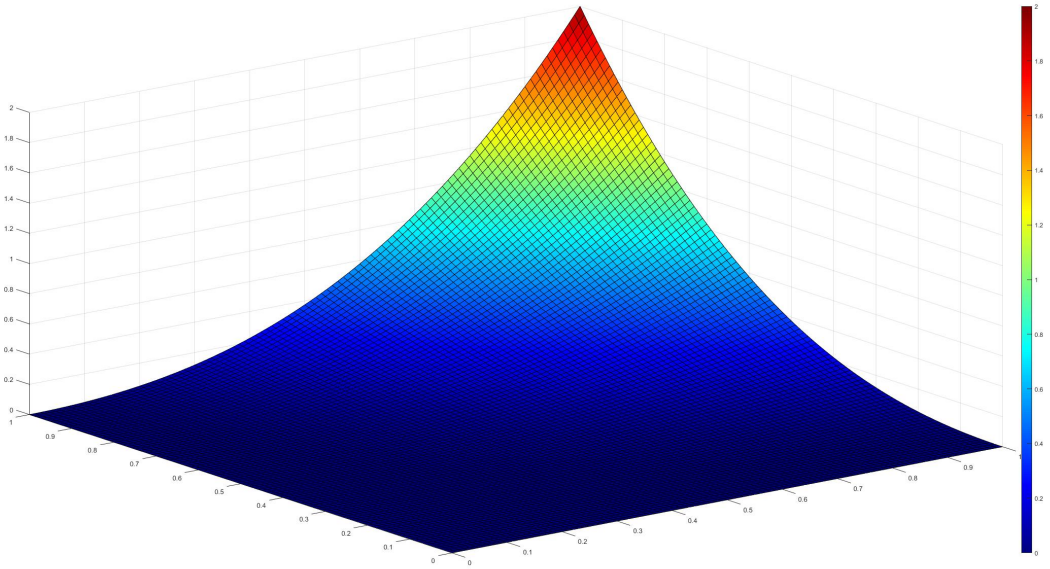
Fonte: O autor.

Tabela 3.6 – Solução numérica do terceiro problema 2D pelo método SOR

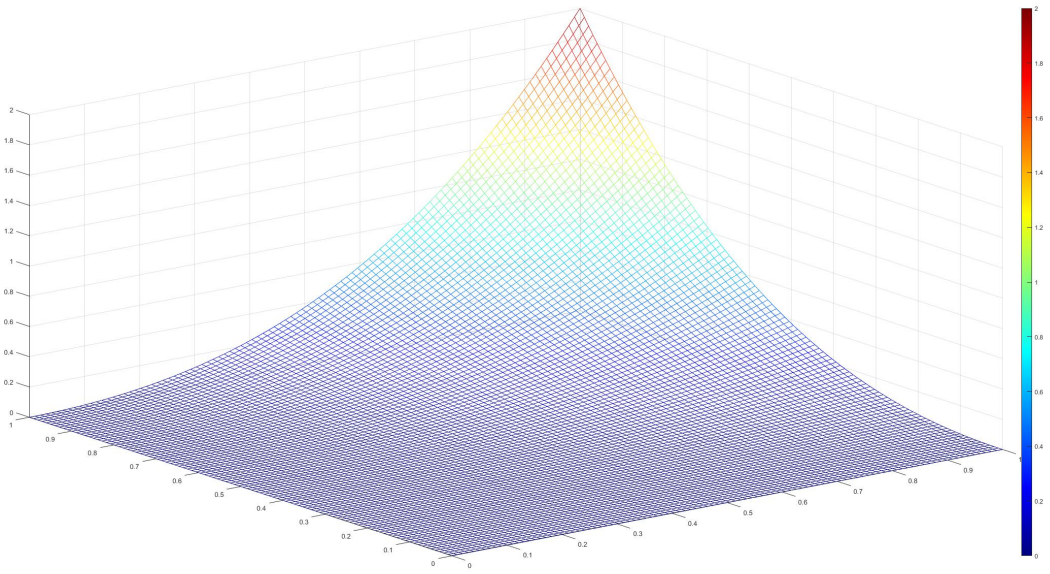
Passo espacial: $hx = hy = 0.1$; w=1.6	
Número de iterações	54
$\|$erro absoluto$\|_\infty$	$1.17616e-011$
Tempo de CPU (s)	$1.375\ s$
Passo espacial: $hx = hy = 0.01$; w=1.9	
Número de iterações	1244
$\|$erro absoluto$\|_\infty$	$3.82409e-012$
Tempo de CPU (s)	$5.638\ s$
Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	34006
$\|$erro absoluto$\|_\infty$	$1.81769e-012$
Tempo de CPU (s)	$2693\ s$
Passo espacial: $hx = hy = 0.001$; w=1.9	
Número de iterações	138985
$\|$erro absoluto$\|_\infty$	$1.06273e-012$
Tempo de CPU (s)	$4.747e+004\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$; w=1.9	
Número de iterações	247
$\|$erro absoluto$\|_\infty$	$1.753e-011$
Tempo de CPU (s)	$2.215\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$; w=1.9	
Número de iterações	3200
$\|$erro absoluto$\|_\infty$	$5.65657e-012$
Tempo de CPU (s)	$106.8\ s$
Passo espacial: $hx = 0.002 \quad hy = 0.00\overline{3}$; w=1.9	
Número de iterações	22781
$\|$erro absoluto$\|_\infty$	$2.9074e-012$
Tempo de CPU (s)	$1238\ s$

Fonte: O autor.

Figura 3.3 – Solução manufaturada do terceiro problema 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-2}$, $w = 1.9$ e 630 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

4. **Quarto problema**

$$\text{Solução exata} : u(x,y) = e^{sen(x)+cos(y)}.$$

Equação de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = \left(sen^2(y) + cos^2(x) - sen(x) - cos(y)\right) e^{sen(x)+cos(y)}.$$

Domínio: $[0,10]\times[0,6]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x,0) = e^{sen(x)+1};$$
$$\text{Fronteira superior} : u(x,6) = e^{sen(x)+cos(6)};$$
$$\text{Fronteira esquerda} : u(0,y) = e^{cos(y)};$$
$$\text{Fronteira direita} : u(10,y) = e^{sen(10)+cos(y)}.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

Tabela 3.7 – Solução numérica do quarto problema 2D pelo método de Gauss-Seidel

Passo espacial: $hx = 1 \quad hy = 0.6$	
Número de iterações	151
$\|$erro absoluto$\|_\infty$	$2.21717e-001$
Tempo de CPU (s)	$9.024\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.06$	
Número de iterações	9576
$\|$erro absoluto$\|_\infty$	$2.30752e-003$
Tempo de CPU (s)	$46.51\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.012$	
Número de iterações	158419
$\|$erro absoluto$\|_\infty$	$3.40134e-003$
Tempo de CPU (s)	$4.12e+004$
Passo espacial: $hx = 0.2 \quad hy = 0.6$	
Número de iterações	970
$\|$erro absoluto$\|_\infty$	$1.57767e-001$
Tempo de CPU (s)	$2.803\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.03$	
Número de iterações	26913
$\|$erro absoluto$\|_\infty$	$2.66628e-003$
Tempo de CPU (s)	$507.2\ s$

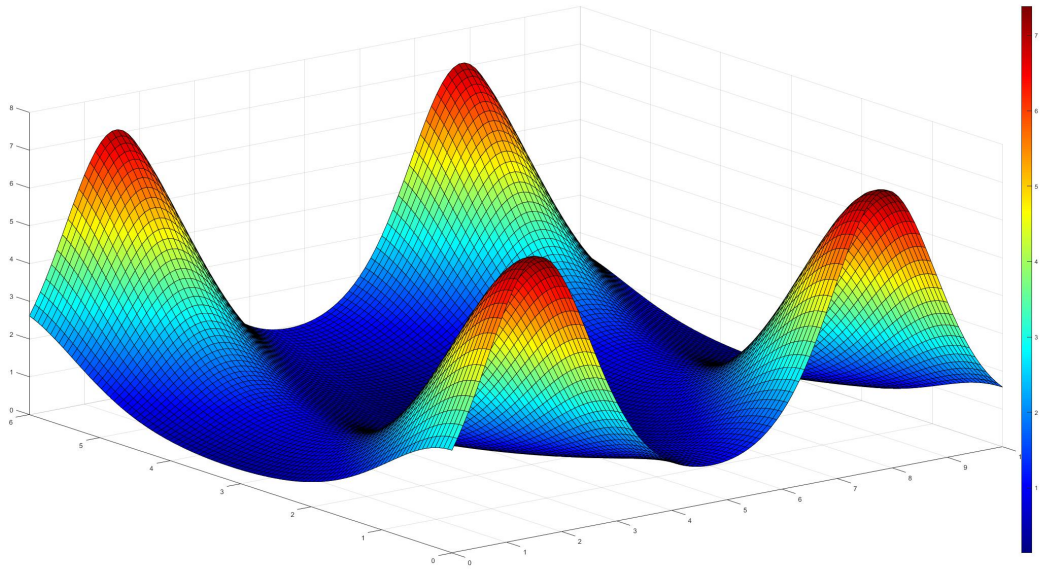
Fonte: O autor.

Tabela 3.8 – Solução numérica do quarto problema 2D pelo método SOR

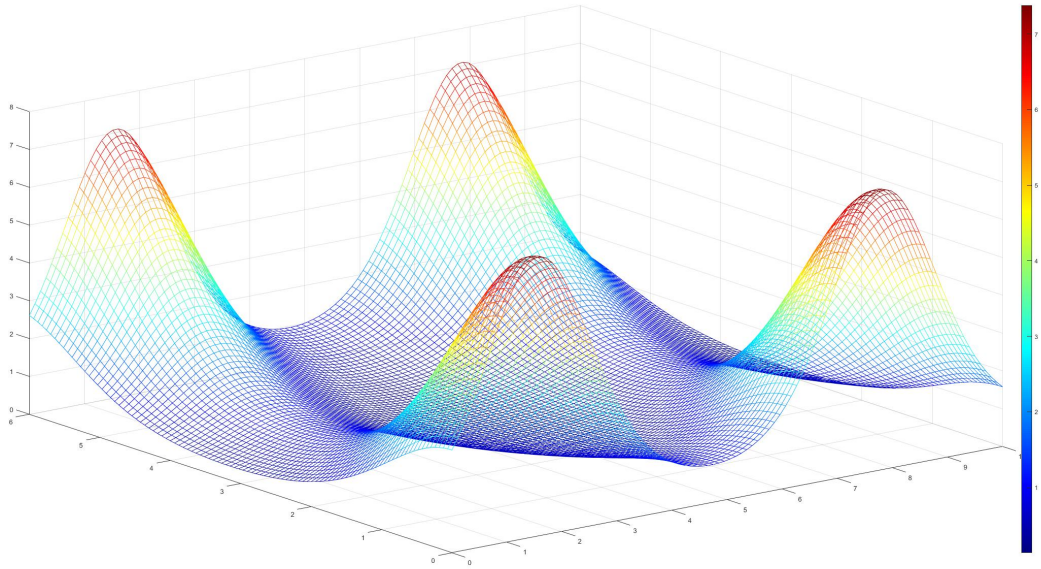
Passo espacial: $hx = 1 \quad hy = 0.6$; w=1.6	
Número de iterações	38
$\|$erro absoluto$\|_\infty$	$2.21717e-001$
Tempo de CPU (s)	$1.861\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.06$; w=1.9	
Número de iterações	621
$\|$erro absoluto$\|_\infty$	$2.32476e-003$
Tempo de CPU (s)	$2.96\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.012$; w=1.9	
Número de iterações	12194
$\|$erro absoluto$\|_\infty$	$2.19903e-004$
Tempo de CPU (s)	$1466\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.006$; w=1.9	
Número de iterações	41504
$\|$erro absoluto$\|_\infty$	$7.31003e-004$
Tempo de CPU (s)	$1.455e+004\ s$
Passo espacial: $hx = 0.2 \quad hy = 0.6$; w=1.9	
Número de iterações	166
$\|$erro absoluto$\|_\infty$	$1.57766e-001$
Tempo de CPU (s)	$2.423\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.03$; w=1.9	
Número de iterações	1884
$\|$erro absoluto$\|_\infty$	$2.74246e-003$
Tempo de CPU (s)	$20.09\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.02$; w=1.9	
Número de iterações	6895
$\|$erro absoluto$\|_\infty$	$2.29626e-004$
Tempo de CPU (s)	$1402\ s$

Fonte: O autor.

Figura 3.4 – Solução manufaturada do quarto problema 2D no domínio $[0, 10]\times[0, 6]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = 10^{-1}$, $hy = 6\text{x}10^{-2}$, $w = 1.9$ e 621 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

5. **Quinto problema**

$$\text{Solução exata : } u(x,y) = \ ln(xy+2)cos(xy).$$

Equação de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = -\ (x^2+y^2)cos(xy)ln(xy+2) - \frac{2(x^2+y^2)sen(xy)}{xy+2} + \\ - \frac{(x^2+y^2)cos(xy)}{(xy+2)^2}.$$

Domínio: $[0,3]\times[0,3]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior : } u(x,0) = ln(2);$$
$$\text{Fronteira superior : } u(x,3) = ln(3x+2)cos(3x);$$
$$\text{Fronteira esquerda : } u(0,y) = ln(2);$$
$$\text{Fronteira direita : } u(3,y) = ln(3y+2)cos(3y).$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

Tabela 3.9 – Solução numérica do quinto problema 2D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = 0.3$	
Número de iterações	136
$\|$erro absoluto$\|_\infty$	$6.8135e-002$
Tempo de CPU (s)	$2.372\ s$
Passo espacial: $hx = hy = 0.03$	
Número de iterações	14361
$\|$erro absoluto$\|_\infty$	$7.86107e-004$
Tempo de CPU (s)	$98.97\ s$
Passo espacial: $hx = hy = 0.006$	
Número de iterações	100001
$\|$erro absoluto$\|_\infty$	$797233e-003$
Tempo de CPU (s)	$9635\ s$
Passo espacial: $hx = 0.06 \quad hy = 0.3$	
Número de iterações	1833
$\|$erro absoluto$\|_\infty$	$5.39952e-002$
Tempo de CPU (s)	$5.959\ s$
Passo espacial: $hx = 0.03 \quad hy = 0.015$	
Número de iterações	35736
$\|$erro absoluto$\|_\infty$	$5.70269e-004$
Tempo de CPU (s)	$688.3\ s$

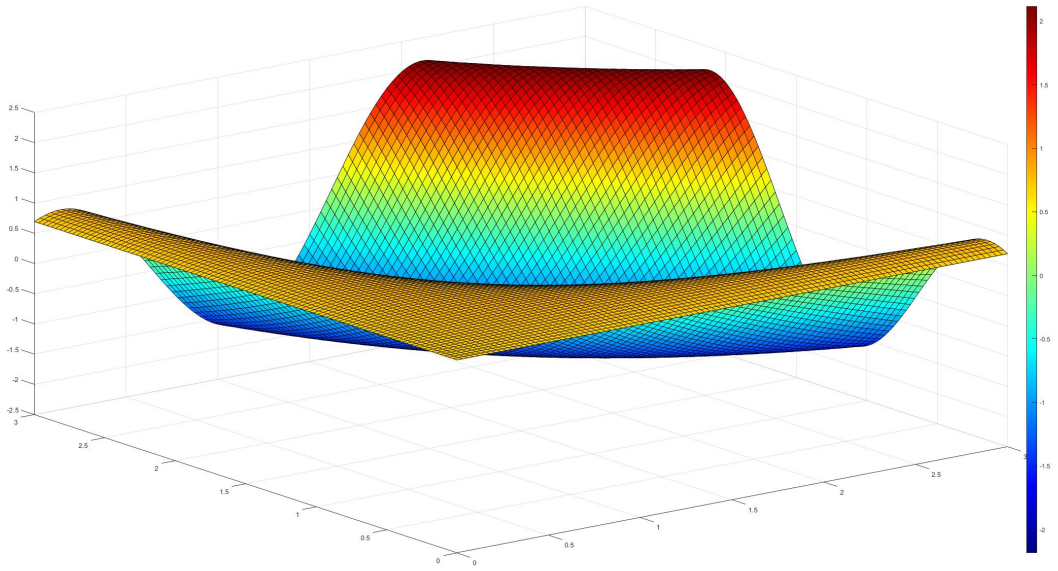
Fonte: O autor.

Tabela 3.10 – Solução numérica do quinto problema 2D pelo método SOR

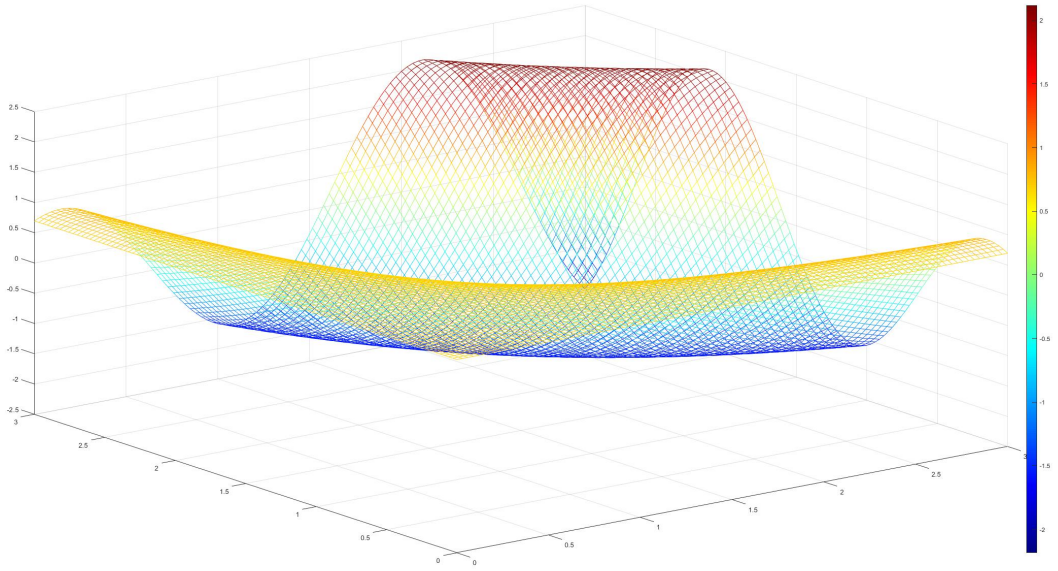
Passo espacial: $hx = hy = 0.3$; w=1.6	
Número de iterações	37
$\|$erro absoluto$\|_\infty$	$6.81354e-002$
Tempo de CPU (s)	$1.188\ s$
Passo espacial: $hx = hy = 0.03$; w=1.9	
Número de iterações	838
$\|$erro absoluto$\|_\infty$	$7.86052e-004$
Tempo de CPU (s)	$3.11\ s$
Passo espacial: $hx = hy = 0.006$; w=1.9	
Número de iterações	21912
$\|$erro absoluto$\|_\infty$	$3.14946e-005$
Tempo de CPU (s)	$5973\ s$
Passo espacial: $hx = hy = 0.003$; w=1.9	
Número de iterações	79519
$\|$erro absoluto$\|_\infty$	$7.90026e-006$
Tempo de CPU (s)	$2.621e+004\ s$
Passo espacial: $hx = 0.06 \quad hy = 0.3$; w=1.9	
Número de iterações	183
$\|$erro absoluto$\|_\infty$	$5.39954e-002$
Tempo de CPU (s)	$1.566\ s$
Passo espacial: $hx = 0.03 \quad hy = 0.015$; w=1.9	
Número de iterações	2227
$\|$erro absoluto$\|_\infty$	$5.70428e-004$
Tempo de CPU (s)	$65.92\ s$
Passo espacial: $hx = 0.006 \quad hy = 0.01$; w=1.9	
Número de iterações	16958
$\|$erro absoluto$\|_\infty$	$6.49659e-005$
Tempo de CPU (s)	$5785\ s$

Fonte: O autor.

Figura 3.5 – Solução manufaturada do quinto problema 2D no domínio $[0,3]\times[0,3]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 3\text{x}10^{-2}$, $w = 1.9$ e 838 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

6. **Sexto problema**

$$\text{Solução exata : } u(x,y) = cos(2\pi x)cos(2\pi y).$$

Equação de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = -8\pi^2 cos(2\pi x)cos(2\pi y).$$

Domínio: $[0,1]\times[0,1]$.

Condições de contorno de Neumann:

$$\text{Fronteira inferior : } -\frac{\partial}{\partial y}u(x,0) = 2\pi cos(2\pi x)sen\,(2\pi(0)) = 0;$$

$$\text{Fronteira superior : } \frac{\partial}{\partial y}u(x,1) = -2\pi cos(2\pi x)sen\,(2\pi(1)) = 0;$$

$$\text{Fronteira esquerda : } -\frac{\partial}{\partial x}u(0,y) = 2\pi sen\,(2\pi(0))\,cos(2\pi y) = 0;$$

$$\text{Fronteira direita : } \frac{\partial}{\partial x}u(1,y) = -2\pi sen\,(2\pi(1))\,cos(2\pi y) = 0.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

Tabela 3.11 – Solução numérica do sexto problema 2D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = 0.1$	
Número de iterações	179
Variação relativa máxima no GS	$9.80862e-007$
$\|$erro absoluto$\|_\infty$	$3.35611e-002$
Tempo de CPU (s)	$0.6505\ s$
Passo espacial: $hx = hy = 0.01$	
Número de iterações	43196
Variação relativa máxima no GS	$5.60654e-007$
$\|$erro absoluto$\|_\infty$	$3.29052e-004$
Tempo de CPU (s)	$70.82\ s$
Passo espacial: $hx = hy = 0.002$	
Número de iterações	100001
Variação relativa máxima no GS	$1.54821e-001$
$\|$erro absoluto$\|_\infty$	$1.38011e-005$
Tempo de CPU (s)	$3855\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$	
Número de iterações	1004
Variação relativa máxima no GS	$9.99565e-007$
$\|$erro absoluto$\|_\infty$	$1.7186e-002$
Tempo de CPU (s)	$0.7623\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.02$	
Número de iterações	1004
Variação relativa máxima no GS	$9.99565e-007$
$\|$erro absoluto$\|_\infty$	$1.7186e-002$
Tempo de CPU (s)	$0.5976\ s$

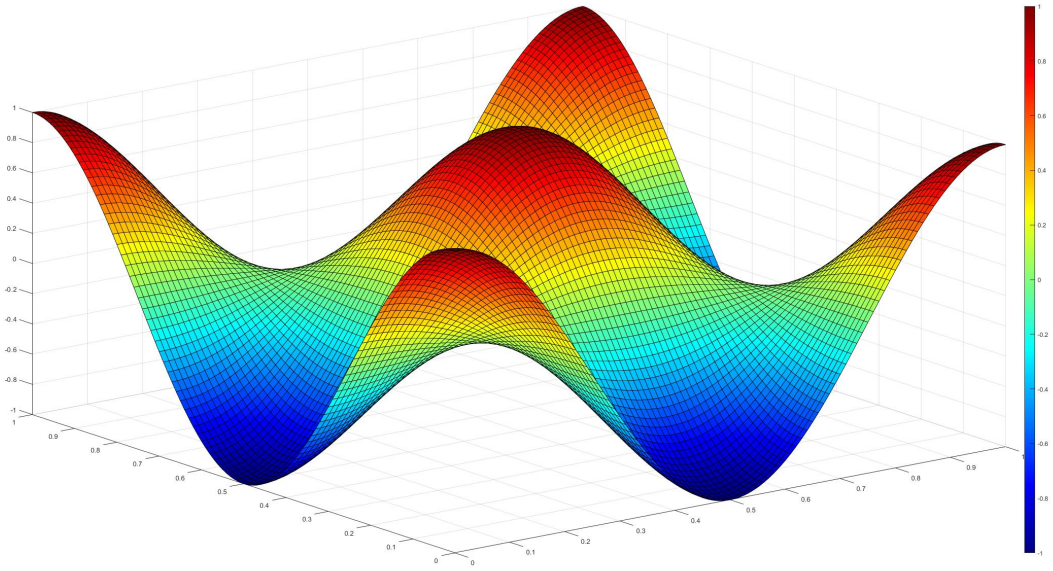
Fonte: O autor.

Tabela 3.12 – Solução numérica do sexto problema 2D pelo método SOR

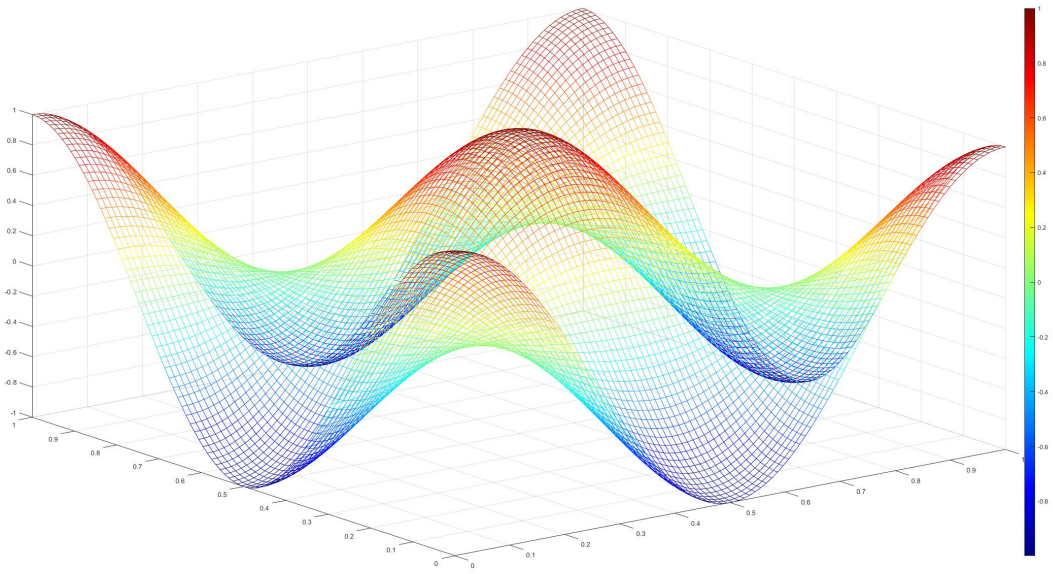
Passo espacial: $hx = hy = 0.1$; w=1.6	
Número de iterações	80
Variação relativa máxima no SOR	$8.98496e-007$
$\|$erro absoluto$\|_\infty$	$3.35592e-002$
Tempo de CPU (s)	$0.4262\ s$
Passo espacial: $hx = hy = 0.01$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	$8.98902e-001$
$\|$erro absoluto$\|_\infty$	$3.29052e-004$
Tempo de CPU (s)	$160.6\ s$
Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	7.0558
$\|$erro absoluto$\|_\infty$	$1.31596e-005$
Tempo de CPU (s)	$3955\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1$; w=1.9	
Número de iterações	263
Variação relativa máxima no SOR	$9.87492e-007$
$\|$erro absoluto$\|_\infty$	$1.71823e-002$
Tempo de CPU (s)	$0.7747\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.02$; w=1.9	
Número de iterações	263
Variação relativa máxima no SOR	$9.87492e-007$
$\|$erro absoluto$\|_\infty$	$1.71823e-002$
Tempo de CPU (s)	$0.5767\ s$
Passo espacial: $hx = 0.01 \quad hy = 0.005$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	$4.96521e-002$
$\|$erro absoluto$\|_\infty$	$2.05636e-004$
Tempo de CPU (s)	$320.1\ s$
Passo espacial: $hx = 0.005 \quad hy = 0.01$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	2.07068
$\|$erro absoluto$\|_\infty$	$205636e-004$
Tempo de CPU (s)	$320.9\ s$

Fonte: O autor.

Figura 3.6 – Solução manufaturada do sexto problema 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-2}$, $w = 1.9$ e 100001 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

3.2 TRIDIMENSIONAIS

Construímos quatro soluções manufaturadas para a equação de Poisson tridimensional. Nas três primeiras (problemas um a três), testamos condições de contorno de Dirichlet; na quarta (problema quatro), condições de contorno de Neumann.

Nos testes com condições de contorno de Dirichlet, aplicados o operador Laplaciano discreto (2.75) nos pontos interiores da malha, mantendo a solução exata nos pontos de fronteira. Já no teste com condições de contorno de Neumann, aplicamos o operador Laplaciano discreto (2.75) em todos os pontos da malha. Neste caso, precisamos calcular valores para pontos fantasmas. Para atribuir valores para estes pontos, empregamos a discretização centrada de segunda ordem para a derivada primeira, definida em (2.16). Isolando nesta expressão o ponto fantasma, obtemos, para cada uma das fronteiras:

$$\begin{aligned}
\text{Fronteira inferior :} \quad u_{i,j,0} &= u_{i,j,2} - 2\Delta z \frac{\partial u}{\partial z}; \\
\text{Fronteira superior :} \quad u_{i,j,l+1} &= u_{i,j,l-1} + 2\Delta z \frac{\partial u}{\partial z}; \\
\text{Fronteira esquerda :} \quad u_{0,j,k} &= u_{2,j,k} - 2\Delta x \frac{\partial u}{\partial x}; \\
\text{Fronteira direita :} \quad u_{m+1,j,k} &= u_{m-1,j,k} + 2\Delta x \frac{\partial u}{\partial x}; \\
\text{Fronteira frontal :} \quad u_{i,0,k} &= u_{i,2,k} - 2\Delta y \frac{\partial u}{\partial y}; \\
\text{Fronteira posterior :} \quad u_{i,n+1,k} &= u_{i,n-1,k} + 2\Delta y \frac{\partial u}{\partial y}.
\end{aligned}$$

onde $m, n, l \in \mathbb{N}$ representam o número de pontos da malha nas direções x, y e z, respectivamente.

Para problemas tridimensionais com condições de contorno de Neumann em todas as fronteiras, a condição de compatibilidade (NÓS; ROMA; CENICEROS, 2005; STRIKWERDA, 1989) é dada por:

$$\iiint_{\Omega} f = \iint_{\partial\Omega} g, \tag{3.4}$$

sendo $g = \dfrac{\partial u}{\partial \eta}$ a condição de contorno de Neumann, e η a normal à fronteira.

1. **Primeiro problema**

$$\text{Solução exata : } u(x,y,z) = xyz.$$

Equação de Laplace :

$$\frac{\partial^2}{\partial x^2}u(x,y,z)+\frac{\partial^2}{\partial y^2}u(x,y,z) + \frac{\partial^2}{\partial z^2}u(x,y,z) = 0.$$

Domínio: $[0,1]\times[0,1]\times[0,1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior : } u(x,y,0) = 0;$$
$$\text{Fronteira superior : } u(x,y,1) = xy;$$
$$\text{Fronteira esquerda : } u(0,y,z) = 0;$$
$$\text{Fronteira direita : } u(1,y,z) = yz;$$
$$\text{Fronteira frontal : } u(x,0,z) = 0;$$
$$\text{Fronteira posterior : } u(x,1,z) = xz.$$

Condição inicial para os pontos interiores do domínio: $u(x,y,z) = 0$.

Tabela 3.13 – Solução numérica do primeiro problema 3D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = hz = 0.1$	
Número de iterações	145
$\|$erro absoluto$\|_\infty$	$1.66916e-007$
Tempo de CPU (s)	$2.265\ s$
Passo espacial: $hx = hy = hz = 0.02$	
Número de iterações	2676
$\|$erro absoluto$\|_\infty$	$7.06986e-006$
Tempo de CPU (s)	$163.7\ s$
Passo espacial: $hx = hy = hz = 0.01$	
Número de iterações	9200
$\|$erro absoluto$\|_\infty$	$3.03518e-005$
Tempo de CPU (s)	$8673\ s$
Passo espacial: $hx = 0.02\ \ hy = 0.1\ \ hz = 0.1$	
Número de iterações	1061
$\|$erro absoluto$\|_\infty$	$2.54059e-006$
Tempo de CPU (s)	$2.743\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.02\ \ hz = 0.1$	
Número de iterações	1061
$\|$erro absoluto$\|_\infty$	$2.54059e-006$
Tempo de CPU (s)	$3.124\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.1\ \ hz = 0.02$	
Número de iterações	1061
$\|$erro absoluto$\|_\infty$	$2.54059e-006$
Tempo de CPU (s)	$2.798\ s$

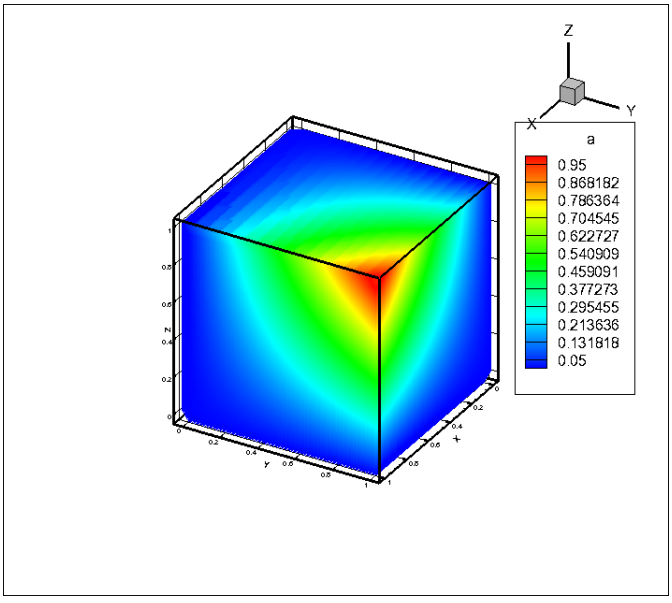
Fonte: O autor.

Tabela 3.14 – Solução numérica do primeiro problema 3D pelo método SOR

Passo espacial: $hx = hy = hz = 0.1$; w=1.6	
Número de iterações	46
$\|$erro absoluto$\|_\infty$	$8.40304e-010$
Tempo de CPU (s)	$0.991\ s$
Passo espacial: $hx = hy = hz = 0.02$; w=1.9	
Número de iterações	233
$\|$erro absoluto$\|_\infty$	$9.27537e-010$
Tempo de CPU (s)	$19.23\ s$
Passo espacial: $hx = hy = hz = 0.01$; w=1.9	
Número de iterações	694
$\|$erro absoluto$\|_\infty$	$3.85123e-007$
Tempo de CPU (s)	$421.6\ s$
Passo espacial: $hx = 0.02\ \ hy = 0.1\ \ hz = 0.1$; w=1.9	
Número de iterações	225
$\|$erro absoluto$\|_\infty$	$1.66474e-010$
Tempo de CPU (s)	$3.785\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.02\ \ hz = 0.1$; w=1.9	
Número de iterações	225
$\|$erro absoluto$\|_\infty$	$1.66474e-010$
Tempo de CPU (s)	$1.369\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.1\ \ hz = 0.02$; w=1.9	
Número de iterações	225
$\|$erro absoluto$\|_\infty$	$1.66474e-010$
Tempo de CPU (s)	$1.377\ s$

Fonte: O autor.

Figura 3.7 – Solução numérica $a[i][j][k]$ do primeiro problema 3D no domínio $[0,1]\times[0,1]\times[0,1]$, usando SOR, com $hx = hy = hz = 10^{-2}$, $w = 1.9$ e 694 iterações



Fonte: O autor com o Tecplot 360 (2021).

2. **Segundo problema**

$$\text{Solução exata : } u(x,y,z) = sen(x)y + cos(y)z + e^z x.$$

Equação de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y,z) + \frac{\partial^2}{\partial y^2}u(x,y,z) + \frac{\partial^2}{\partial z^2}u(x,y,z) = -sen(x)y - cos(y)z + e^z x.$$

Domínio: $[0,1]\times[0,1]\times[0,1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior : } u(x,y,0) = sen(x)y + x;$$
$$\text{Fronteira superior : } u(x,y,1) = sen(x)y + cos(y) + ex;$$
$$\text{Fronteira esquerda : } u(0,y,z) = cos(y)z;$$
$$\text{Fronteira direita : } u(1,y,z) = sen(1)y + cos(y)z + e^z;$$
$$\text{Fronteira frontal : } u(x,0,z) = z + e^z x;$$
$$\text{Fronteira posterior : } u(x,1,z) = sen(x) + cos(1)z + e^z x.$$

Condição inicial para os pontos interiores do domínio: $u(x,y,z) = 0$.

Tabela 3.15 – Solução numérica do segundo problema 3D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = hz = 0.1$	
Número de iterações	129
$\|$erro absoluto$\|_\infty$	$6.87848e-005$
Tempo de CPU (s)	$0.6132\ s$
Passo espacial: $hx = hy = hz = 0.02$	
Número de iterações	2351
$\|$erro absoluto$\|_\infty$	$2.94193e-004$
Tempo de CPU (s)	$80.5\ s$
Passo espacial: $hx = hy = hz = 0.01$	
Número de iterações	7968
$\|$erro absoluto$\|_\infty$	$1.21129e-003$
Tempo de CPU (s)	$2497\ s$
Passo espacial: $hx = 0.02\ \ hy = 0.1\ \ hz = 0.1$	
Número de iterações	945
$\|$erro absoluto$\|_\infty$	$5.108e-005$
Tempo de CPU (s)	$1.48\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.02\ \ hz = 0.1$	
Número de iterações	939
$\|$erro absoluto$\|_\infty$	$6.17376e-005$
Tempo de CPU (s)	$1.368\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.1\ \ hz = 0.02$	
Número de iterações	945
$\|$erro absoluto$\|_\infty$	$7.3711e-005$
Tempo de CPU (s)	$1.578\ s$

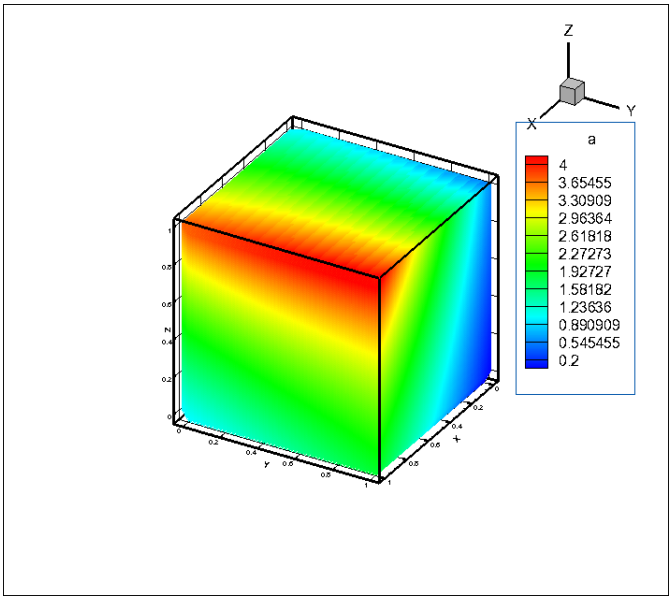
Fonte: O autor.

Tabela 3.16 – Solução numérica do segundo problema 3D pelo método SOR

Passo espacial: $hx = hy = hz = 0.1$; w=1.6	
Número de iterações	38
$\|$erro absoluto$\|_\infty$	$7.58368e-005$
Tempo de CPU (s)	$0.6274\ s$
Passo espacial: $hx = hy = hz = 0.02$; w=1.9	
Número de iterações	179
$\|$erro absoluto$\|_\infty$	$3.16834e-006$
Tempo de CPU (s)	$7.263\ s$
Passo espacial: $hx = hy = hz = 0.01$; w=1.9	
Número de iterações	576
$\|$erro absoluto$\|_\infty$	$3.94529e-005$
Tempo de CPU (s)	$163.4\ s$
Passo espacial: $hx = 0.02$ $hy = 0.1$ $hz = 0.1$; w=1.9	
Número de iterações	175
$\|$erro absoluto$\|_\infty$	$6.53754e-005$
Tempo de CPU (s)	$0.7885\ s$
Passo espacial: $hx = 0.1$ $hy = 0.02$ $hz = 0.1$; w=1.9	
Número de iterações	174
$\|$erro absoluto$\|_\infty$	$5.62796e-005$
Tempo de CPU (s)	$0.7039\ s$
Passo espacial: $hx = 0.1$ $hy = 0.1$ $hz = 0.02$; w=1.9	
Número de iterações	175
$\|$erro absoluto$\|_\infty$	$3.43253e-005$
Tempo de CPU (s)	$0.775\ s$

Fonte: O autor.

Figura 3.8 – Solução numérica $a[i][j][k]$ do segundo problema 3D no domínio $[0,1]\times[0,1]\times[0,1]$, usando SOR, com $hx = hy = hz = 10^{-2}$, $w = 1.9$ e 576 iterações



Fonte: O autor com o Tecplot 360 (2021).

3. **Terceiro problema**

$$\text{Solução exata : } u(x,y,z) = x^2e^y + y^2sen(z) + z^2cos(x).$$

Equação de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y,z) + \frac{\partial^2}{\partial y^2}u(x,y,z) + \frac{\partial^2}{\partial z^2}u(x,y,z) = 2\left(e^y + sin(z) + cos(x)\right) + \\ -z^2cos(x) + x^2e^y - y^2sin(z)).$$

Domínio: $[0,1]\times[0,1]\times[0,1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior : } u(x,y,0) = x^2e^y;$$
$$\text{Fronteira superior : } u(x,y,1) = x^2e^y + y^2sen(1) + cos(x);$$
$$\text{Fronteira esquerda : } u(0,y,z) = y^2sen(z) + z^2;$$
$$\text{Fronteira direita : } u(1,y,z) = e^y + y^2sen(z) + z^2cos(1);$$
$$\text{Fronteira frontal : } u(x,0,z) = x^2 + z^2cos(x);$$
$$\text{Fronteira posterior : } u(x,1,z) = x^2e + sen(z) + z^2cos(x).$$

Condição inicial para os pontos interiores do domínio: $u(x,y,z) = 0$.

Tabela 3.17 – Solução numérica do terceiro problema 3D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = hz = 0.1$	
Número de iterações	138
$\|$erro absoluto$\|_\infty$	$4.49052e-005$
Tempo de CPU (s)	$0.6567\ s$
Passo espacial: $hx = hy = hz = 0.02$	
Número de iterações	2523
$\|$erro absoluto$\|_\infty$	$8.92937e-005$
Tempo de CPU (s)	$125.2\ s$
Passo espacial: $hx = hy = hz = 0.01$	
Número de iterações	8626
$\|$erro absoluto$\|_\infty$	$3.78902e-004$
Tempo de CPU (s)	$3534\ s$
Passo espacial: $hx = 0.02\ \ hy = 0.1\ \ hz = 0.1$	
Número de iterações	1008
$\|$erro absoluto$\|_\infty$	$2.14633e-005$
Tempo de CPU (s)	$2.499\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.02\ \ hz = 0.1$	
Número de iterações	1004
$\|$erro absoluto$\|_\infty$	$1.70853e-005$
Tempo de CPU (s)	$2.049\ s$
Passo espacial: $hx = 0.1\ \ hy = 0.1\ \ hz = 0.02$	
Número de iterações	1006
$\|$erro absoluto$\|_\infty$	$2.23853e-005$
Tempo de CPU (s)	$2.888\ s$

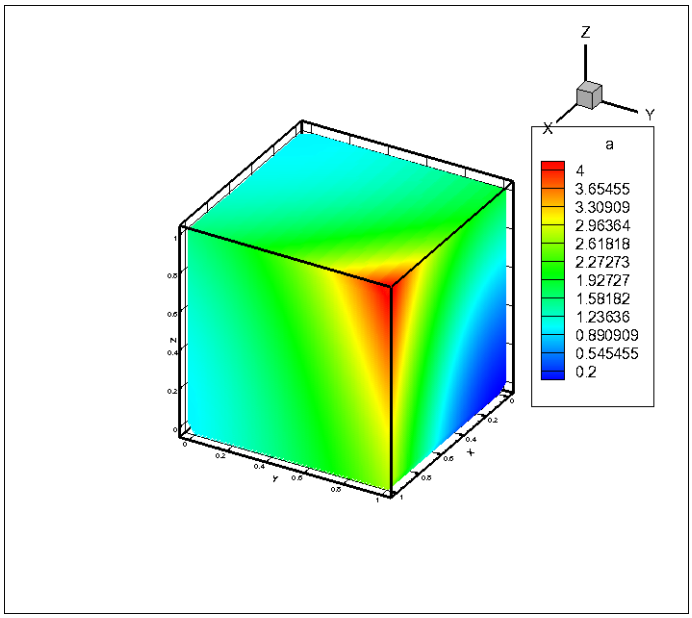
Fonte: O autor.

Tabela 3.18 – Solução numérica do terceiro problema 3D pelo método SOR

Passo espacial: $hx = hy = hz = 0.1$; w=1.6	
Número de iterações	43
$\|$erro absoluto$\|_\infty$	$4.62281e-005$
Tempo de CPU (s)	$0.8859\ s$
Passo espacial: $hx = hy = hz = 0.02$; w=1.9	
Número de iterações	219
$\|$erro absoluto$\|_\infty$	$1.91005e-006$
Tempo de CPU (s)	$18.55\ s$
Passo espacial: $hx = hy = hz = 0.01$; w=1.9	
Número de iterações	635
$\|$erro absoluto$\|_\infty$	$7.67582e-006$
Tempo de CPU (s)	$258.2\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1 \quad hz = 0.1$; w=1.9	
Número de iterações	184
$\|$erro absoluto$\|_\infty$	$3.64771e-005$
Tempo de CPU (s)	$0.9684\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.02 \quad hz = 0.1$; w=1.9	
Número de iterações	179
$\|$erro absoluto$\|_\infty$	$2.30278e-005$
Tempo de CPU (s)	$0.6102\ s$
Passo espacial: $hx = 0.1 \quad hy = 0.1 \quad hz = 0.02$; w=1.9	
Número de iterações	184
$\|$erro absoluto$\|_\infty$	$3.9744e-005$
Tempo de CPU (s)	$0.9077\ s$

Fonte: O autor.

Figura 3.9 – Solução numérica $a[i][j][k]$ do terceiro problema 3D no domínio $[0,1]\times[0,1]\times[0,1]$, usando SOR, com $hx = hy = hz = 10^{-2}$, $w = 1.9$ e 635 iterações



Fonte: O autor com o Tecplot 360 (2021).

4. **Quarto problema**

$$\text{Solução exata : } u(x,y,z) = cos(2\pi x)cos(2\pi y)cos(2\pi z).$$

Equação de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y,z) + \frac{\partial^2}{\partial y^2}u(x,y,z) + \frac{\partial^2}{\partial z^2}u(x,y,z) = -12\pi^2 cos(2\pi x)cos(2\pi y)cos(2\pi z).$$

Domínio: $[0,1]\times[0,1]\times[0,1]$.

Condições de contorno de Neumann:

$$\text{Fronteira inferior : } -\frac{\partial}{\partial z}u(x,y,0) = 2\pi sen\,(2\pi(0))\,cos(2\pi x)cos(2\pi y) = 0;$$

$$\text{Fronteira superior : } \frac{\partial}{\partial z}u(x,y,1) = -2\pi sen\,(2\pi(1))\,cos(2\pi x)cos(2\pi y) = 0;$$

$$\text{Fronteira esquerda : } -\frac{\partial}{\partial x}u(0,y,z) = 2\pi sen\,(2\pi(0))\,cos(2\pi y)cos(2\pi z) = 0;$$

$$\text{Fronteira direita : } \frac{\partial}{\partial x}u(1,y,z) = -2\pi sen\,(2\pi(1))\,cos(2\pi y)cos(2\pi z) = 0;$$

$$\text{Fronteira frontal : } -\frac{\partial}{\partial y}u(x,0,z) = 2\pi sen\,(2\pi(0))\,cos(2\pi x)cos(2\pi z) = 0;$$

$$\text{Fronteira posterior : } \frac{\partial}{\partial y}u(x,1,z) = -2\pi sen\,(2\pi(1))\,cos(2\pi x)cos(2\pi z) = 0.$$

Condição inicial para os pontos interiores do domínio: $u(x,y,z) = 0$.

Tabela 3.19 – Solução numérica do quarto problema 3D pelo método de Gauss-Seidel

Passo espacial: $hx = hy = hz = 0.1$	
Número de iterações	133
Variação relativa máxima no GS	$9.59132e-007$
$\|$erro absoluto$\|_\infty$	$3.35588e-002$
Tempo de CPU (s)	$0.61\ s$
Passo espacial: $hx = hy = hz = 0.02$	
Número de iterações	1858
Variação relativa máxima no GS	$9.99903e-007$
$\|$erro absoluto$\|_\infty$	$1.31707e-003$
Tempo de CPU (s)	$73.12\ s$
Passo espacial: $hx = hy = hz = 0.0125$	
Número de iterações	32173
Variação relativa máxima no GS	$6.41806e-007$
$\|$erro absoluto$\|_\infty$	$5.142e-004$
Tempo de CPU (s)	$5073\ s$
Passo espacial: $hx = 0.02\ \ hy = 0.1\ \ hz = 0.1$	
Número de iterações	513
Variação relativa máxima no GS	$9.86846e-007$
$\|$erro absoluto$\|_\infty$	$2.25835e-002$
Tempo de CPU (s)	$1.503\ s$

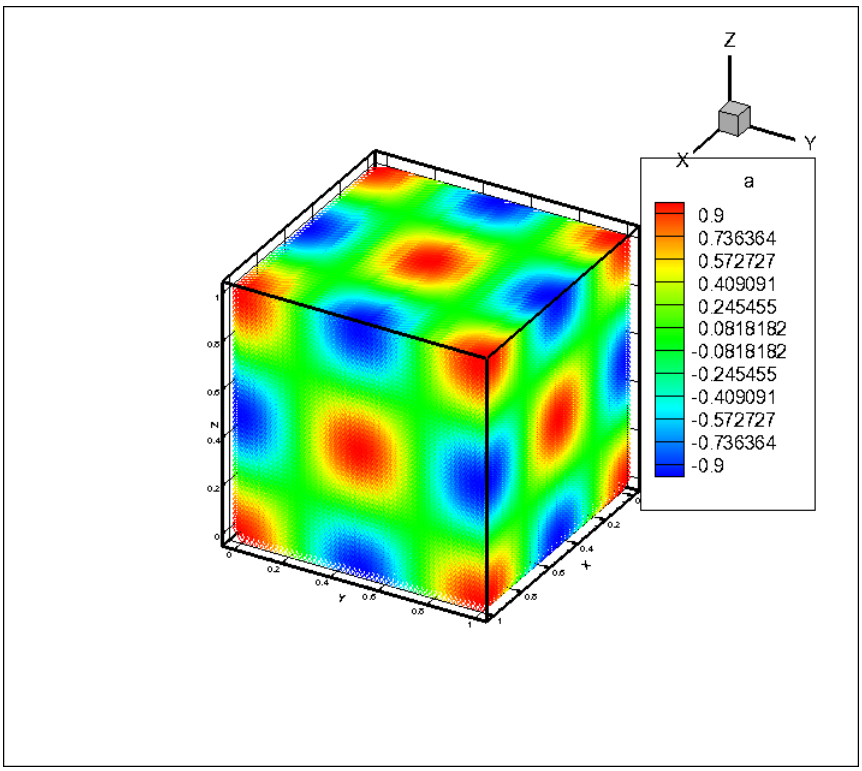
Fonte: O autor.

Tabela 3.20 – Solução numérica do quarto problema 3D pelo método SOR

Passo espacial: $hx = hy = hz = 0.1$; w=1.6	
Número de iterações	92
Variação relativa máxima no SOR	$9.64194e-007$
$\|$erro absoluto$\|_\infty$	$3.35588e-002$
Tempo de CPU (s)	$0.5834\ s$
Passo espacial: $hx = hy = hz = 0.02$; w=1.9	
Número de iterações	620
Variação relativa máxima no SOR	$9.82617e-007$
$\|$erro absoluto$\|_\infty$	$1.31701e-003$
Tempo de CPU (s)	$34.43\ s$
Passo espacial: $hx = hy = hz = 0.0125$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	24013.4
$\|$erro absoluto$\|_\infty$	$5.142e-004$
Tempo de CPU (s)	$1.575e+004\ s$
Passo espacial: $hx = 0.02 \quad hy = 0.1 \quad hz = 0.1$; w=1.9	
Número de iterações	295
Variação relativa máxima no SOR	$9.85619e-007$
$\|$erro absoluto$\|_\infty$	$2.2583e-002$
Tempo de CPU (s)	$0.9348\ s$

Fonte: O autor.

Figura 3.10 – Solução numérica $a[i][j][k]$ do quarto problema 3D no domínio $[0,1]\times[0,1]\times[0,1]$, usando SOR, com $hx = hy = hz = 2\text{x}10^{-2}$, $w = 1.9$ e 620 iterações



Fonte: O autor com o Tecplot 360 (2021).

3.3 ANÁLISE DAS SIMULAÇÕES

Na análise das Tabelas 3.1 a 3.20, com resultados ilustrados nas Figuras 3.1 a 3.10, a eficiência do método está atrelada a quatro parâmetros: número de iterações; variação relativa máxima; erro absoluto; tempo de CPU.

3.3.1 BIDIMENSIONAIS

Analisando os testes com condições de contorno de Dirichlet (problemas um a cinco), concluímos que:

1. o método SOR, com w adequado, mostrou-se mais eficiente do que o método de Gauss-Seidel quanto aos parâmetros número de iterações e tempo de CPU, determinando, geralmente, erros absolutos menores;

2. no método SOR, os parâmetros de relaxação $w = 1.6$ e $w = 1.9$ foram mais eficientes (aceleração da convergência), respectivamente, em malhas mais grossas e em malhas mais finas;

3. o método SOR, com w adequado, foi eficiente em malhas isotrópicas e anisotrópicas;

4. com um maior refinamento da malha, os erros absolutos aumentaram.

Quanto ao teste com condições de contorno de Neumann (problema seis), a condição de compatibilidade (3.3) foi atendida na construção da solução manufaturada, ou seja,

$$\int_0^1\int_0^1 cos(2\pi x)cos(2\pi y)dydx = \int_{\partial\Omega} g = 0.$$

Neste teste, os métodos de Gauss-Seidel e SOR são ineficientes em uma malha 500x500, uma vez que a variação relativa máxima é grande ($>> 10^{-6}$) e o número de interações é o valor máximo imposto (ITMAX = 10^5). Em uma malha 100x100, o método de Gauss-Seidel é eficiente, o que equivale ao método SOR com $w = 1$.

3.3.2 TRIDIMENSIONAIS

Nos testes com condições de contorno de Dirichlet (problemas uma a três), constatamos que:

1. o método SOR, com w adequado, mostrou-se mais eficiente do que o método de Gauss-Seidel quanto aos parâmetros número de iterações e tempo de CPU, determinando, geralmente, erros absolutos menores;

2. no método SOR, os parâmetros de relaxação $w = 1.6$ e $w = 1.9$ foram mais eficientes (aceleração da convergência), respectivamente, em malhas mais grossas e em malhas mais finas;

3. o método SOR, com w adequado, foi eficiente em malhas isotrópicas e anisotrópicas;

4. com um maior refinamento da malha, os erros absolutos aumentaram.

Quanto ao teste com condições de contorno de Neumann (problema quatro), a condição de compatibilidade (3.4) foi respeitada na construção da solução manufaturada, isto é,

$$\int_0^1\int_0^1\int_0^1 cos(2\pi x)cos(2\pi y)cos(2\pi z)dzdydx = \iint_{\partial\Omega} g = 0.$$

Neste teste, o método SOR é ineficiente em uma malha 80x80x80, pois a variação relativa máxima é grande ($>> 10^{-6}$) e o número de interações é o valor máximo imposto (ITMAX = 10^5). Já o método de Gauss-Seidel foi eficiente nessa malha.

4 SIMULAÇÕES COMPUTACIONAIS: PROBLEMAS SINGULARES 2D

Neste capítulo, simulamos numericamente cinco problemas elípticos bidimensionais. Os quatro primeiros, com solução exata conhecida, foram propostos por Mitchell (2017) para testar singularidades em malhas adaptativas; o quinto é uma adaptação do problema de transmissão de calor analisado por Gpsgui (2019), com solução exata desconhecida.

4.1 SOLUÇÃO ANALÍTICA SUAVE

Este é um problema bem comportado, isto é, um problema onde pequenas variações nos parâmetros implicam em pequenas variações na solução. O número inteiro a determina o grau do polinômio na solução exata.
Solução exata: $u(x,y) = 2^{4a}x^a(1-x)^a y^a(1-y)^a$.
Solução exata para $a = 10$: $u(x,y) = 2^{40}x^{10}(1-x)^{10}y^{10}(1-y)^{10}$.
Equação de Poisson para $a = 10$:

$$\begin{aligned}\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) =& 2^{40}y^{10}(1-y)^{10}\left(90x^8(1-x)^{10} - 200x^9(1-x)^9 + 90x^{10}(1-x)^8\right) + \\ &+2^{40}x^{10}(1-x)^{10}\left(90y^8(1-y)^{10} - 200y^9(1-y)^9 + 90y^{10}(1-y)^8\right).\end{aligned}$$

Domínio: $[0,1]\times[0,1]$.
Condições de contorno de Dirichlet:

$$\begin{aligned}\text{Fronteira inferior} &: u(x,0) = 0;\\ \text{Fronteira superior} &: u(x,1) = 0;\\ \text{Fronteira esquerda} &: u(0,y) = 0;\\ \text{Fronteira direita} &: u(1,y) = 0.\end{aligned}$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

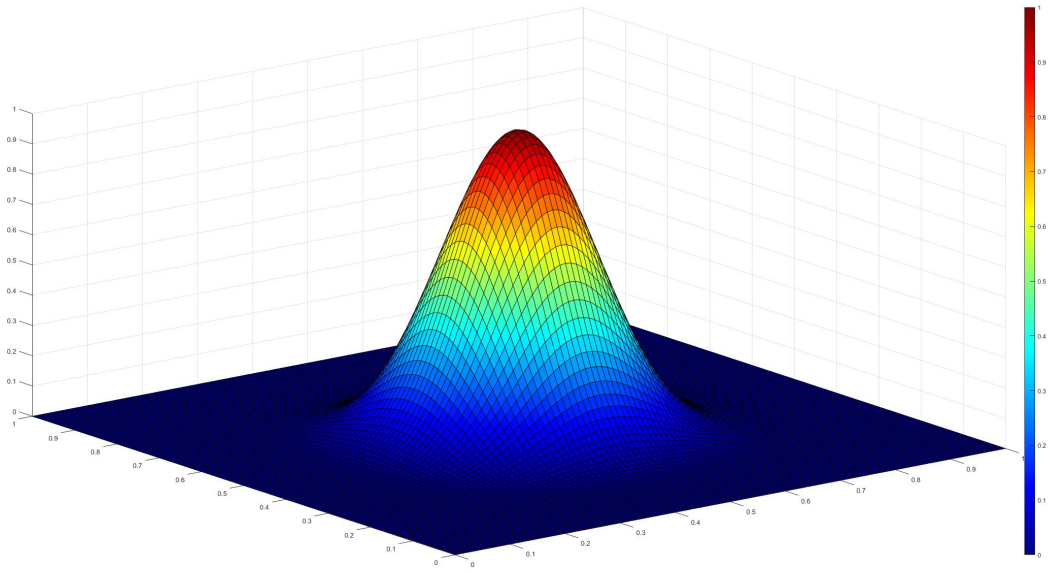
Tabela 4.1 – Solução numérica do problema *Solução analítica suave* 2D pelo método SOR

Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	37663
Variação relativa máxima no SOR	$9.99497e-007$
$\|\text{erro absoluto}\|_\infty$	$3.92699e-005$
Tempo de CPU (s)	$3148\ s$
Passo espacial: $hx = hy = 0.001$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	$8.18913e-002$
$\|\text{erro absoluto}\|_\infty$	$9.81529e-006$
Tempo de CPU (s)	$3.013e+004\ s$

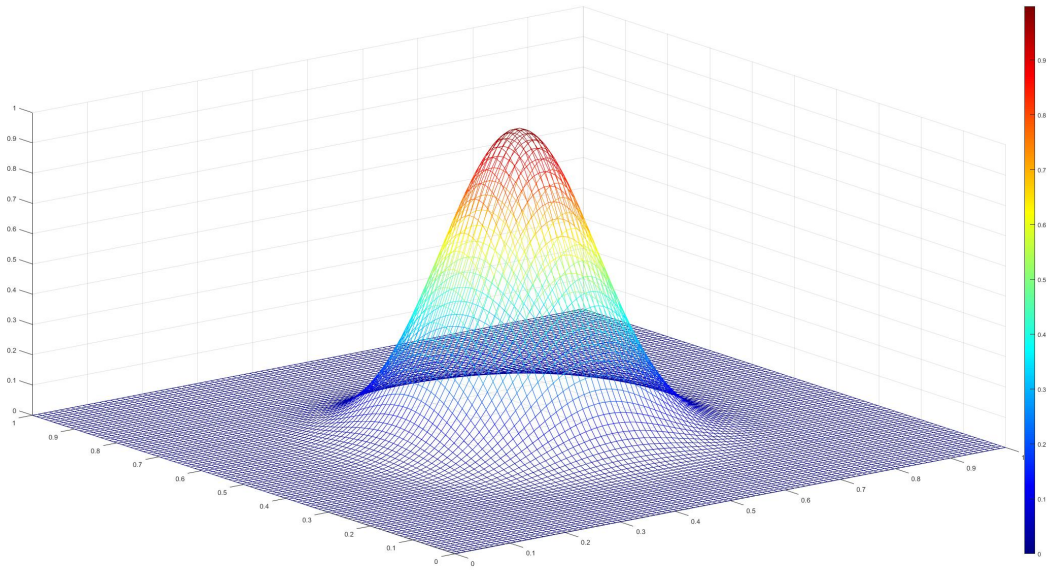
Fonte: O autor.

A Tabela 4.1 mostra os resultados das simulações em malhas uniformes 500x500 e 1000x1000 obtidas com o método SOR. As soluções exata e numérica são ilustradas, respectivamente, nas Figuras 4.1(a) e 4.1(b).

Figura 4.1 – Solução do problema *Solução analítica suave* 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-3}$, $w = 1.9$ e 100001 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

4.2 PICO

Este problema apresenta um pico exponencial no interior do domínio. O par ordenado (x_c, y_c) sinaliza a localização do pico, enquanto o parâmetro α determina a amplitude do pico.
Solução exata: $u(x,y) = e^{-\alpha((x-x_c)^2+(y-y_c)^2)}$.
Solução exata para $\alpha = 1000$ e $(x_c, y_c) = (0.5, 0.5)$: $u(x,y) = e^{-1000((x-0.5)^2+(y-0.5)^2)}$.
Equação de Poisson para $\alpha = 1000$ e $(x_c, y_c) = (0.5, 0.5)$:

$$\begin{aligned}\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) =& 4000000e^{-1000\left(x^2-x+y^2-y+0.5\right)}x^2 + \\ & -4000000e^{-1000\left(x^2-x+y^2-y+0.5\right)}x + \\ & +4000000e^{-1000\left(x^2-x+y^2-y+0.5\right)}y^2 + \\ & -4000000e^{-1000\left(x^2-x+y^2-y+0.5\right)}y + \\ & +1996000e^{-1000(x^2-x+y^2-y+0.5)}.\end{aligned}$$

Domínio: $[0,1]\times[0,1]$.
Condições de contorno de Dirichlet:

$$\begin{aligned}\text{Fronteira inferior} &: u(x,0) = e^{-1000((x-0.5)^2+(-0.5)^2)};\\ \text{Fronteira superior} &: u(x,1) = e^{-1000((x-0.5)^2+(0.5)^2)};\\ \text{Fronteira esquerda} &: u(0,y) = e^{-1000((-0.5)^2+(y-0.5)^2)};\\ \text{Fronteira direita} &: u(1,y) = e^{-1000((0.5)^2+(y-0.5)^2)}.\end{aligned}$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

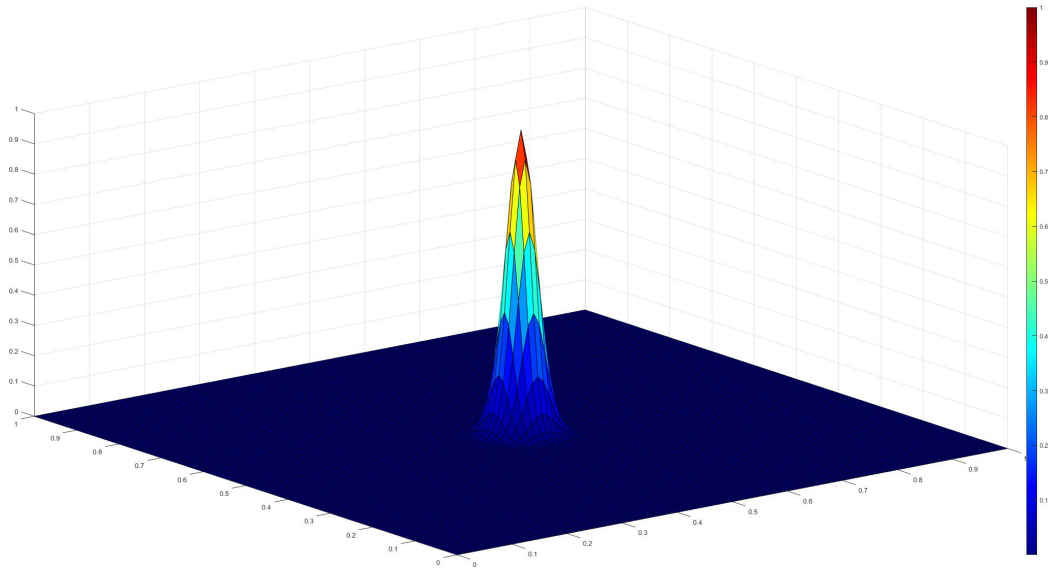
Tabela 4.2 – Solução numérica do problema *Pico* 2D pelo método SOR

Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	36674
Variação relativa máxima no SOR	$9.99412e-007$
$\|\text{erro absoluto}\|_\infty$	$1.00121e-003$
Tempo de CPU (s)	$3434\ s$
Passo espacial: $hx = hy = 0.001$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	$3.74592e-002$
$\|\text{erro absoluto}\|_\infty$	$2.50074e-004$
Tempo de CPU (s)	$3.68e+004\ s$

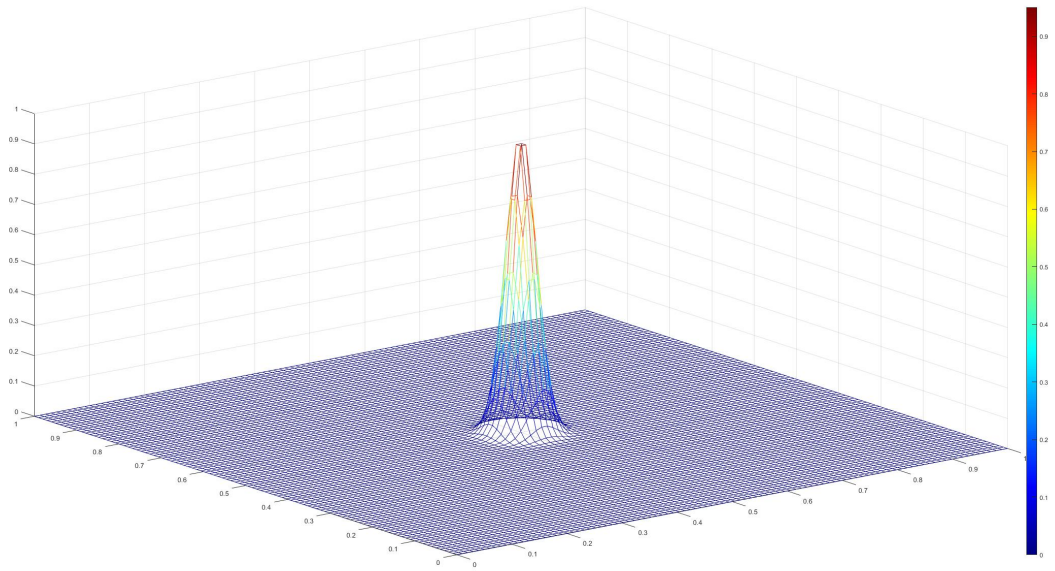
Fonte: O autor.

A Tabela 4.2 mostra os resultados das simulações em malhas uniformes 500x500 e 1000x1000 obtidas com o método SOR. As soluções exata e numérica são ilustradas, respectivamente, nas Figuras 4.2(a) e 4.2(b).

Figura 4.2 – Solução do problema *Pico* 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-3}$, $w = 1.9$ e 100001 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

4.3 SINGULARIDADE NA FRONTEIRA ESQUERDA

De acordo com Mitchell (2017), muitos artigos empregam um exemplo 1D com uma singularidade da forma x^{α} no ponto final à esquerda do domínio. Esse exemplo pode ser estendido

para 2D considerando-se a solução constante em y. No quadrado unitário, o resultado é uma solução que é singular ao longo da fronteira esquerda. O parâmetro $\alpha \geq \frac{1}{2}$ indica a "força" da singularidade.

Solução exata: x^{α}.

Solução exata para $\alpha = 0.6$: $u(x, y) = x^{0.6}$.

Equação de Poisson para $\alpha = 0.6$:

$$\frac{\partial^2}{\partial x^2}u(x, y) + \frac{\partial^2}{\partial y^2}u(x, y) = -0.24x^{-1.4}.$$

Domínio: $[0, 1]\times[0, 1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x, 0) = x^{0.6};$$
$$\text{Fronteira superior} : u(x, 1) = x^{0.6};$$
$$\text{Fronteira esquerda} : u(0, y) = 0;$$
$$\text{Fronteira direita} : u(1, y) = 1.$$

Condição inicial para os pontos interiores do domínio: $u(x, y) = 0$.

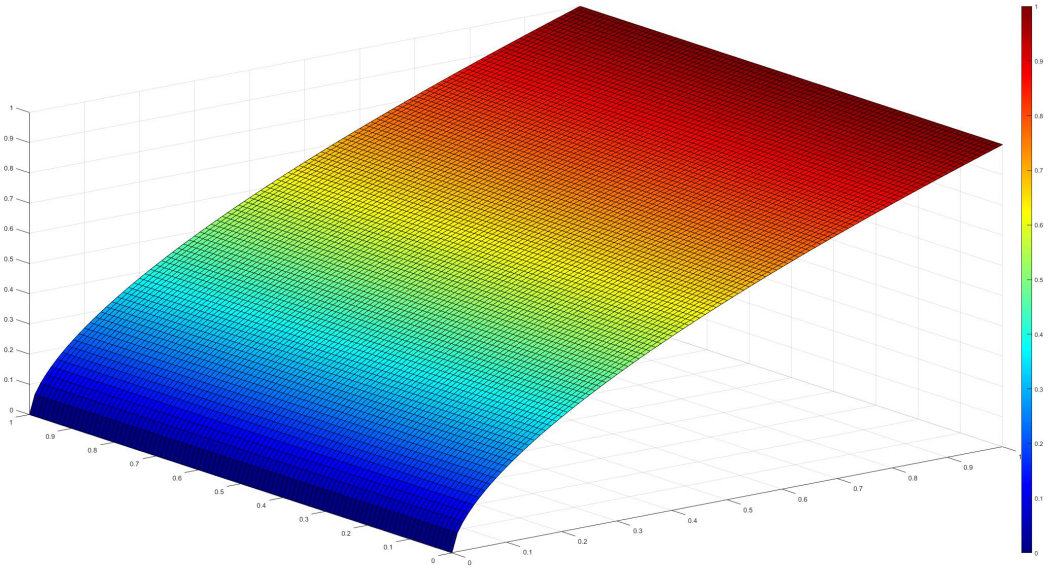
Tabela 4.3 – Solução numérica do problema *Singularidade na fronteira esquerda* 2D pelo método SOR

Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	9651
Variação relativa máxima no SOR	$9.99512e-007$
$\|$erro absoluto$\|_{\infty}$	$6.38978e-003$
Tempo de CPU (s)	$2355\ s$
Passo espacial: $hx = hy = 0.001$; w=1.9	
Número de iterações	31087
Variação relativa máxima no SOR	$9.99899e-007$
$\|$erro absoluto$\|_{\infty}$	$4.58187e-003$
Tempo de CPU (s)	$4763\ s$

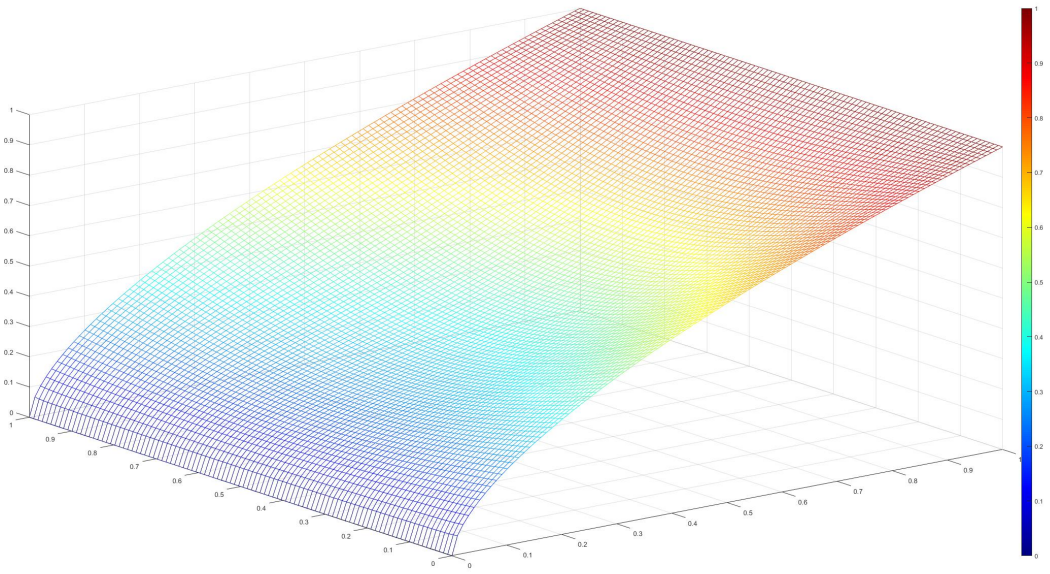
Fonte: O autor.

A Tabela 4.3 mostra os resultados das simulações em malhas uniformes 500x500 e 1000x1000 obtidas com o método SOR. As soluções exata e numérica são ilustradas, respectivamente, nas Figuras 4.3(a) e 4.3(b).

Figura 4.3 – Solução do problema *Singularidade na fronteira esquerda* 2D no domínio $[0,1]\times[0,1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 10^{-3}$, $w = 1.9$ e 100001 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

4.4 SINGULARIDADE INTERIOR

Este problema é uma extensão do problema apresentado na seção 4.3, idealizado para definir uma linha singular inclinada que não coincida com a fronteira. Os parâmetros α e β determinam, respectivamente, a magnitude da singularidade e a inclinação da linha singular.

Solução exata: $u(x,y) = \begin{cases} cos(\pi y/2) & x \leq \beta(y+1) \\ cos(\pi y/2) + (x - \beta(y+1))^{\alpha} & x > \beta(y+1) \end{cases}$.

Domínio: $[-1,1]\times[-1,1]$.

Solução exata para $\alpha = 2.5$ e $\beta = 0$: $u(x,y) = \begin{cases} cos(\pi y/2) & \text{se } -1 \leq x \leq 0 \\ cos(\pi y/2) + x^{2.5} & \text{se } 0 < x \leq 1 \end{cases}$.

Equação de Poisson para $\alpha = 2.5$ e $\beta = 0$:

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = \begin{cases} -\dfrac{\pi^2}{4}cos\left(\dfrac{\pi y}{2}\right) & \text{se } -1 \leq x \leq 0 \\ \\ 3.75x^{1/2} - \dfrac{\pi^2}{4}cos\left(\dfrac{\pi y}{2}\right) & \text{se } 0 < x \leq 1 \end{cases}.$$

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x,-1) = \begin{cases} 0 & \text{se } -1 \leq x \leq 0 \\ x^{2.5} & \text{se } 0 < x \leq 1 \end{cases};$$

$$\text{Fronteira superior} : u(x,1) = \begin{cases} 0 & \text{se } -1 \leq x \leq 0 \\ x^{2.5} & \text{se } 0 < x \leq 1 \end{cases};$$

$$\text{Fronteira esquerda} : u(-1,y) = cos(\pi y/2);$$

$$\text{Fronteira direita} : u(1,y) = cos(\pi y/2) + 1.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

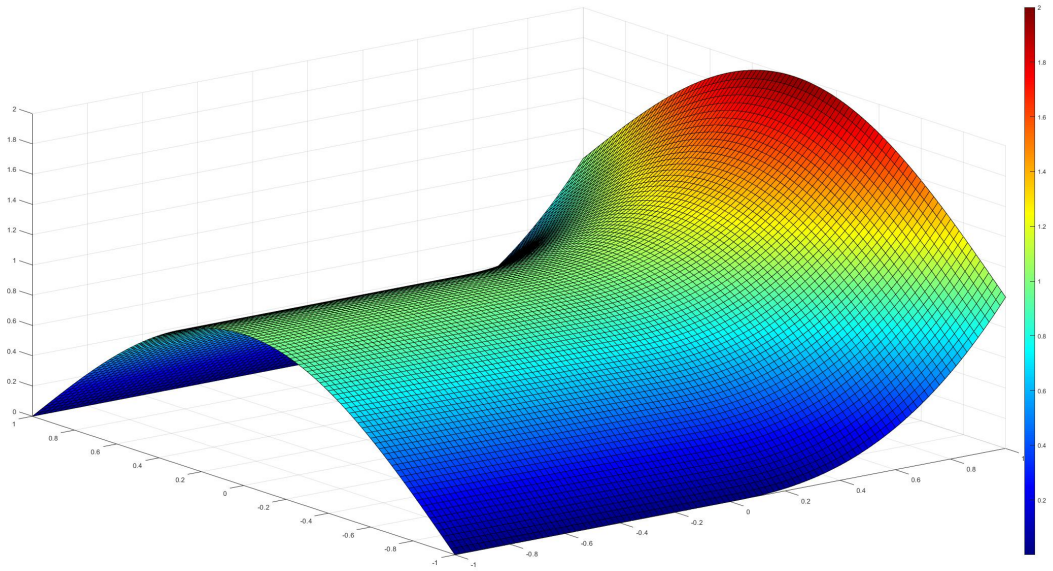
Tabela 4.4 – Solução numérica do problema *Singularidade interior* 2D, com $\alpha = 2.5$ e $\beta = 0$, pelo método SOR

Passo espacial: $hx = hy = 0.004$; w=1.9	
Número de iterações	9403
Variação relativa máxima no SOR	$9.99948e-007$
$\|$erro absoluto$\|_{\infty}$	$1.14475e-003$
Tempo de CPU (s)	$316.7\ s$
Passo espacial: $hx = hy = 0.002$; w=1.9	
Número de iterações	29994
Variação relativa máxima no SOR	$9.99947e-007$
$\|$erro absoluto$\|_{\infty}$	$5.03542e-003$
Tempo de CPU (s)	$3603\ s$

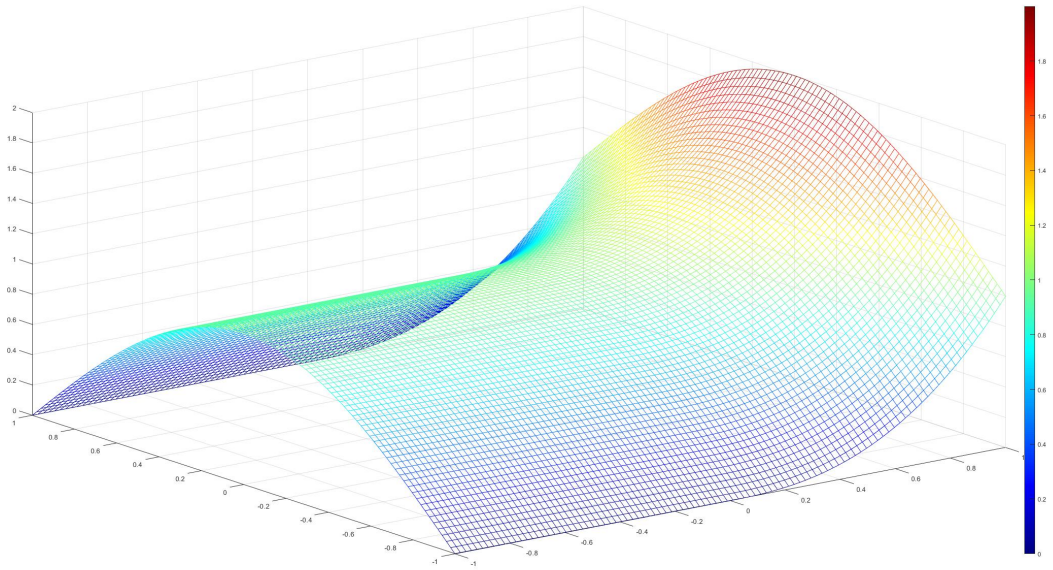
Fonte: O autor.

A Tabela 4.4 mostra os resultados das simulações em malhas uniformes 500x500 e 1000x1000 obtidas com o método SOR. A solução numérica é ilustrada na Figura 4.4. Para os parâmetros $\alpha = 2.5$ e $\beta = 0$, observamos uma linha singular suave em $x = 0$.

Figura 4.4 – Solução do problema *Singularidade interior* 2D, com $\alpha = 2.5$ e $\beta = 0$, no domínio $[-1, 1]\times[-1, 1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 4\text{x}10^{-3}$, $w = 1.9$ e 9403 iterações



(a)



(b)

Fonte: O autor com o Matlab (2021).

Domínio: $[-1,1]\times[-1,1]$.

Solução exata para $\alpha = 1.1$ e $\beta = 0$: $u(x,y) = \begin{cases} cos(\pi y/2) & \text{se } -1 \leq x \leq 0 \\ cos(\pi y/2) + x^{1.1} & \text{se } 0 < x \leq 1 \end{cases}$.

Equação de Poisson para $\alpha = 1.1$ e $\beta = 0$:

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = \begin{cases} -\frac{\pi^2}{4}cos\left(\frac{\pi y}{2}\right) & \text{se } -1 \leq x \leq 0 \\ 0.11x^{-0.9} - \frac{\pi^2}{4}cos\left(\frac{\pi y}{2}\right) & \text{se } 0 < x \leq 1 \end{cases}.$$

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x,-1) = \begin{cases} 0 & \text{se } -1 \leq x \leq 0 \\ x^{1.1} & \text{se } 0 < x \leq 1 \end{cases};$$

$$\text{Fronteira superior} : u(x,1) = \begin{cases} 0 & \text{se } -1 \leq x \leq 0 \\ x^{1.1} & \text{se } 0 < x \leq 1 \end{cases};$$

$$\text{Fronteira esquerda} : u(-1,y) = cos(\pi y/2);$$

$$\text{Fronteira direita} : u(1,y) = cos(\pi y/2) + 1.$$

Condição inicial para os pontos interiores do domínio: $u(x,y) = 0$.

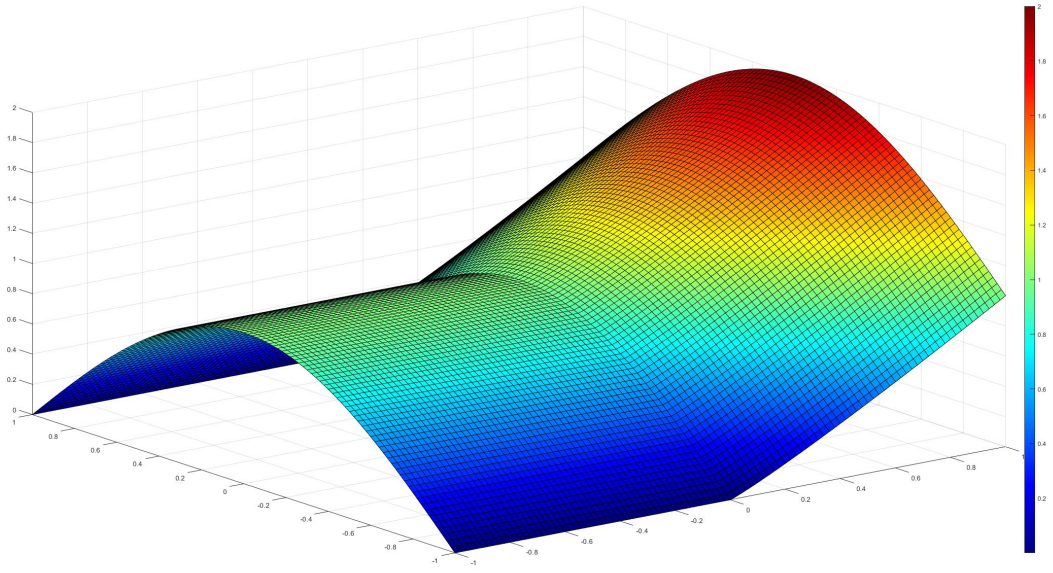
Tabela 4.5 – Solução numérica do problema *Singularidade interior* 2D, com $\alpha = 1.1$ e $\beta = 0$, pelo método SOR

Passo espacial: $hx = hy = 0.004$; w=1.4	
Número de iterações	53164
Variação relativa máxima no SOR	$9.99963e-007$
$\|$erro absoluto$\|_\infty$	$1.88862e-001$
Tempo de CPU (s)	$2214\ s$
Passo espacial: $hx = hy = 0.002$; w=1.4	
Número de iterações	100001
Variação relativa máxima no SOR	$3.85374e-006$
$\|$erro absoluto$\|_\infty$	$4.48575e-002$
Tempo de CPU (s)	$1.654e+004\ s$

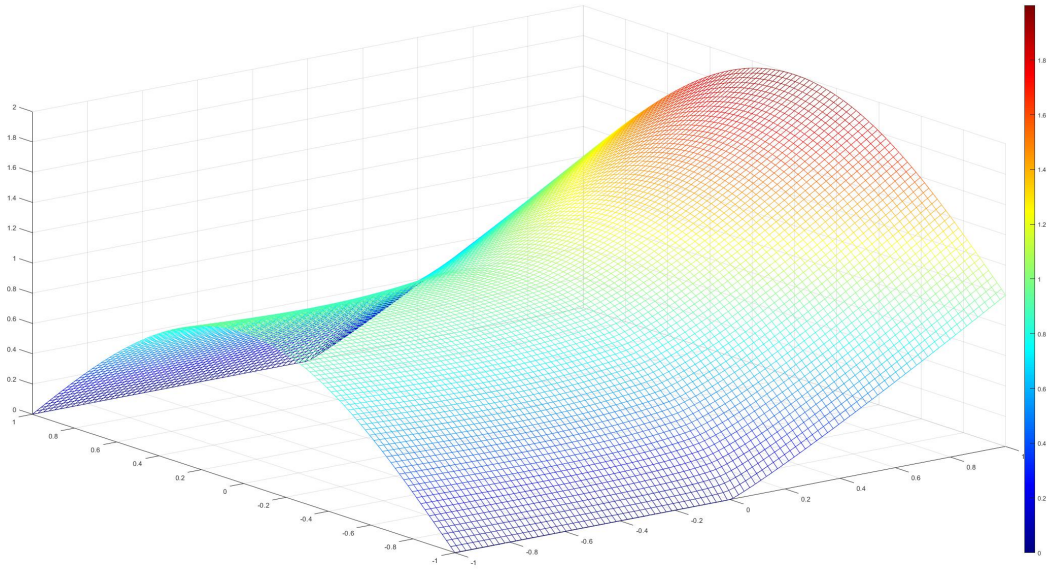
Fonte: O autor.

A Tabela 4.5 mostra os resultados das simulações em malhas uniformes 500x500 e 1000x1000 obtidas com o método SOR. A solução numérica é ilustrada na Figura 4.5. Para os parâmetros $\alpha = 1.1$ e $\beta = 0$, observamos uma acentuada linha singular em $x = 0$.

Figura 4.5 – Solução do problema *Singularidade interior* 2D, com $\alpha = 1.1$ e $\beta = 0$, no domínio $[-1, 1]\times[-1, 1]$: (a) solução exata; (b) solução numérica usando SOR, com $hx = hy = 2\text{x}10^{-3}$, $w = 1.4$ e 100001 iterações



(a)



(b)

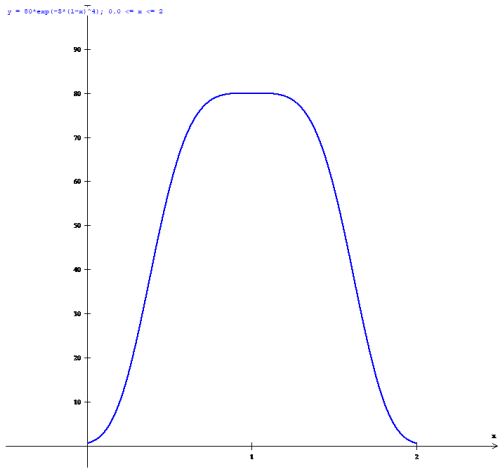
Fonte: O autor com o Matlab (2021).

4.5 DISTRIBUIÇÃO DE TEMPERATURA

Neste problema, adaptado de Gpsgui (2019), temos um canal retangular no qual o calor gerado por uma fonte é dissipado.

Para simular a fonte de calor, usamos a função ilustrada na Figura 4.6 como condição de contorno na fronteira superior do canal. Essa função tem um máximo igual a 80 em $x = 1$. Na simulação numérica, aplicamos o operador Laplaciano (2.70) em todo o domínio, exceto na fronteira superior, que é mantida fixa. Desta forma, queremos observar se a temperatura é distribuída de forma a preservar o valor máximo 80 somente na fronteira superior, onde está localizada a "fonte" de calor.

Figura 4.6 – Condição de contorno $u(x, y) = 80e^{-5(1-x)^4}$ para a fronteira superior do problema *Distribuição de temperatura*



Fonte: O autor com o Winplot (2012).

Na equação de Poisson, a função $f = -\dfrac{5}{10^6}$ modela a dissipação do calor (GPSGUI, 2019).

Solução exata: não há.

Equação de Poisson:

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = -\frac{5}{10^6}.$$

Domínio: $[0, 2]\times[0, 1]$.

Condições de contorno de Dirichlet:

$$\text{Fronteira inferior} : u(x, 0) = 20;$$
$$\text{Fronteira superior} : u(x, 1) = 80e^{-5(1-x)^4};$$
$$\text{Fronteira esquerda} : u(0, y) = 20;$$
$$\text{Fronteira direita} : u(2, y) = 20.$$

Condição inicial para os pontos interiores do domínio: $u(x, y) = 0$.

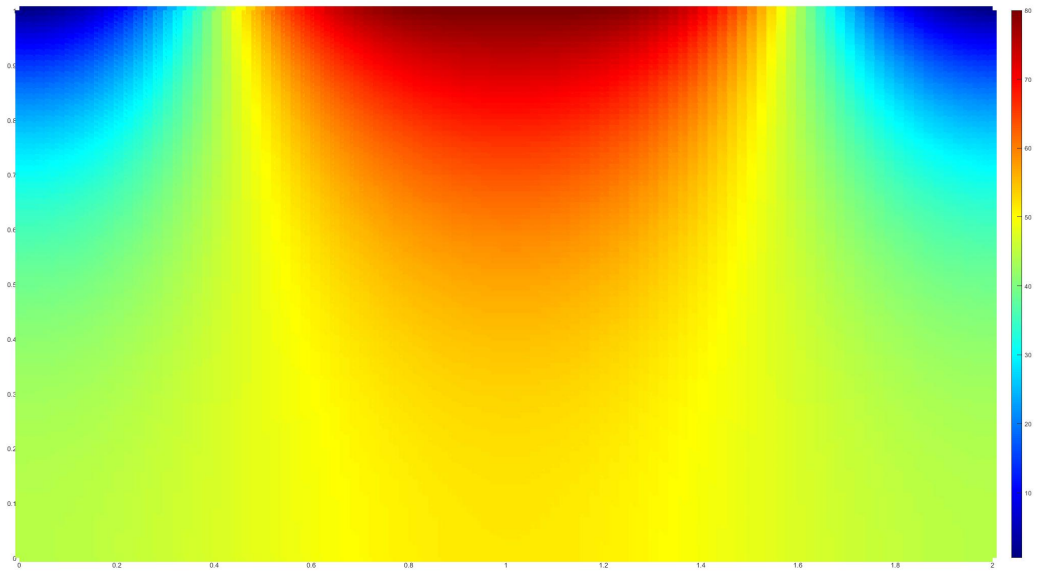
Tabela 4.6 – Solução numérica do problema *Distrituição de temperatura* pelo método SOR

Passo espacial: $hx = 0.004, hy = 0.002$; w=1.9	
Número de iterações	35878
Variação relativa máxima no SOR	$9.99946e-007$
Tempo de CPU (s)	$738\ s$
Passo espacial: $hx = 0.002, hy = 0.001$; w=1.9	
Número de iterações	100001
Variação relativa máxima no SOR	$1.28106e-006$
Tempo de CPU (s)	$8453\ s$

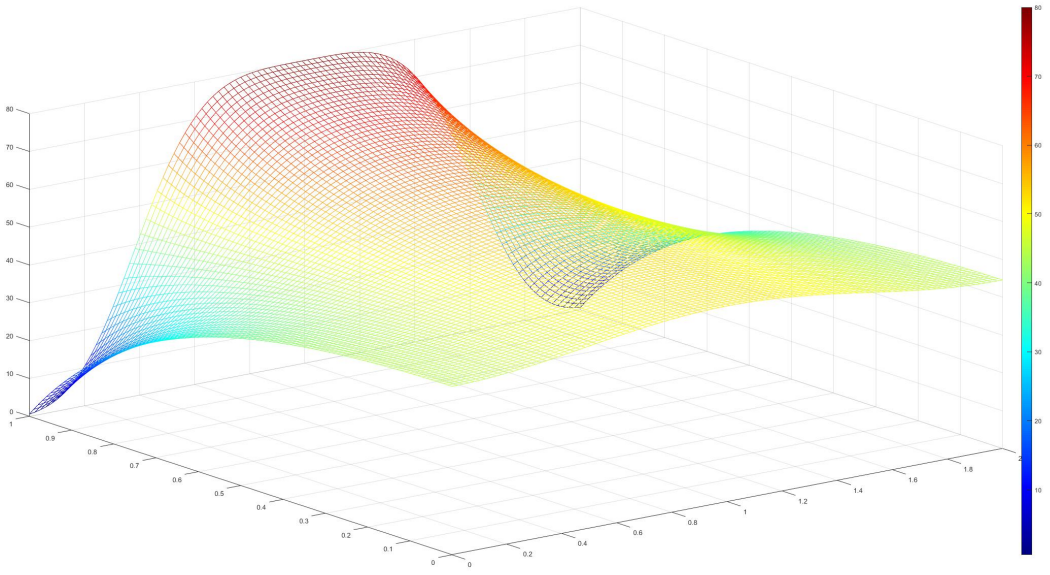
Fonte: O autor.

A Tabela 4.6 mostra os resultados das simulações em malhas uniformes 500x500 e 1000x1000 obtidas com o método SOR. A solução numérica em ambas as malhas é ilustrada nas Figuras 4.7 e 4.8. Podemos observar nessas figuras que a temperatura máxima ocorre na fronteira superior, onde se localiza a "fonte" de calor.

Figura 4.7 – Solução numérica do problema *Distribuição de temperatura* 2D no domínio $[0, 2]\times[0, 1]$, usando SOR, com $hx = 4 \cdot 10^{-3}$, $hy = 2 \cdot 10^{-3}$, $w = 1.9$ e 35878 iterações: (a) visualização bidimensional; (b) visualização tridimensional



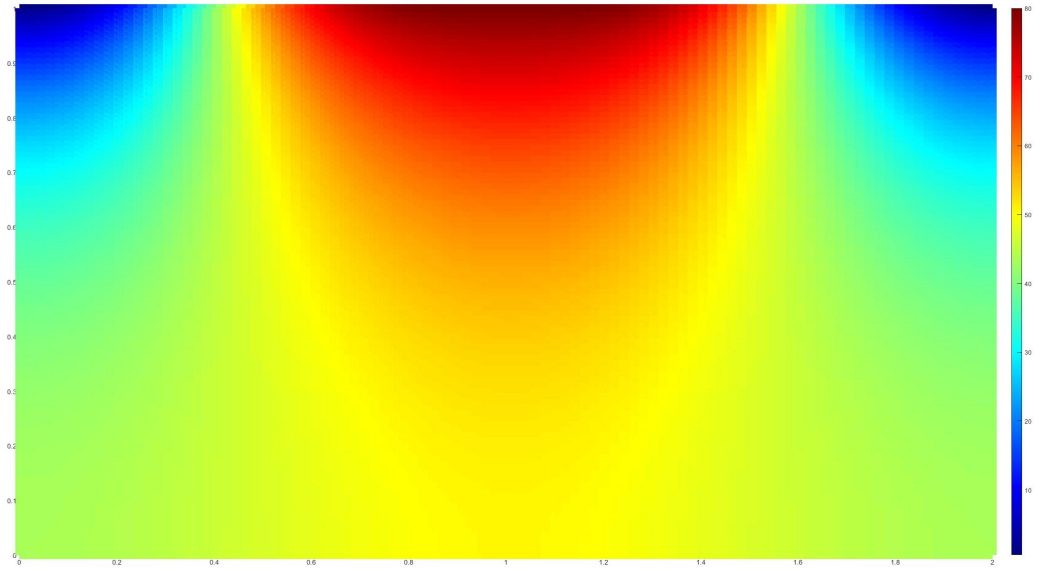
(a)



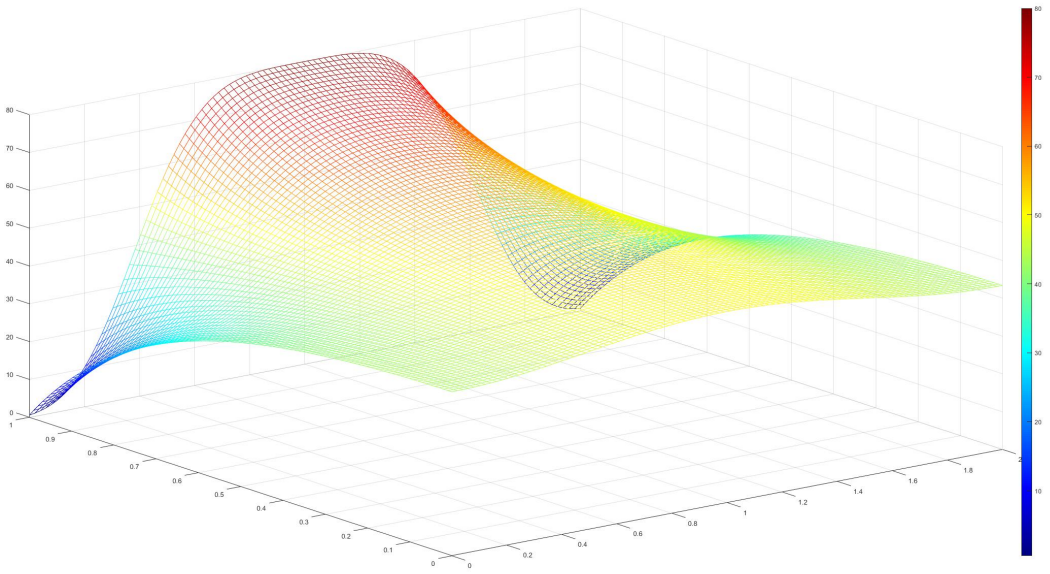
(b)

Fonte: O autor com o Matlab (2021).

Figura 4.8 – Solução numérica do problema *Distribuição de temperatura* 2D no domínio $[0, 2]\times[0, 1]$, usando SOR, com $hx = 2 \cdot 10^{-3}$, $hy = 10^{-3}$, $w = 1.9$ e 100001 iterações: (a) visualização bidimensional; (b) visualização tridimensional



(a)



(b)

Fonte: O autor com o Matlab (2021).

4.6 ANÁLISE DAS SIMULAÇÕES

Um ponto singular, ou singularidade, é um ponto no qual uma função não é definida ou deixa de ser "bem-comportada" devido, por exemplo, à falta de diferenciabilidade ou de analiticidade.

Ao simular numericamente modelos com singularidades, devemos avaliar aspectos tais como: método de discretização; tipo de malha; método de solução de sistemas lineares.

Analisando as simulações deste capítulo, concluímos que o método SOR, quando empregado para solucionar sistemas lineares provenientes da discretização com diferenças finitas de EDPs elípticas de segunda ordem em malhas estruturadas, não é eficiente na solução numérica de problemas com singularidades fortes, como evidencia a problema *Singularidade interior* 2D, com $\alpha = 1.1$ e $\beta = 0$.

5 CONCLUSÕES

Apresentamos neste trabalho soluções manufaturadas para a equação de Poisson, uma equação diferencial parcial elíptica de segunda ordem. Aproximamos essas soluções numericamente empregando o método de diferenças finitas em malhas estruturadas em duas e três dimensões. Nas aproximações, calculadas através de códigos computacionais construídos em Linguagem C, utilizamos os métodos iterativos de Gauss-Seidel e SOR para solucionar os sistemas lineares provenientes da discretização das EDPs. Com o auxílio dos softwares Matlab e Tecplot 360, visualizamos as soluções exata e numérica.

Baseados nos testes executados no Capítulo 3, concluímos que o método SOR exige, geralmente, um número consideravelmente menor de iterações do que o método de Gauss-Seidel para solucionar os sistemas lineares advindos do processo de discretização usando diferenças finitas. Mesmo assim, como mostram alguns testes do Capítulo 3 e as simulações do Capítulo 4, o método SOR tem convergência lenta quando aplicado a problemas com condições de contorno de Neumann e a problemas com singularidades. Estes problemas podem ser solucionados de forma mais eficiente com o emprego do método multigrid em malhas adaptativas com refinamento localizado (NÓS; ROMA; CENICEROS, 2005; NÓS, 2007; CENICEROS; NÓS; ROMA, 2010; NÓS; CENICEROS; ROMA, 2012; NÓS et al., 2017).

Os principais desafios/dificuldades enfrentados pelo autor na elaboração deste trabalho foram: compreensão/associação das estruturas da linguagem C; seleção de testes para o Capítulo 4; teste de valores ótimos para o parâmetro de relaxação w no método SOR; emprego do Matlab para construir figuras tridimensionais. Quanto ao Matlab, o autor não conseguiu empregá-lo para construir figuras 4D. Essas figuras foram geradas no Tecplot 360.

Este trabalho pode ser melhorado com a inclusão de testes manufaturados com condições de contorno periódicas e de Neumann, com a presença de singularidades fortes e não linearidades. O intuito é destacar sob quais condições o método SOR é eficiente na solução numérica de EDPs elípticas de segunda ordem discretizadas com diferenças finitas em malhas estruturadas 2D e 3D.

REFERÊNCIAS

ALMEIDA, A. P. de. **Geração computacional de malhas bidimensionais estruturadas em torno da asa de uma aeronave**. Monografia (Trabalho de Conclusão de Curso) — UTFPR, Campus Curitiba, 2017. 17, 20

BOYCE, W. E.; DIPRIMA, R. C. **Equações diferenciais elementares e problemas de contorno**. 9. ed. Rio de Janeiro: LTC, 2011. 13

BURDEN, R. L.; FAIRES, D. J.; BURDEN, A. M. **Análise numérica**. 3. ed. São Paulo: Cengage Learning, 2016. 21, 22, 33, 36, 91, 92

CENICEROS, H. D.; NÓS, R. L.; ROMA, A. M. Three-dimensional, fully adaptive simulations of phase-field fluid models. **Journal of Computational Physics**, v. 229, p. 6135–6155, 2010. 87

CHENG, L. **Finite difference methods for Poisson equation**. 2020. 38, 39

CUMINATO, J. A.; JUNIOR, M. M. **Discretização de equações diferenciais parciais**. 1. ed. Rio de Janeiro: SBM, 2013. 26, 28

FERNANDO, H. J. **Procedimentos numéricos para a solução das equações da advecção, da difusão e advecção-difusão pelo método das diferenças finitas**. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, 2004. 30

FERZIGER, J. H. **Numerical methods for engineering appliation**. New York: John Wiley & Sons, 1981. 21, 22

FIGUEIREDO, D. G. de. **Análise de Fourier e equações diferenciais parciais**. Rio de Janeiro: IMPA, 1997. 14

FORTUNA, A. de O. **Técnicas computacionais para dinâmica dos fluidos: conceitos básicos e aplicações**. São Paulo: Editora da Universidade de São Paulo, 2000. 15, 16, 17, 20, 22, 25, 26, 27, 31, 35, 36, 37

GARCIA, M. V. P.; HUMES, C.; STERN, J. M. Generalized line criterion for gauss-seidel method. **Computational and Applied Mathematics**, v. 22, n. 1, p. 91–97, 2003. 33

GERALD, C. F.; WHEATLEY, P. O. **Applied numerical analysis**. New York: Addison-Wesley Publishing Company, 1994. 21, 22

GPSGUI. **Diferenças finitas: um exemplo**. 2019. Disponível em: <https://tudosobcontrole.net/2017/08/21/diferencas-finitas-um-exemplo/>. Acesso em: 01 dez. 2019. 16, 73, 83

GUIDORIZZI, H. L. **Um curso de cálculo**. v. 1. Rio de Janeiro: LTC, 2018. 14

HILE, G. N. **Winplot**. 2012. Disponível em: <https://math.hawaii.edu/wordpress/winplot/>. Acesso em: 21 jul. 2021. 83

HUMES, A. F. P. C. et al. **Noções de cálculo numérico**. São Paulo: Mcgraw-Hill do Brasil, 1984. 21, 22, 31, 33, 91

IÓRIO, V. **EDP: um curso de graduação**. Rio de Janeiro: IMPA, 1989. 14, 22

JOHN, F. **Partial differential equations**. 4. ed. New York: Springer-Verlag, 1982. 22

KALABA, R. E.; SPINGARN, K. A criterion for the convergence of the gauss-seidel method. **Applied Mathematics and Computation**, v. 4, n. 4, p. 359–367, 1978. 33

LAPLACE, C. **Dev-C++ official website**. 2020. Disponível em: <https://www.bloodshed.net/>. Acesso em: 01 mar. 2020. 43

LOGAN, J. D. **Applied partial differential equations**. 3. ed. New York: Springer, 2015. 26

MALISKA, C. R. **Transferência de calor e mecânica dos fluídos computacional**. Rio de Janeiro: LTC, 2004. 17, 18

MATLAB. **MathWorks**. 2021. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 15 jun. 2021. 22, 43, 46, 49, 52, 55, 58, 61, 74, 76, 78, 80, 82, 84, 85

MITCHELL, W. F. A collection of 2d elliptic problems for testing adaptive grid refinement algorithms. **Preprint submitted to Elsevier**, p. 1–19, 2017. 73, 76

MU, S.-Y. **Dispersion and local-error analysis of compact LFE-27 formula for obtaining sixth-order accurate numerical solutions of 3D Helmholtz equation**. 2013. Disponível em: <https://www.researchgate.net/figure/Illustration-of-a-compact-7-point-stencil-layout-The-central-point-is-shown-in-black_fig1_275876371>. Acesso em: 07 jul. 2021. 40

NÓS, R. L. **Simulações de escoamentos tridimensionais bifásicos empregando métodos adaptativos e modelos de campo de fase**. Tese (Doutorado) — Instituto de Matemática e Estatística, Universidade de São Paulo, 2007. 18, 87

NÓS, R. L. **Séries de Fourier e aplicações**. Middletown: Kindle Direct Publishing, 2019. 14, 22

NÓS, R. L.; CENICEROS, H. D.; ROMA, A. M. Simulação tridimensional adaptativa da separação das fases de uma mistura bifásica usando a equação de cahn-hilliard. **TEMA Tendências em Matemática Aplicada e Computacional**, v. 13, p. 37–50, 2012. 87

NÓS, R. L.; ROMA, A. M.; CENICEROS, H. D. Solução de equações diferenciais parciais elípticas por técnicas multinível-multigrid em malhas tridimensionais bloco-estruturadas com refinamento localizado. In: **XXVIII Congresso Nacional de Matemática Aplicada e Computacional**. Santo Amaro: SBMAC, 2005. 43, 62, 87

NÓS, R. L. et al. Three-dimensional coarsening dynamics of a conserved, nematic liquid crystal-isotropic fluid mixture. **Journal of Non-Newtonian Fluid Mechanics**, v. 248, p. 62–73, 2017. 87

OVERLEAF. **Latex, evoluído**. 2019. Disponível em: <https://pt.overleaf.com/>. Acesso em: 29 mar. 2019. 22

POINTWISE. **How meshing is better with pointwise**. 2019. Disponível em: <https://www.pointwise.com/pointwise/>. Acesso em: 3 out. 2019. 19

RESEARCHGATE. **Malhas não estruturadas**. 2019. Disponível em: <https://www.researchgate.net/figure/Malhas-nao-estruturadas-geradas-levando-em-conta-o-criterio-de-ortogonalidade_fig24_312490054>. Acesso em: 24 out. 2019. 19

SCHILDT, H. **C - Completo e total**. 3. ed. São Paulo: Makron Books, 1997. 22

SCHWARZ, H. R. **Numerical analysis: a comprehensive introduction**. New York: John Wiley & Sons, 1989. 21, 22

STRIKWERDA, J. C. **Finite difference schemes and partial differential equations**. New York: Chapman & Hall, 1989. 22, 43, 62

TECPLOT. **Tecplot 360**. 2021. Disponível em: <https://www.tecplot.com/products/tecplot-360/>. Acesso em: 01 jul. 2021. 43, 64, 66, 68, 70

THOMPSON, J. F.; SONI, B. K.; WEATHERILL, N. P. **Handbook of grid generation**. Boca Raton: CRC Press LLC, 1999. 18, 19

VARÓN, L. A. B. **Estudo teórico da hipertermia induzida por radiofrequência em tecidos carregados com nanopartículas**. 2019. Disponível em: <https://www.researchgate.net/figure/Figura-4-Distribuicao-de-Temperatura-sem-Campo-Eletromagnetico-e-sem-Nanoparticulas-A_fig2_264897694>. Acesso em: 01 dez. 2019. 16

WILLIAMS, G. **Five-point-stencil**. 2011. Disponível em: <https://source.ggy.bris.ac.uk/wiki/File:Five-point-stencil.jpg>. Acesso em: 06 jul. 2021. 39

APÊNDICE A

NORMAS

Seja M^{mxn} o espaço vetorial das matrizes mxn reais ou complexas. Uma norma $||\cdot||$ é uma função que associa a cada matriz um número real não negativo e satisfaz as seguintes propriedades:

1. $||A|| = 0$ se e somente se A é uma matriz nula;
2. $||\kappa A|| = |\kappa|\,||A||$;
3. $||A + B|| \leq ||A|| + ||B||$ (desigualdade triangular),

com $A, B \in M^{mxn}$ e $\kappa \in \mathbb{R}$.

Uma norma $||\cdot|| : \mathbb{R}^n \to \mathbb{R}$, que associa a um vetor de $\mathbb{R}^n$ um número real, é denominada *norma vetorial*. Similarmente, uma função $||\cdot|| : \mathbb{R}^{n\times n} \to \mathbb{R}$ é uma *norma matricial*.

A norma de uma matriz é uma medida de quão grande são seus elementos; é uma medida para determinar o "tamanho" de uma matriz de tal forma que não é necessário saber quantas linhas e colunas a matriz tem (BURDEN; FAIRES; BURDEN, 2016). A norma de uma matriz associada a um sistema linear pode ser um indicativo da convergência do método numérico empregado para solucioná-lo (HUMES et al., 1984).

A NORMA-1

A norma-1 de uma matriz A é definida como sendo

$$||A||_1 = \max_{1\leq j\leq n} \left(\sum_{i=1}^{n} |a_{ij}| \right),$$

ou seja, é a maior soma dos módulos dos elementos das colunas de A.

Exemplo *A norma-1 da matriz*

$$A = \begin{bmatrix} 2 & -5 \\ -1 & -3 \end{bmatrix}$$

é $||A||_1 = 8$*, uma vez que a soma dos valores absolutos dos elementos da primeira e da segunda colunas de* A *é, respectivamente, igual a:* $2 + |-1| = 3$*;* $|-5| + |-3| = 8$.

A NORMA EUCLIDIANA

A norma Euclidiana de uma matriz A, também denominada norma-2, é definida como sendo

$$||A||_2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij})^2},$$

ou seja, é a raiz quadrada da soma dos quadrados dos elementos de A.

Exemplo *A norma Euclidiana da matriz*

$$A = \begin{bmatrix} 2 & -5 \\ -1 & -3 \end{bmatrix}$$

é $||A||_2 = \sqrt{39}$*, porque* $\sqrt{2^2 + (-5)^2 + (-1)^2 + (-3)^2} = \sqrt{39}$.

A NORMA DO MÁXIMO

A norma do máximo de uma matriz A, também denominada norma infinito, é definida como sendo

$$||A||_\infty = \text{máx}\{|a_{ij}|\},$$

ou seja, é o maior valor absoluto dos elementos de A (BURDEN; FAIRES; BURDEN, 2016).

Exemplo *A norma do máximo da matriz*

$$A = \begin{bmatrix} 2 & -5 \\ -1 & -3 \end{bmatrix}$$

é $||A||_\infty = 5$*, isto porque* $|-5| > |-3| > |2| > |-1|$.

Podemos empregar o conceito de norma de uma matriz para determinar a magnitude do erro cometido na solução numérica de uma EDP pelo método das diferenças finitas. Exemplificando, suponhamos que $A = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}$ e $B = \begin{bmatrix} 1.1 \\ 4.2 \\ 7 \end{bmatrix}$ representem, respectivamente, as soluções exata e numérica de uma EDP em um determinado conjunto de pontos. Ao calcularmos $B - A = \begin{bmatrix} 0.1 \\ 0.2 \\ 0 \end{bmatrix}$, temos que $||B - A||_\infty = 0.2$. Esta medida representa o maior erro absoluto cometido na aproximação e reflete o desempenho do método numérico na solução numérica da EDP.

APÊNDICE B

CÓDIGO PARA GERAÇÃO DE FIGURAS 3D PELO MATLAB

```
% Program to generate a graphic of a function by a txt file
datl = importdata('D:\Testes\2D\numeric_poisson2D.txt');
x = datl(:,1);
y = datl(:,2);
z = datl(:,3);
F = scatteredInterpolant(x, y, z);
xx = linspace(min(x), max(x));
yy = linspace(min(y), max(y));
[XX, YY] = meshgrid(xx, yy);
ZZ = F(XX, YY);
mesh(XX, YY, ZZ, 'facecolor', 'none');
colorbar;
% Program to generate a surface in matlab
[X,Y]= meshgrid(0:0.01:1,0:0.01:1);
 surf(X,Y,X.*Y);
 colorbar
% Program to generate a function defined by two sentences
 [X,Y]= meshgrid(-1:0.02:1,-1:0.02:1)
 z=cos(pi*Y/2);
 mask = X >0;
 z(mask) = z(mask)+X(mask).^1.1;
 surf(X,Y,z);
 colorbar
```

CÓDIGO C BIDIMENSIONAL

```
//Program to solve 2D Poisson's equation by finite difference method

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>

#define imax 1005
#define jmax 1005
#define tol pow(10,-4)
```

```
11 #define itmax pow(10,5) //Number of maximum iterations in
      Gauss-Seidel Method
12
13 //******************************************************
14 //Functions
15 //******************************************************
16
17 double ex(double x,double y){
18         double exact;
19         exact=cos(2*M_PI*x)*cos(2*M_PI*y);
20         return exact;
21 }
22
23 double f(double x,double y){
24         double value;
25         value=-8*pow(M_PI,2.)*cos(2*M_PI*x)*cos(2*M_PI*y);
26         return value;
27 }
28
29 void initialdata (void);
30 void dirichletboundary (void);
31 void neumannboundary1 (void);
32 void neumannboundary2 (void);
33 void initialconditions1 (void);
34 void initialconditions2 (void);
35 void ghostpoints (void);
36 void gaussseidel (void);
37 void sor (void);
38 void outputdata (void);
39
40 //******************************************************
41 //Global variables
42 //******************************************************
43
44 int i,j,it,m,n;
45 double hx,hy,sx,sy,erro=-1.0,erroaux;
46 double a[imax][jmax],aaux[imax][jmax];
47 //******************************************************
48 //Main
49 //******************************************************
50
51 int main(){
```

```
52
53          initialdata ();
54          //dirichletboundary ();
55          neumannboundary1 ();
56          //neumannboundary2 ();
57          initialconditions1 ();
58          //initialconditions2 ();
59          ghostpoints ();
60          //gaussseidel ();
61          sor ();
62          outputdata ();
63
64          printf("\nData stored\nPress any key to exit...");
65          getch ();
66          return 0;
67  }
68
69  //*******************************************************
70  //Initial data
71  //*******************************************************
72
73  void initialdata (void) {
74
75  //Length in x direction
76  sx=1;
77  //Number of steps in x direction
78  m=100;
79  //Length in y direction
80  sy=1;
81  //Number of steps in y direction
82  n=100;
83  //Spatial step
84  hx=sx/m;
85  hy=sy/n;
86  //number of mesh points is one more than number of steps
87  m=m+1;
88  n=n+1;
89  }
90
91  //*******************************************************
92  //Assigning boundary conditions
93  //*******************************************************
```

```
94
95  void dirichletboundary (void) {
96
97          for(i=1;i<=m;i++){
98                  a[i][1]=exp(sin((i-1)*hx)+1.0); //bottom
99                  a[i][n]=exp(sin((i-1)*hx)+cos(6.0)); //top
100         }
101         for(j=1;j<=n;j++){
102                 a[1][j]=exp(cos((j-1)*hy)); //left
103                 a[m][j]=exp(sin(10.0)+cos((j-1)*hy)); //right
104         }
105 }
106
107 //Exact first derivative
108
109 void neumannboundary1 (void) {
110
111         for(i=1;i<=m;i++){
112                 a[i][1]=0.0; //bottom
113                 a[i][n]=0.0; //top
114         }
115         for(j=1;j<=n;j++){
116                 a[1][j]=0.0; //left
117                 a[m][j]=0.0; //right
118         }
119 }
120
121 //Second-order discrete first derivative
122
123 void neumannboundary2 (void) {
124
125         for(i=1;i<=m;i++){
126                 a[i][1]=(-3*a[i][1]+4*a[i][2]-a[i][3])/(2*hy); //
                    bottom
127                 a[i][n]=(3*a[i][n]-4*a[i][n-1]+a[i][n-2])/(2*hy); //
                    top
128         }
129         for(j=1;j<=n;j++){
130                 a[1][j]=(-3*a[1][j]+4*a[2][j]-a[3][j])/(2*hx); //left
131                 a[m][j]=(3*a[m][j]-4*a[m-1][j]+a[m-2][j])/(2*hx); //
                    right
132         }
```

```
133 }
134
135 //*******************************************************
136 //Assigning initial condition
137 //*******************************************************
138
139 //Interior points
140
141 void initialconditions1 (void) {
142
143         for(i=2;i<m;i++) {
144                 for(j=2;j<n;j++) {
145                         a[i][j]=0.0;
146                         aaux[i][j]=a[i][j];
147                 }
148         }
149 }
150
151 //Entire domain
152
153 void initialconditions2 (void) {
154
155         for(i=1;i<=m;i++) {
156                 for(j=1;j<=n;j++) {
157                         a[i][j]=0.0;
158                         aaux[i][j]=a[i][j];
159                 }
160         }
161 }
162
163 //*******************************************************
164 //Assigning ghost points
165 //*******************************************************
166
167 void ghostpoints (void) {
168
169         for(i=1;i<=m;i++){
170                 a[i][0]=a[i][2]; //bottom
171                 a[i][n+1]=a[i][n-1]; //top
172         }
173         for(j=0;j<=(n+1);j++){
174                 a[0][j]=a[2][j]; //left
```

```
175                 a[m+1][j]=a[m-1][j]; //right
176         }
177 }
178
179 //*******************************************************
180 // Finite difference scheme - Iterative Gauss-Seidel
181 //*******************************************************
182
183 void gaussseidel (void) {
184
185         int it=0;
186         double errogs=1.0,erroaux2;
187         FILE *gs;
188         gs=fopen("iterations2D_GS.txt","w");
189
190         while ((errogs>=tol) && (it<=itmax)){
191                 errogs=0.0;
192                 for(i=1;i<=m;i++){
193                         for(j=1;j<=n;j++) {
194                         a[i][j]=(-pow(hx,2)*pow(hy,2)*f((i-1)*hx,(j
                            -1)*hy)+pow(hy,2)*(a[i-1][j]+a[i+1][j])+
                            pow(hx,2)*(a[i][j-1]+a[i][j+1]))/(2*(pow(
                            hx,2)+pow(hy,2)));
195                         if (a[i][j]==0)
196                                 erroaux2=1.0;
197                         else
198                                 erroaux2=fabs((a[i][j]-aaux[i][j])/a[
                                    i][j]);
199                         if (erroaux2 > errogs)
200                                         errogs = erroaux2;
201                         aaux[i][j]=a[i][j];
202                         }
203                 }
204                 ghostpoints ();
205                 it++;
206         }
207         fprintf(gs,"The relative variation in the Gauss-Seidel method
                is equal to: %.6g\n The number of iterations in the Gauss
               -Seidel method is equal to: %d\n",errogs,it);
208         fclose(gs);
209 }
210
```

```
211 //*****************************************************
212 // Finite difference scheme - Iterative SOR
213 //*****************************************************
214
215 void sor (void) {
216
217         int it=0;
218         double errosor=1.0,erroaux3,w=1.9;
219         FILE *gsor;
220         gsor=fopen("iterations2D_SOR.txt","w");
221
222         while ((errosor>=tol) && (it<=itmax)){
223                 errosor=0.0;
224                 for(i=1;i<=m;i++){
225                         for(j=1;j<=n;j++) {
226                         a[i][j]=(-pow(hx,2)*pow(hy,2)*f((i-1)*hx,(j
                            -1)*hy)+pow(hy,2)*(a[i-1][j]+a[i+1][j])+
                            pow(hx,2)*(a[i][j-1]+a[i][j+1]))/(2*(pow(
                            hx,2)+pow(hy,2)));
227                         a[i][j]=(1.0-w)*aaux[i][j]+w*a[i][j];
228                         if (a[i][j]==0)
229                                 erroaux3=1.0;
230                         else
231                                 erroaux3=fabs((a[i][j]-aaux[i][j])/a[
                                    i][j]);
232                         if (erroaux3 > errosor)
233                                         errosor = erroaux3;
234                         aaux[i][j]=a[i][j];
235                         }
236                 }
237                 ghostpoints ();
238                 it++;
239         }
240         fprintf(gsor,"The relative variation in the SOR method is
                equal to: %.6g\n The number of iterations in the SOR
                method is equal to: %d\n",errosor,it);
241         fclose(gsor);
242 }
243
244 //*****************************************************
245 //Output files - Maximum norm of error
246 //*****************************************************
```

```
247
248 void outputdata (void) {
249
250         FILE *fp;
251         fp=fopen("numeric_poisson2D.txt","w");
252         FILE *er;
253         er=fopen("maximumerror2D.txt","w");
254
255         for(i=1;i<=m;i++) {
256                 for(j=1;j<=n;j++){
257                         erroaux = fabs(a[i][j]-ex((i-1)*hx,(j-1)*hy))
                            ;
258                         if (erroaux>erro) {
259                                 erro = erroaux;
260                         }
261                 fprintf(fp,"%.6g %.6g %.6g\n",(i-1)*hx,(j-1)*hy,a[i][
                    j]);
262                 }
263         }
264         fprintf(er,"The maximum error made (infinite norm) is equal
               to: %.6g\n",erro);
265         fclose(er);
266         fclose(fp);
267 }
```

CÓDIGO C TRIDIMENSIONAL

```
1  //Program to solve 3D Poisson's equation by finite difference method
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <math.h>
6  #include <conio.h>
7
8  #define imax 105
9  #define jmax 105
10 #define kmax 105
11 #define tol pow(10,-4)
12 #define itmax pow(10,5) //Number of maximum iterations in
       Gauss-Seidel Method
13
14 //*****************************************************
```

```
15 //Functions
16 //*******************************************************
17
18 double ex(double x,double y,double z){
19         double exact;
20         exact=cos(2*M_PI*x)*cos(2*M_PI*y)*cos(2*M_PI*z);
21         return exact;
22 }
23
24 double f(double x,double y,double z){
25         double value;
26         value=-12*pow(M_PI,2.)*cos(2*M_PI*x)*cos(2*M_PI*y)*cos(2*M_PI
              *z);
27         return value;
28 }
29
30 void initialdata (void);
31 void dirichletboundary (void);
32 void neumannboundary1 (void);
33 void neumannboundary2 (void);
34 void initialconditions1 (void);
35 void initialconditions2 (void);
36 void ghostpoints (void);
37 void gaussseidel (void);
38 void sor (void);
39 void outputdata (void);
40
41 //*******************************************************
42 //Global variables
43 //*******************************************************
44
45 int i,j,k,it,m,n,l;
46 double hx,hy,hz,sx,sy,sz,erro=-1.0,erroaux;
47 double a[imax][jmax][kmax],aaux[imax][jmax][kmax];
48 //*******************************************************
49 //Main
50 //*******************************************************
51
52 int main(){
53
54         initialdata ();
55         //dirichletboundary ();
```

```
56          neumannboundary1 ();
57          //neumannboundary2 ();
58          initialconditions1 ();
59          //initialconditions2 ();
60          ghostpoints ();
61          //gaussseidel ();
62          sor ();
63          outputdata ();
64
65          printf("\nData stored\nPress any key to exit...");
66          getch ();
67          return 0;
68  }
69
70  //*******************************************************
71  //Initial data
72  //*******************************************************
73
74  void initialdata (void) {
75
76  //Length in x direction
77  sx=1.0;
78  //Number of steps in x direction
79  m=100;
80  //Length in y direction
81  sy=1.0;
82  //Number of steps in y direction
83  n=100;
84  //Length in z direction
85  sz=1.0;
86  //Number of steps in z direction
87  l=100;
88  //Spatial step
89  hx=sx/m;
90  hy=sy/n;
91  hz=sz/l;
92  //number of mesh points is one more than number of steps
93  m=m+1;
94  n=n+1;
95  l=l+1;
96  }
97
```

```
98  //*******************************************************
99  //Assigning boundary conditions
100 //*******************************************************
101
102 void dirichletboundary (void) {
103
104         for(i=1;i<=m;i++){
105                 for(j=1;j<=n;j++){
106                         a[i][j][1]=0.0; //bottom
107                         a[i][j][l]=((i-1)*hx)*((j-1)*hy); //top
108                 }
109         }
110         for(j=1;j<=n;j++){
111                 for(k=1;k<=l;k++){
112                         a[1][j][k]=0.0; //left
113                         a[m][j][k]=((j-1)*hy)*((k-1)*hz); //right
114                 }
115         }
116         for(i=1;i<=m;i++){
117                 for(k=1;k<=l;k++){
118                         a[i][1][k]=0.0; //front
119                         a[i][n][k]=((i-1)*hx)*((k-1)*hz); //back
120                 }
121         }
122 }
123
124 //Exact first derivative
125
126 void neumannboundary1 (void) {
127
128         for(i=1;i<=m;i++){
129                 for(j=1;j<=n;j++){
130                         a[i][j][1]=0.0; //bottom
131                         a[i][j][l]=0.0; //top
132                 }
133         }
134         for(j=1;j<=n;j++){
135                 for(k=1;k<=l;k++){
136                         a[1][j][k]=0.0; //left
137                         a[m][j][k]=0.0; //right
138                 }
139         }
```

```
140         for(i=1;i<=m;i++){
141                 for(k=1;k<=l;k++){
142                         a[i][1][k]=0.0; //front
143                         a[i][n][k]=0.0; //back
144                 }
145         }
146 }
147
148 //Second-order discrete first derivative
149
150 void neumannboundary2 (void) {
151
152         for(i=1;i<=m;i++){
153                 for(j=1;j<=n;j++){
154                         a[i][j][1]=(-3*a[i][j][1]+4*a[i][j][2]-a[i][j
                            ][3])/(2*hz); //bottom
155                         a[i][j][l]=(3*a[i][j][l]-4*a[i][j][l-1]+a[i][
                            j][l-2])/(2*hz); //top
156                 }
157         }
158         for(j=1;j<=n;j++){
159                 for(k=1;k<=l;k++){
160                         a[1][j][k]=(-3*a[1][j][k]+4*a[2][j][k]-a[3][j
                            ][k])/(2*hx); //left
161                         a[m][j][k]=(3*a[m][j][k]-4*a[m-1][j][k]+a[m
                            -2][j][k])/(2*hx); //right
162                 }
163         }
164         for(i=1;i<=m;i++){
165                 for(k=1;k<=l;k++){
166                         a[i][1][k]=(-3*a[i][1][k]+4*a[i][2][k]-a[i
                            ][3][k])/(2*hy); //front
167                         a[i][n][k]=(3*a[i][n][k]-4*a[i][n-1][k]+a[i][
                            n-2][k])/(2*hy); //back
168                 }
169         }
170 }
171
172 //*******************************************************
173 //Assigning initial condition
174 //*******************************************************
175
```

```
176 //Interior points
177
178 void initialconditions1 (void) {
179
180         for(i=2;i<m;i++) {
181                 for(j=2;j<n;j++) {
182                         for(k=2;k<l;k++) {
183                                 a[i][j][k]=0.0;
184                                 aaux[i][j][k]=a[i][j][k];
185                         }
186                 }
187         }
188 }
189
190 //Entire domain
191
192 void initialconditions2 (void) {
193
194         for(i=1;i<=m;i++) {
195                 for(j=1;j<=n;j++) {
196                         for(k=1;k<=l;k++) {
197                                 a[i][j][k]=0.0;
198                                 aaux[i][j][k]=a[i][j][k];
199                         }
200                 }
201         }
202 }
203
204 //*******************************************************
205 //Assigning ghost points
206 //*******************************************************
207
208 void ghostpoints (void) {
209
210         for(i=1;i<=m;i++){
211                 for(j=1;j<=n;j++){
212                         a[i][j][0]=a[i][j][2]; //bottom
213                         a[i][j][l+1]=a[i][j][l-1]; //top
214                 }
215         }
216         for(j=1;j<=n;j++){
217                 for(k=1;k<=l;k++){
```

```
218                         a[0][j][k]=a[2][j][k]; //left
219                         a[m+1][j][k]=a[m-1][j][k]; //right
220                 }
221         }
222         for(i=0;i<=(m+1);i++){
223                 for(k=0;k<=(l+1);k++){
224                         a[i][0][k]=a[i][2][k]; //front
225                         a[i][n+1][k]=a[i][n-1][k]; //back
226                 }
227         }
228 }
229
230
231 //*******************************************************
232 // Finite difference scheme - Iterative Gauss-Seidel
233 //*******************************************************
234
235 void gaussseidel (void) {
236
237         int it=0;
238         double errogs=1.0,erroaux2;
239         FILE *gs;
240         gs=fopen("iterations3D_GS.txt","w");
241
242         while ((errogs>=tol) && (it<=itmax)){
243                 errogs=0.0;
244                 for(k=1;k<=l;k++) {
245                         for(i=1;i<=m;i++){
246                                 for(j=1;j<=n;j++) {
247                                 a[i][j][k]=(-pow(hx,2)*pow(hy,2)*pow(
                                    hz,2)*f((i-1)*hx,(j-1)*hy,(k-1)*hz
                                    )+
248                                         pow(hy,2)*pow(hz,2)*(a[i
                                            -1][j][k]+a[i+1][j][k])
                                            +pow(hx,2)*pow(hz,2)*(a
                                            [i][j-1][k]+a[i][j+1][k
                                            ])+
249                                         pow(hx,2)*pow(hy,2)*(a[i][
                                            j][k-1]+a[i][j][k+1]))
                                            /(2*(pow(hy,2)*pow(hz
                                            ,2)+pow(hx,2)*pow(hz,2)
                                            +pow(hx,2)*pow(hy,2)));
```

```
250                                     if (a[i][j][k]==0)
251                                             erroaux2=1.0;
252                                     else
253                                             erroaux2=fabs((a[i][j][k]-
                                                aaux[i][j][k])/a[i][j][k])
                                                ;
254                                     if (erroaux2 > errogs)
255                                                     errogs = erroaux2;
256                             aaux[i][j][k]=a[i][j][k];
257                                     }
258                             }
259                     }
260                     ghostpoints ();
261                     it++;
262             }
263             fprintf(gs,"The relative variation in the Gauss-Seidel method
                    is equal to: %.6g\n The number of iterations in the Gauss
                   -Seidel method is equal to: %d\n",errogs,it);
264             fclose(gs);
265     }
266
267     //*******************************************************
268     // Finite difference scheme - Iterative SOR
269     //*******************************************************
270
271     void sor (void) {
272
273             int it=0;
274             double errosor=1.0,erroaux3,w=1.9;
275             FILE *gsor;
276             gsor=fopen("iterations3D_SOR.txt","w");
277
278             while ((errosor>=tol) && (it<=itmax)){
279                     errosor=0.0;
280                     for(k=1;k<=l;k++) {
281                             for(i=1;i<=m;i++){
282                                     for(j=1;j<=n;j++) {
283                                     a[i][j][k]=(-pow(hx,2)*pow(hy,2)*pow(
                                        hz,2)*f((i-1)*hx,(j-1)*hy,(k-1)*hz
                                        )+
284                                             pow(hy,2)*pow(hz,2)*(a[i
                                                -1][j][k]+a[i+1][j][k])
```

```
                                                    +pow(hx,2)*pow(hz,2)*(a
                                                    [i][j-1][k]+a[i][j+1][k
                                                    ])+
285                                                 pow(hx,2)*pow(hy,2)*(a[i][
                                                    j][k-1]+a[i][j][k+1]))
                                                    /(2*(pow(hy,2)*pow(hz
                                                    ,2)+pow(hx,2)*pow(hz,2)
                                                    +pow(hx,2)*pow(hy,2)));
286                                 a[i][j][k]=(1.0-w)*aaux[i][j][k]+w*a[
                                       i][j][k];
287                                 if (a[i][j][k]==0)
288                                         erroaux3=1.0;
289                                 else
290                                         erroaux3=fabs((a[i][j][k]-
                                            aaux[i][j][k])/a[i][j][k])
                                            ;
291                                 if (erroaux3 > errosor)
292                                               errosor = erroaux3;
293                                 aaux[i][j][k]=a[i][j][k];
294                                 }
295                         }
296                 }
297                 ghostpoints ();
298                 it++;
299         }
300         fprintf(gsor,"The relative variation in the SOR method is
               equal to: %.6g\n The number of iterations in the SOR
               method is equal to: %d\n",errosor,it);
301         fclose(gsor);
302 }
303
304 //*******************************************************
305 //Output files - Maximum norm of error
306 //*******************************************************
307
308 void outputdata (void) {
309
310         FILE *fp;
311         fp=fopen("numeric_poisson3D.dat","w");
312         fprintf(fp,"%s %s %s %s %s\n","VARIABLES=","X","Y","Z","a");
313         FILE *er;
314         er=fopen("maximumerror3D.txt","w");
```

```
315
316         for(k=1;k<=l;k++) {
317                 for(i=1;i<=m;i++) {
318                         for(j=1;j<=n;j++){
319                                 erroaux = fabs(a[i][j][k]-ex((i-1)*hx
                                        ,(j-1)*hy,(k-1)*hz));
320                                 if (erroaux>erro) {
321                                         erro = erroaux;
322                                 }
323                         fprintf(fp,"%.6g %.6g %.6g %.6g\n",(i-1)*hx,(
                                j-1)*hy,(k-1)*hz,a[i][j][k]);
324                 }
325                 }
326         }
327         fprintf(er,"The maximum error made (infinite norm) is equal
                to: %.6g\n",erro);
328         fclose(er);
329         fclose(fp);
330 }
```

ÍNDICE