

Relatório Exercício-Programa 1
MAP - 3121
Métodos Numéricos e Aplicações

Lui Damianci Ferreira - 10770579
Victor A. C. Athanasio - 9784401

1 de Maio de 2020

Resumo

Este relatório, existe como parte integrante da nossa solução para o Exercício Programa 1 proposto pela matéria MAP-3121, Métodos Numéricos e Aplicações.

Nele está documentado não somente o nosso entendimento do problema proposto como também nossa proposta para solução do mesmo e análise sobre os resultados provenientes da aplicação de métodos numéricos.

Tanto o script em **Python** quanto todas as imagens geradas encontram-se em anexo no arquivo **.zip** no qual está contido este relatório.

Conteúdo

1 O problema	5
2 Métodos de solução	5
2.1 Euler	5
2.1.1 Equação	6
2.1.2 Erro	6
2.2 Métodos implícitos	6
2.3 Euler implícito	6
2.3.1 Matriz	6
2.3.2 Erro	7
2.4 Crank-Nicolson	7
2.4.1 Matriz	7
2.4.2 Erro	7
3 Implementação	7
3.1 Matrizes	7
3.2 Decomposição de Cholesky	8
3.3 Acelerações	9
3.3.1 Bibliotecas	9
3.3.2 Stencil operations	9
3.3.3 Técnicas antigas	9
3.3.4 Técnicas novas	10
4 Erro	10
5 Problema A antigo	10
5.1 Analise de resultados	11
5.1.1 Euler	11
5.1.2 Euler implícito	12
5.1.3 Crank-Nicolson	12
5.2 Analise de erro	13
5.2.1 Euler	13
5.2.2 Euler implícito	17
5.2.3 Crank-Nicolson	19
5.3 Conclusões finais e analises	20
5.3.1 Euler	20
5.3.2 Euler implícito	22
5.3.3 Crank-Nicolson	23
6 Problema A	25
6.1 Analise de resultados	25
6.1.1 Euler	25
6.1.2 Euler implícito	26
6.1.3 Crank-Nicolson	27
6.2 Analise de erro	28
6.2.1 Euler	28
6.2.2 Euler implícito	32
6.2.3 Crank-Nicolson	34
6.3 Conclusões finais e analises	35
6.3.1 Euler	35

6.3.2	Euler implícito	37
6.3.3	Crank-Nicolson	38
7	Problema B	40
7.1	Analise de resultados	40
7.1.1	Euler	40
7.1.2	Euler implícito	41
7.1.3	Crank-Nicolson	42
7.2	Analise de erro	43
7.2.1	Euler	43
7.2.2	Euler implícito	47
7.2.3	Crank-Nicolson	49
7.3	Conclusões finais e analises	50
7.3.1	Euler	50
7.3.2	Euler implícito	52
7.3.3	Crank-Nicolson	53
8	Problema C	54
8.1	Analise de resultados	55
8.1.1	Euler	55
8.1.2	Euler implícito	56
8.1.3	Crank-Nicolson	57
8.2	Analise de erro	58
8.3	Conclusões finais e analises	58
8.3.1	Euler	59
8.3.2	Euler implícito	59
8.3.3	Crank-Nicolson	60
9	Conclusões finais	61
10	Apêndice	61
10.1	Gráficos não utilizados	61
10.2	Código fonte	61
10.3	Dados consolidados	61

1 O problema

O problema a ser resolvido é tido como um problema direto. Esses problemas são em geral bem-postos. A noção de problema bem-posto foi introduzida por Jacques Hadamard (1865-1963). Um problema é bem-posto se ele satisfaz estas três condições:

- Existência: existe pelo menos uma solução.
- Unicidade: se existir uma solução, ela é única
- Estabilidade: a solução deve depender continuamente dos dados

O problema a ser resolvido, é, dada as condições iniciais de uma barra, as condições de fronteira, e uma fonte de calor, qual será o estado final da barra?

Pela equação de calor ser uma equação diferencial parcial (EDP), este é um problema não só direto como também bem posto, e será o foco das nossas atenções.

2 Métodos de solução

Nosso principal objetivo ao longo desse exercício programa é dada as equações de 1 a 4, quais são todos os estados da barra entre os instantes 0 e T.

$$u_t(t, x) = u_{xx}(t, x) + f(t, x) \text{ em } [0, T] \times [0, 1] \quad (1)$$

$$u(0, x) = u_0(x) \text{ em } [0, 1] \quad (2)$$

$$u(t, 0) = g1(t) \text{ em } [0, T] \quad (3)$$

$$u(t, 1) = g2(t) \text{ em } [0, T] \quad (4)$$

Aqui, t é a variável temporal e x a variável espacial. Estamos usando uma notação compacta para as derivadas parciais. Por exemplo:

$$u_{xx}(t, x) = \frac{\partial^2 u(t, x)}{\partial^2 x} \quad (5)$$

Normalizamos o comprimento da barra para 1 e vamos integrar a equação num intervalo de tempo de 0 a T. A variável u(t, x) descreve a temperatura no instante t na posição x, sendo a distribuição inicial u0(x) dada. Na descrição acima as condições de fronteira eqs. (3) e (4) são do tipo Dirichlet, com as temperaturas nos extremos da barra prescritas. Alternativamente poderia ser prescrito o fluxo de calor nos extremos, com as derivadas de u dadas. A função f descreve as fontes de calor ao longo do tempo.

2.1 Euler

No método de Euler, para acharmos uma aproximação numérica para a solução deste problema, vamos basicamente trocar as derivadas parciais por suas expansões de Taylor e desta forma aproxima-las.

2.1.1 Equação

Feito o passo a anterior, chegamos a seguinte equação:

$$u_i^{k+1} = u_i^k + \Delta t \left(\frac{u_{i-1}^k - 2.u_i^k + u_{i+1}^k}{\Delta x^2} + f(t_k, x_i) \right) \quad (6)$$

Com $i = 1, \dots, N-1$, e $k = 0, \dots, M-1$.

Vale notar que o estado de cada ponto e cada instante, depende única e exclusivamente da fonte de calor e de 3 pontos no instante anterior (o imediatamente atrás dele e os vizinhos diretos deste).

2.1.2 Erro

Ao analisarmos o erro de truncamento proveniente da eq. (6), podemos ver que o mesmo converge para zero, se e somente se, $\lambda \leq 0.5$.

$$\lambda = \frac{\Delta t}{\Delta x^2}$$

Deste modo, podemos dizer que o método de Euler é condicionalmente convergente de ordem 2 em Δx . Isso significa que refinamentos na malha em X, nos forçam a ter o mesmo refinamento ao quadrado em T, para garantir-se a convergência do método numérico.

2.2 Métodos implícitos

Para a obtenção de soluções refinadas, o método de Euler mostra-se imprático, uma vez que que a complexidade do algoritmo é cúbica em N, e rapidamente o recurso computacional necessário torna-se demasiadamente grande.

A solução para este problema é a obtenção de métodos implícitos, onde os pontos não dependem somente de vizinhos no instante anterior, mas dependem também dos vizinhos no próprio instante.

2.3 Euler implícito

Nessa abordagem, o valor de cada ponto passa a ser uma combinação linear entre os pontos vizinhos no instante, o ponto no instante anterior e a fonte de calor. Desta forma, o estado da barra em cada instante pode ser determinado resolvendo-se um sistema linear de equações.

2.3.1 Matriz

Tal sistema linear de equações, pode ser descrito na forma matricial da seguinte forma:

$$\begin{bmatrix} 1 + 2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1 + 2\lambda & -\lambda & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\lambda & 1 + 2\lambda & -\lambda \\ 0 & \cdots & 0 & -\lambda & 1 + 2\lambda \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \begin{bmatrix} u_1^k + \Delta t f_1^{k+1} + \lambda g_1(t^{k+1}) \\ u_2^k + \Delta t f_2^{k+1} \\ \vdots \\ u_{N-2}^k + \Delta t f_{N-2}^{k+1} \\ u_{N-1}^k + \Delta t f_{N-1}^{k+1} + \lambda g_2(t^{k+1}) \end{bmatrix}$$

2.3.2 Erro

Analisando a convergência desse método, podemos ver que ele é de ordem 2 em Δx e ordem 1 em Δt . Embora tal método seja incondicionalmente convergente, a precisão da solução ainda dependerá do refinamento em Δt

2.4 Crank-Nicolson

Para superar o empecilho de convergência de ordem 1 em Δt , surge o método de Crank-Nicolson, onde cada ponto depende tanto dos vizinhos no instante atual, quanto dos vizinhos no instante anterior e na fonte de calor. De forma similar ao método de Euler implícito o estado da barra em cada instante pode ser determinado resolvendo-se um sistema linear de equações.

2.4.1 Matriz

Tal sistema linear de equações, pode ser descrito na forma matricial da seguinte forma:

$$\begin{bmatrix} 1 + \lambda & -\frac{\lambda}{2} & 0 & \cdots & 0 \\ -\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} \\ 0 & \cdots & 0 & -\frac{\lambda}{2} & 1 + \lambda \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \begin{bmatrix} (1 - \lambda)u_1^k + \frac{\Delta t}{2}(f_1^k + f_1^{k+1}) + \frac{\lambda}{2}(u_0^k + u_2^k + g_1(t^{k+1})) \\ (1 - \lambda)u_2^k + \frac{\Delta t}{2}(f_2^k + f_2^{k+1}) + \frac{\lambda}{2}(u_1^k + u_3^k) \\ \vdots \\ (1 - \lambda)u_{N-2}^k + \frac{\Delta t}{2}(f_{N-2}^k + f_{N-2}^{k+1}) + \frac{\lambda}{2}(u_{N-3}^k + u_{N-1}^k) \\ (1 - \lambda)u_{N-1}^k + \frac{\Delta t}{2}(f_{N-1}^k + f_{N-1}^{k+1}) + \frac{\lambda}{2}(u_{N-2}^k + u_N^k + g_2(t^{k+1})) \end{bmatrix}$$

Lembrando que u_0^k e u_N^k são definidos pelas equações de contorno, e portanto devem ser mensurados através delas.

2.4.2 Erro

Novamente analisando a convergência desse método, podemos ver que ele é de ordem 2 tanto em Δx quanto em Δt . Diferentemente do método Euler implícito, a qualidade dos resultados da análise numérica não são gargaladas pelo refinamento em Δt

3 Implementação

A implementação das soluções foi feita em python, onde uma classe chamada *heat_solver* (heat equation solver) foi definida. Tal classe é dotada dos métodos e algoritmos necessários não só para a solução dos 3 métodos, mas também para a geração de diversos gráficos e de criação de logs com informações relevantes de cada iteração para possíveis análises posteriores.

3.1 Matrizes

Para a implementação de todos os métodos, todos os estados da barras foram armazenados em matrizes, onde o eixo y representa a barra e o eixo x o passar do tempo. Por exemplo, acessar o ponto (0.5,0.7) corresponde a ver a temperatura na posição 0.7 da barra no instante 0.5.

3.2 Decomposição de Cholesky

Para os métodos implícitos, a cada passo, é necessário resolver um sistema linear de equações em que somente o lado direito muda. Para execução destas soluções, foi desenvolvido um algoritmo baseado na decomposição de Cholesky, mesmo algoritmo utilizado por processadores de álgebra linear como LAPACK e FORTRAN, devido a sua grande eficiência computacional.

Dada uma matrix da maneira descrita em 7, onde A e B são constantes.

$$\begin{bmatrix} A & B & 0 & \cdots & 0 \\ B & A & B & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & B & A & B \\ 0 & \cdots & 0 & B & A \end{bmatrix} \quad (7)$$

Uma decomposição do tipo LDL^t , representada a seguir,

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ L_0 & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & L_{N-4} & 1 & 0 \\ 0 & \cdots & 0 & L_{N-3} & 1 \end{bmatrix} \begin{bmatrix} D_0 & 0 & 0 & \cdots & 0 \\ 0 & D_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & D_{N-3} & 0 \\ 0 & \cdots & 0 & 0 & D_{N-2} \end{bmatrix} \begin{bmatrix} 1 & L_0 & 0 & \cdots & 0 \\ 0 & 1 & L_1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & 1 & L_{N-3} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

pode ser facilmente encontrada por meio da relação de recorrência eq. (8):

$$\begin{cases} D_0 = A. \\ L_N = B/D_N. \quad N = 0, \dots, N-3 \\ D_N = A - (L_{N-1})^2 * D_{N-1}. \quad N = 1, \dots, N-2 \end{cases} \quad (8)$$

A partir de uma decomposição LDT^t a solução de um sistema linear passa a ter 3 etapas e ser trivial. As três etapas podem ser descritas como *forward solution*, *diagonal solution* e *backward solution*.

A *forward solution* pode ser feita a partir substituição eq. (9).

$$LDL^t * X = L * Y = U \iff Y = DL^t * X \quad (9)$$

A partir desse ponto, a solução se da pela relação de recorrência eq. (10).

$$\begin{cases} Y_0 = U_0. \\ Y_N = U_N - Y_{N-1} * L_{N-1}. \quad N = 1, \dots, N-2 \end{cases} \quad (10)$$

Para a *diagonal solution*, pode se fazer a substituição eq. (11).

$$DL^t * X = D * W = Y \iff W = L^t * X \quad (11)$$

Continuamos a solução por outra razão de recorrência, (12).

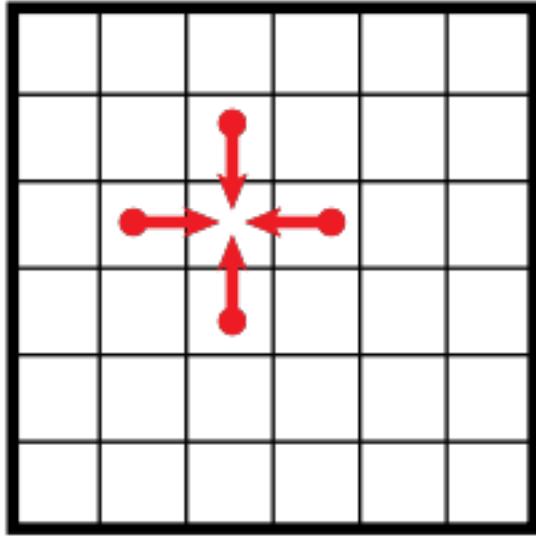
$$\begin{cases} W_N = \frac{Y_N}{D_N}. \quad N = 0, \dots, N-2 \end{cases} \quad (12)$$

Agora, para a *backward solution*, devemos usar a relação de recorrência (13).

$$\begin{cases} X_{N-2} = W_{N-2}. \\ X_N = W_N - X_{N+1} * L_{N+1}. \quad N = N-3, \dots, 0 \end{cases} \quad (13)$$

Figura 1: Representação de uma operação do tipo Stencil

Figura 2: Exemplo de núcleo de uma operação stencil



3.3 Acelerações

Devido ao grande poder computacional, e necessidade de memória RAM, demandado pelas soluções do método não implícito, diversas otimizações foram feitas ao código, para que o mesmo tornasse-se o mais rápido possível. Embora sem grande necessidade, tais otimizações também foram feitas para os métodos implícitos. As otimizações mais significativas vieram por meio do manejo de alocação de memória para os vetores, de forma a reduzir ao máximo a necessidade de cópia das informações e alocações de memória desnecessárias.

3.3.1 Bibliotecas

Uma biblioteca utilizada nesta etapa de otimização foi a Numba. Ela é uma biblioteca capaz de compilar em linguagem de máquina um trecho de código de python, e criar um CPU Dispatcher para executar as operações com velocidades similares a C++. Ela também é capaz de executar operações diretamente na GPU, porém para este processo ser vantajoso, você precisa estar lidando com muitas operações, uma vez que a comunicação entre a CPU e a GPU adiciona um tempo significativo para o processo.

Outra biblioteca utilizada foi a plotly, pela capacidade de produção de gráficos interativos, alguns exemplos encontram-se na pasta 'Gráficos_interativos' no mesmo .zip que esse relatório se encontra.

3.3.2 Stencil operations

Stencil operations são operações definidas localmente em uma matriz (figura: 1, são muito comuns para processamento de imagem, e extremamente eficientes, computacionalmente falando).

Esse tipo de operação foi utilizado para a solução do método de Euler, em que cada ponto depende de uma combinação linear de pontos vizinhos no instante anterior. Esse método também se aplicou para a criação dos lados direitos dos sistemas do método de Crank-Nicolson, pelos mesmos motivos.

3.3.3 Técnicas antigas

Entre as técnicas utilizadas na primeira versão do EP estão:

- Criação de um array vazio para armazenar os estados da barra
- Criação de um array contendo todos os valores de F
- Iteração sobre arrays do numpy
- etc ...

3.3.4 Técnicas novas

Entre as medidas tomadas para melhoria do desempenho, as principais foram:

- Criação de um único array que guardava os valores de F e os estados da barra.

Em vez de criar 2 arrays, apenas um array foi criado, iniciado com os valores de F, e esses valores foram tratados de forma a serem o mais próximo do necessário para os cálculos definidos pelos métodos. Por exemplo, multiplicando por $\frac{\lambda}{2}$ ou por Δ , isso depende do método que estava sendo executado.

E assim que tais informações foram utilizadas, e não serão mais acessadas, elas são apagadas, pelo próprio estado da barra, que passa a ser armazenado no array.

Os principais ganhos de performance ao fazer isso, é uma diminuição em 50% da memória alocada, e diminuição significativa no tempo de execução, que não precisa mais alocar um array de grandes dimensões na memória.

- Criação de operadores do tipo stencil.
- Utilização do numba para compilar processos-chave que eram chamados muitas vezes.
- etc ...

4 Erro

Para calcular o erro, optamos por subtrair a matriz calculada pelo método, da matriz construída pela equação da solução exata, ponto a ponto, em todos os pontos. E tomando o erro como a média dos valores absolutos dos pontos.

5 Problema A antigo

As equações que descrevem esse problema são:

$$g_1(t) = 0$$

$$g_2(t) = 0$$

$$f(t,x) = 10x^2(x - 1) - 60xt + 20t$$

$$u_0(x) = 0$$

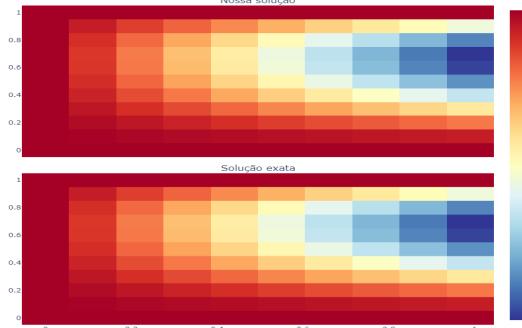
$$u(t,x) = 10tx^2(x - 1)$$

5.1 Analise de resultados

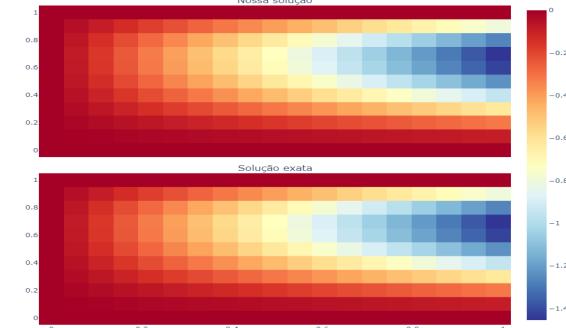
5.1.1 Euler

Podemos ver, na figura 3 uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções para ambos λ .

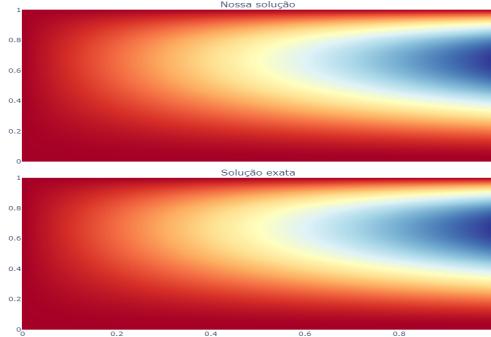
Evol. da temp na barra em função de t, lambda = 0.5, N = 10
Matrix shape: (11, 201); Execution time: 0.012 secs;
mean_error = 5.969e-17; Método: Euler; simulation = a_old



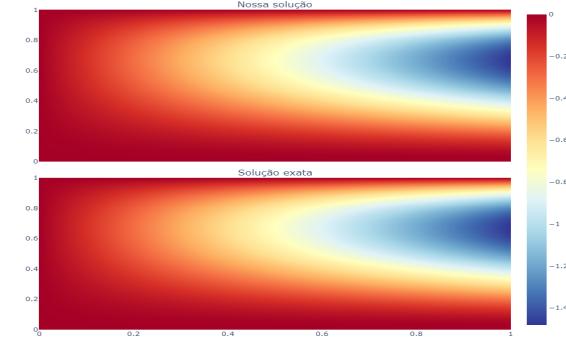
Evol. da temp na barra em função de t, lambda = 0.25, N = 10
Matrix shape: (11, 401); Execution time: 0.022 secs;
mean_error = 1.007e-16; Método: Euler; simulation = a_old



Evol. da temp na barra em função de t, lambda = 0.5, N = 320
Matrix shape: (321, 204801); Execution time: 8.38 secs;
mean_error = 2.362e-15; Método: Euler; simulation = a_old



Evol. da temp na barra em função de t, lambda = 0.25, N = 320
Matrix shape: (321, 409601); Execution time: 16.59 secs;
mean_error = 4.699e-15; Método: Euler; simulation = a_old



Evol. da temp na barra em função de t, lambda = 0.5, N = 320

Método: Euler, simulation = a_old

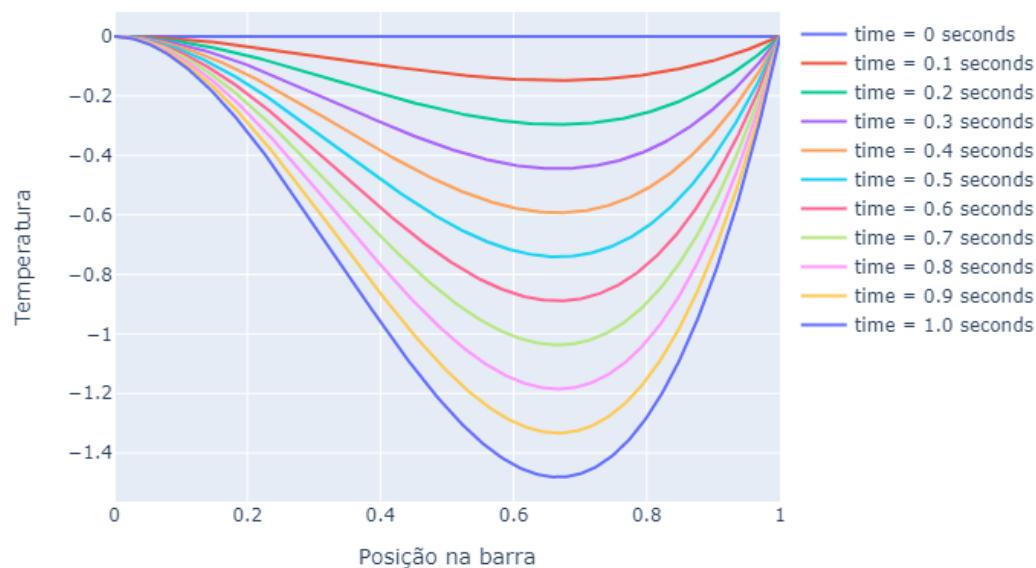
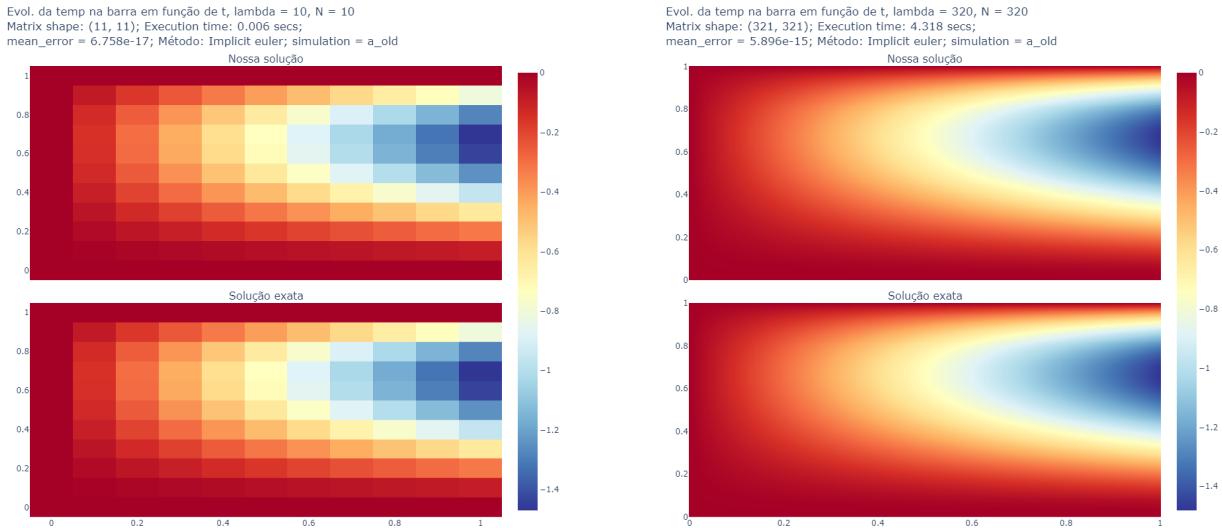


Figura 3: Podemos ver a evolução da temperatura na barra em diversos cenários, usando o método de Euler, com $\lambda= 0.5$ a esquerda, e $\lambda= 0.25$ a direita.

5.1.2 Euler implícito

Podemos ver, na figura 4, uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções, vale notar, que embora tenhamos mantido a mesma precisão nesse cenário, o tempo de execução e o custo computacional para tal solução, foi extremamente reduzido, se comparado com o método de Euler. Vale também ressaltar a ausência da dependência em λ



Evol. da temp na barra em função de t, lambda = 320, N = 320
Método: Implicit euler, simulation = a_old

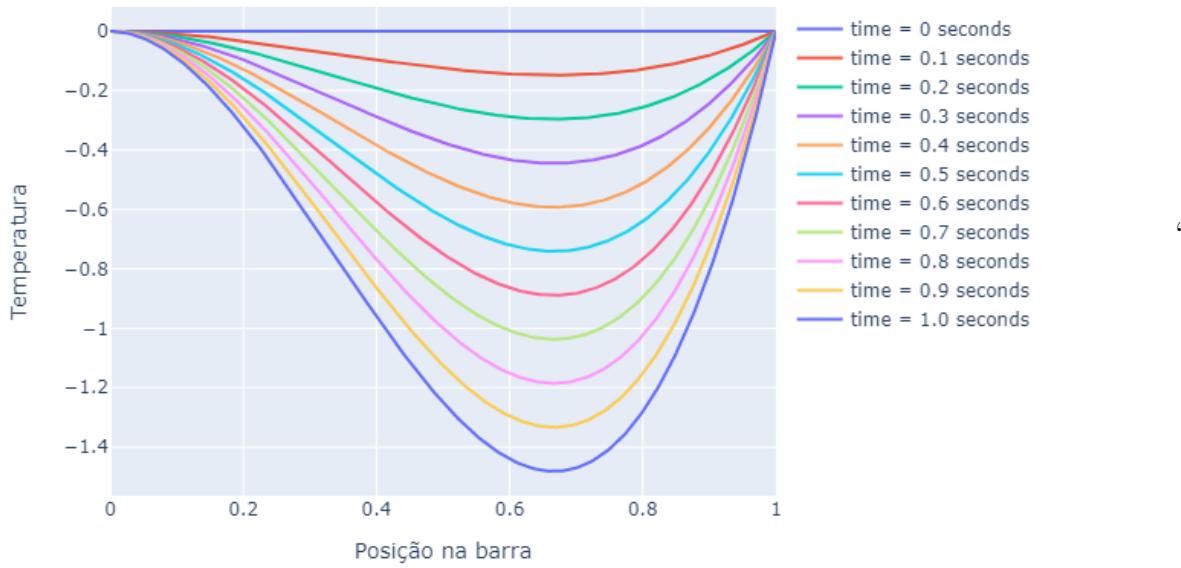
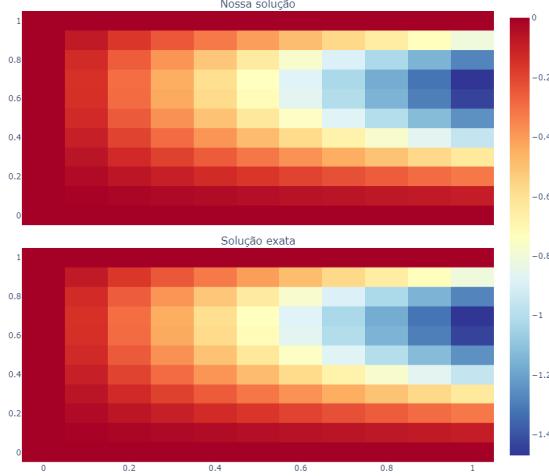


Figura 4: Resultados pelo método de Euler implícito

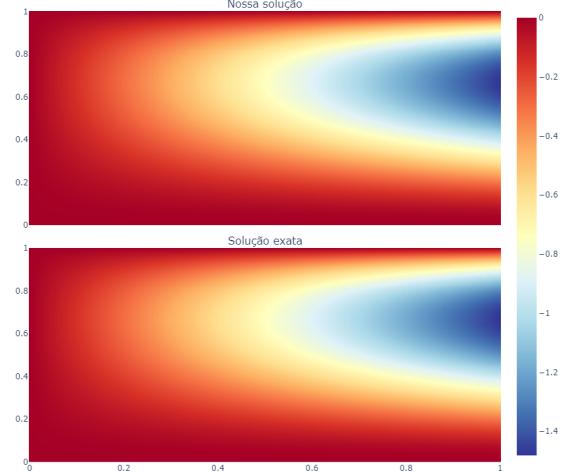
5.1.3 Crank-Nicolson

Podemos ver, na figura 5 uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções, o desempenho deste método neste cenário, mostrou-se muito semelhante ao do método implícito de Euler.

Evol. da temp na barra em função de t, lambda = 10, N = 10
Matrix shape: (11, 11); Execution time: 0.008 secs;
mean_error = 1.651e-16; Método: Crank-Nicolson; simulation = a_old



Evol. da temp na barra em função de t, lambda = 320, N = 320
Matrix shape: (321, 321); Execution time: 4.38 secs;
mean_error = 2.914e-14; Método: Crank-Nicolson; simulation = a_old



Evol. da temp na barra em função de t, lambda = 320, N = 320
Método: Crank-Nicolson, simulation = a_old

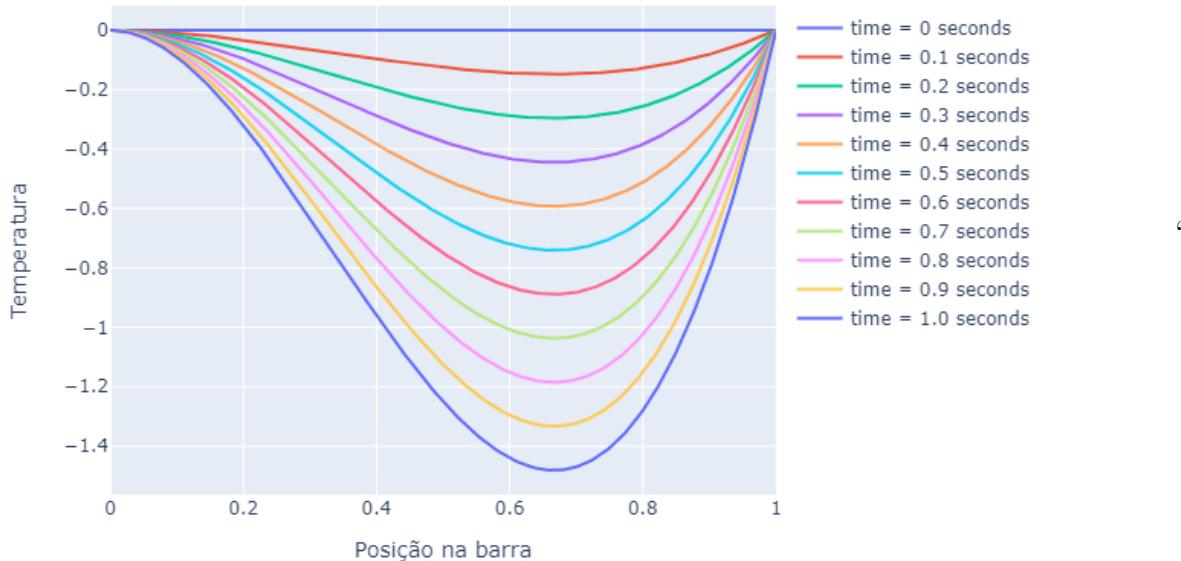


Figura 5: Resultados pelo método de Crank-Nicolson

5.2 Analise de erro

Para este problema, veremos uma tendência curiosa ocorrer, conforme aumentarmos o refinamento da malha, maior será o nosso erro, em todos os métodos. Isso ocorre, pois estes métodos são derivados a partir de séries de Taylor, que para polinômios, fornecem a solução exata. Os erros que vemos acontecer neste cenário são provenientes das limitações das implementações de números para computadores. No nosso caso, mais precisamente, estamos usando números do tipo *float64*. Isso também gera a falta de padrão nas distribuições do erro.

5.2.1 Euler

Podemos ver nas figuras, 6, 7 e 8, como o erro se desenvolve nas simulações em diversos refinamentos de malha. Vale notar que nestes cenários, a influência do parâmetro λ , é mínima, e

que não existe um padrão para a distribuição deste erro.

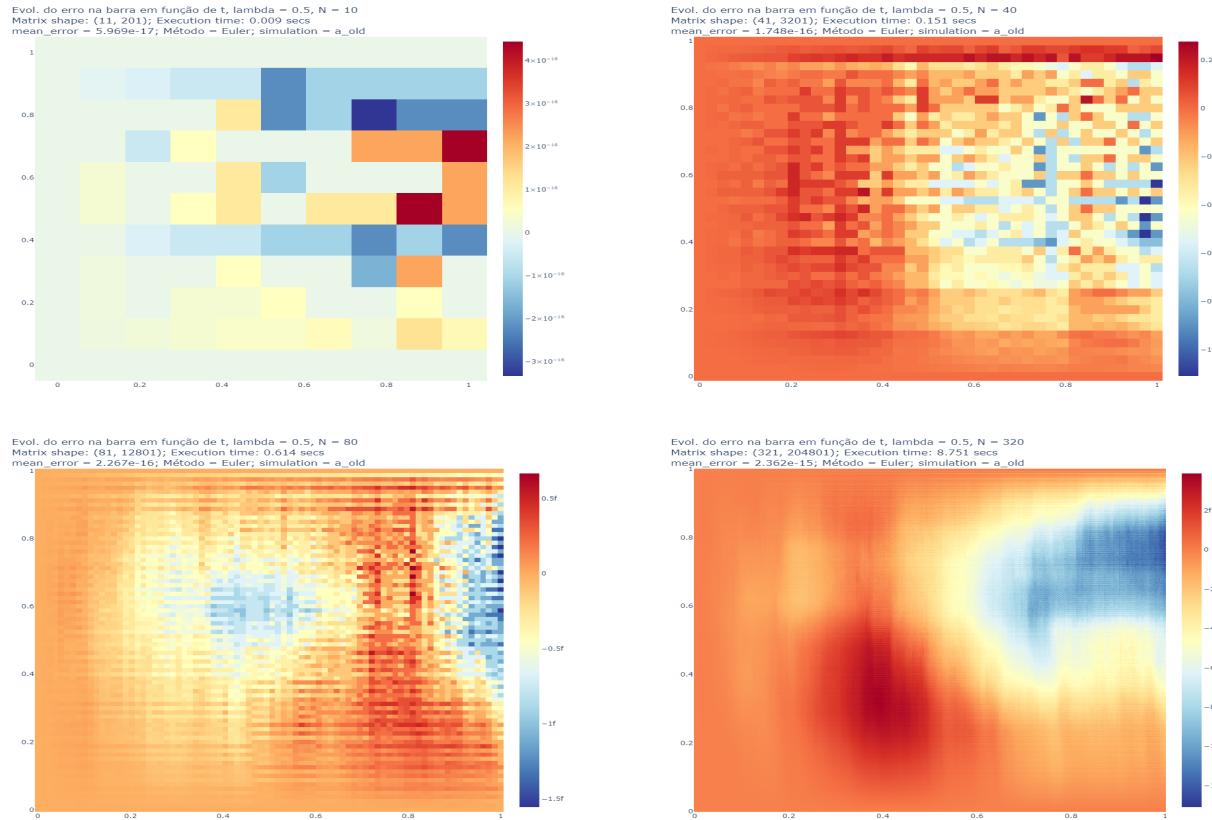


Figura 6: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.5$, método de Euler

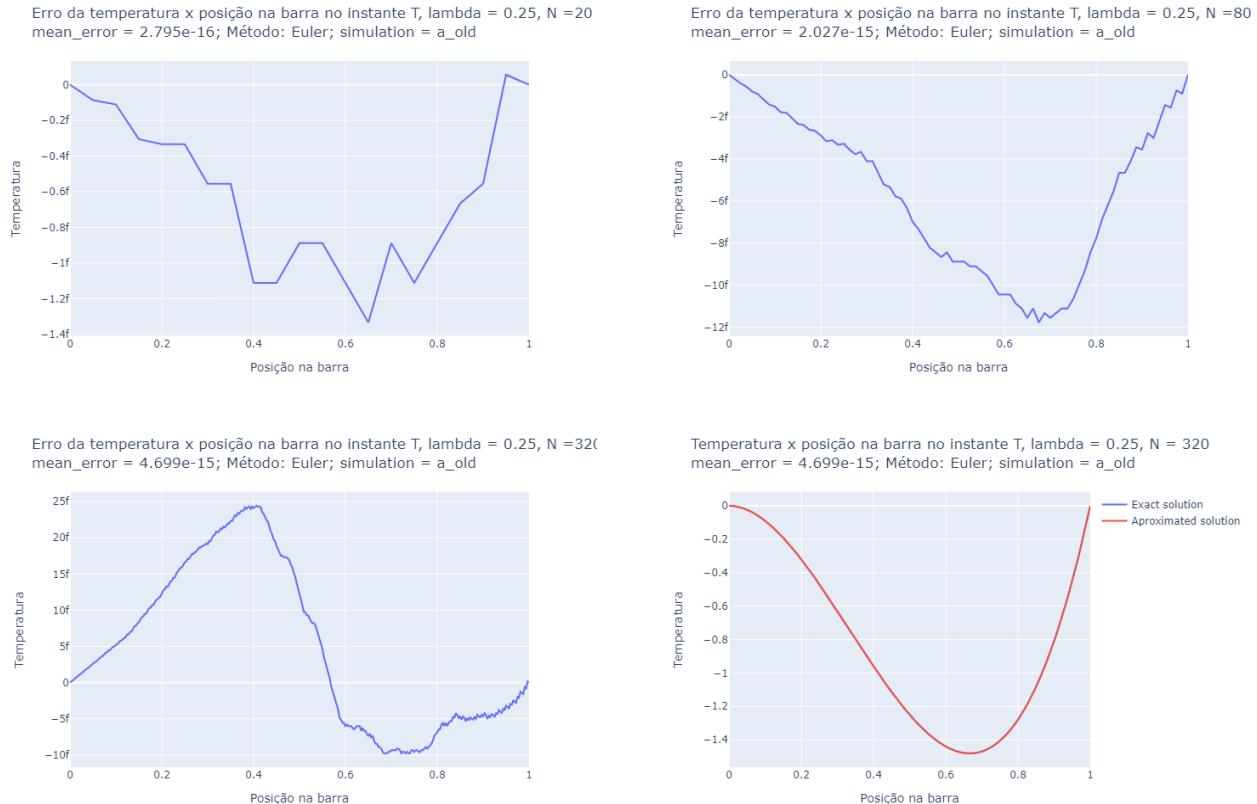
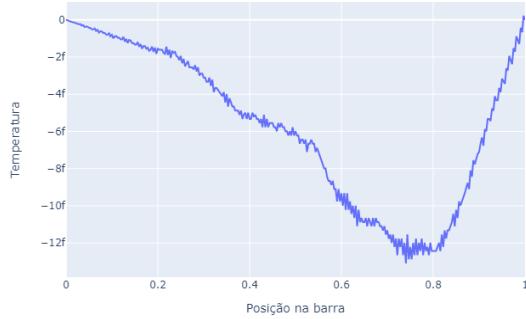


Figura 7: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.25$, método de Euler

Erro da temperatura x posição na barra no instante T, $\lambda = 0.5$, $N = 20$
 mean_error = 6.003e-17; Método: Euler; simulation = a_old



Erro da temperatura x posição na barra no instante T, $\lambda = 0.5$, $N = 320$
 mean_error = 2.362e-15; Método: Euler; simulation = a_old



Erro da temperatura x posição na barra no instante T, $\lambda = 0.5$, $N = 80$
 mean_error = 2.267e-16; Método: Euler; simulation = a_old



Temperatura x posição na barra no instante T, $\lambda = 0.5$, $N = 320$
 mean_error = 2.362e-15; Método: Euler; simulation = a_old

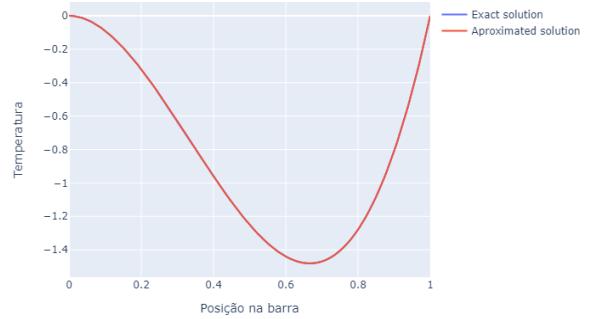


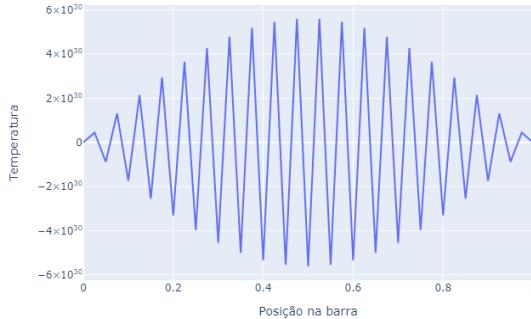
Figura 8: Distribuição do erro, no instante final, para diversos refinamentos de malha, com $\lambda = 0.5$, método de Euler

Nas figuras, 9, 10 e 11, podemos começar a ver as limitações deste método, onde para $\lambda > 0.50$, ele não converge para a solução. Para N pequenos, a solução não se distancia de forma significativa da solução correta, uma vez que não existem passos suficientes para que as instabilidades cresçam.

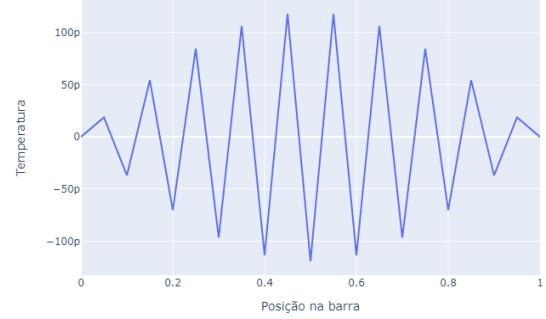
Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 10
mean_error = 4.107e-17; Método: Euler; simulation = a_old



Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 40
mean_error = 9.419e+28; Método: Euler; simulation = a_old



Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 20
mean_error = 5.520e-12; Método: Euler; simulation = a_old



Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 80
mean_error = 1.018e+187; Método: Euler; simulation = a_old

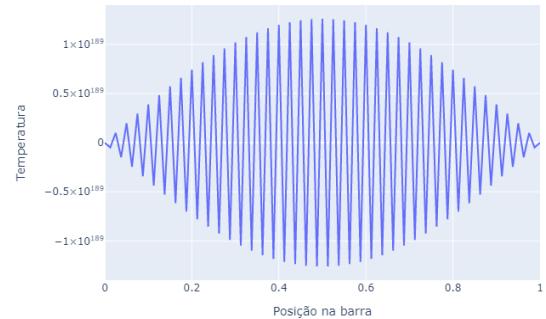
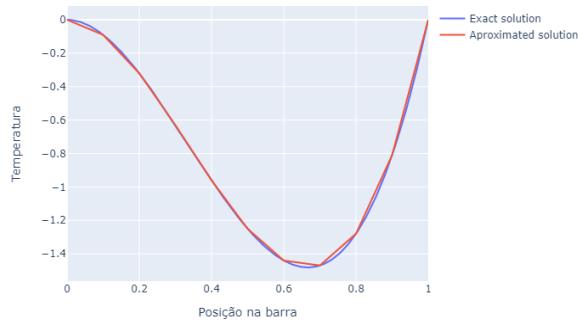
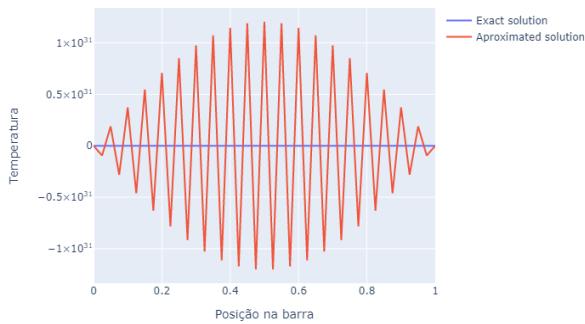


Figura 9: Distribuição do erro, no instante final, para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

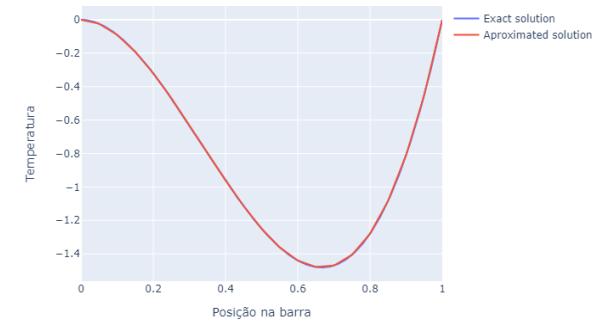
Temperatura x posição na barra no instante T, lambda = 0.51, N = 10
mean_error = 4.107e-17; Método: Euler; simulation = a_old



Temperatura x posição na barra no instante T, lambda = 0.51, N = 40
mean_error = 9.419e+28; Método: Euler; simulation = a_old



Temperatura x posição na barra no instante T, lambda = 0.51, N = 20
mean_error = 5.520e-12; Método: Euler; simulation = a_old



Temperatura x posição na barra no instante T, lambda = 0.51, N = 80
mean_error = 1.018e+187; Método: Euler; simulation = a_old

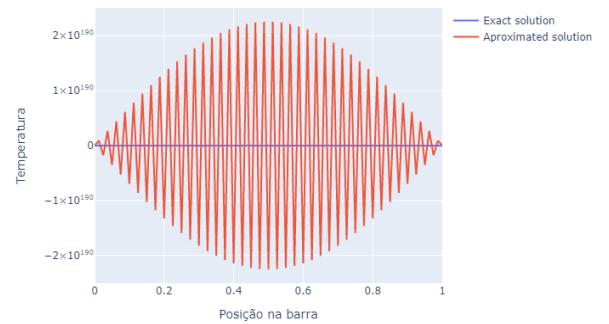


Figura 10: Estado final para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

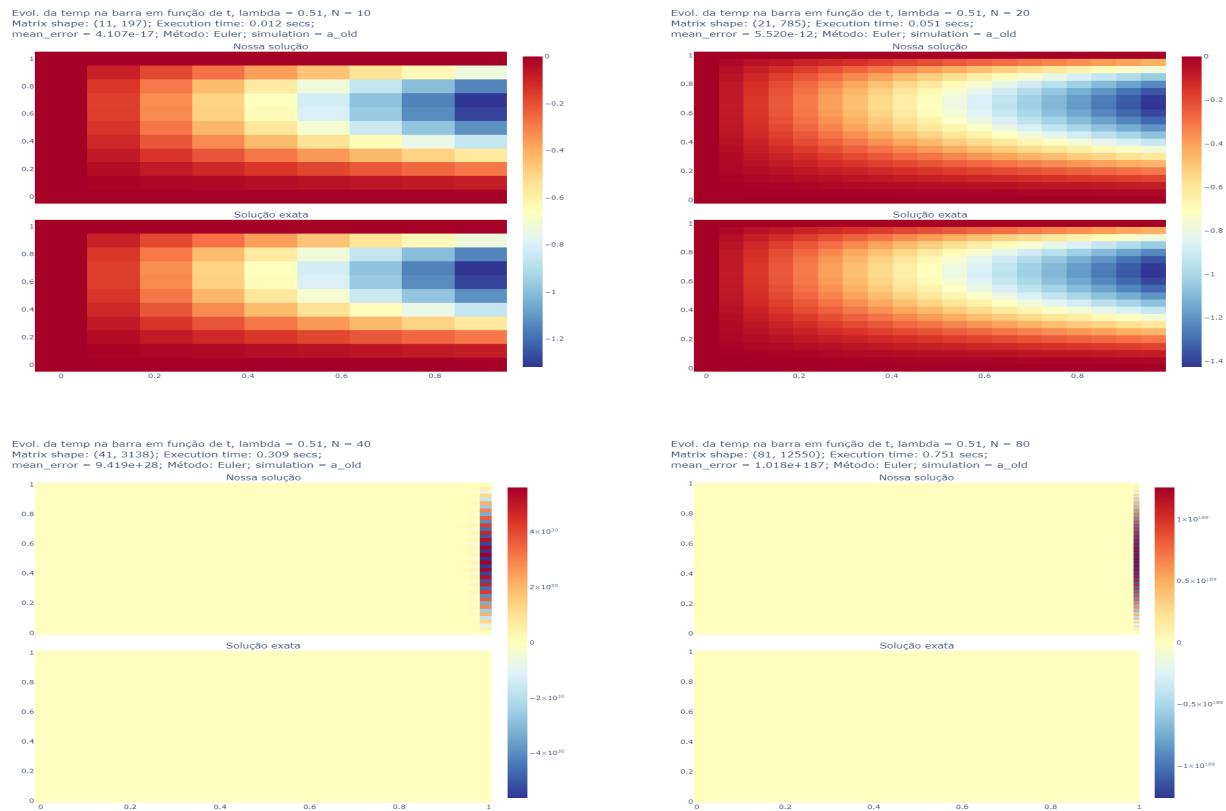


Figura 11: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

5.2.2 Euler implícito

Podemos ver nas figuras, 12 e 13 como o erro se desenvolve nas simulações em diversos refinamentos de malha. Vale notar que nestes cenários, não existe um padrão para a distribuição desse erro.

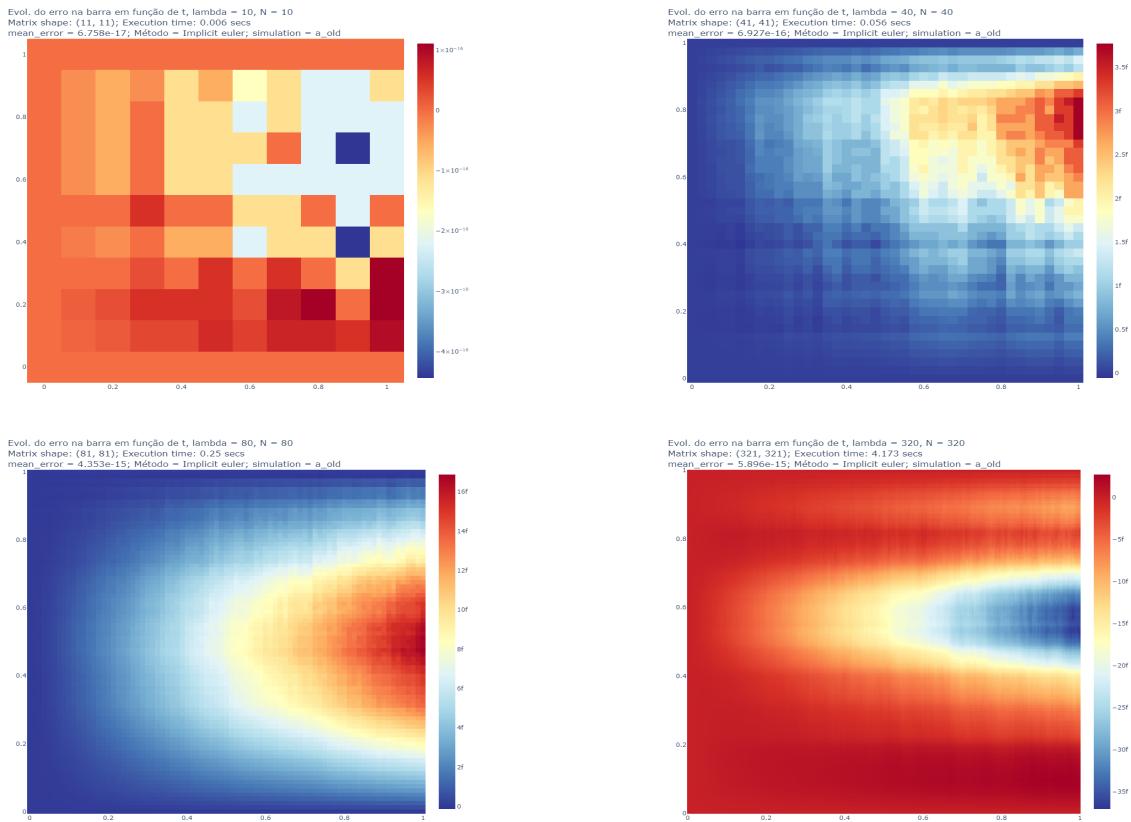


Figura 12: Distribuição do erro para diversos refinamentos de malha, método de Euler implícito

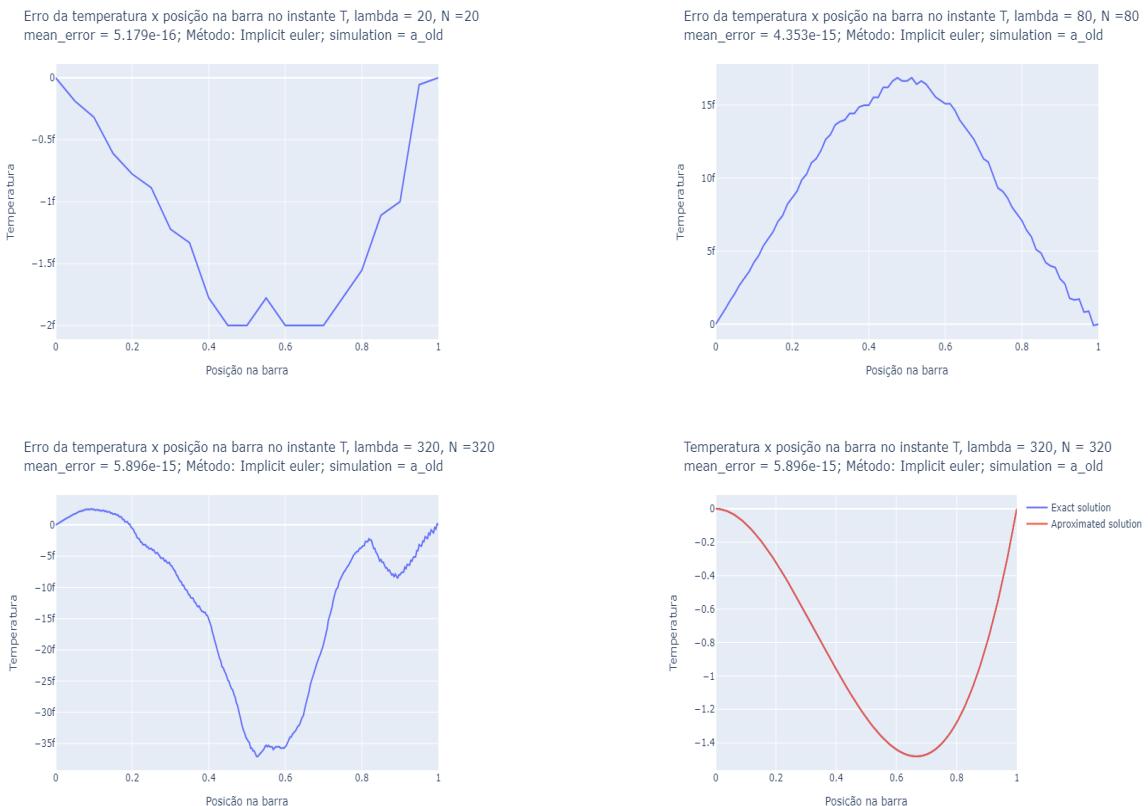


Figura 13: Distribuição do erro, no instante final, para diversos refinamentos de malha, método de Euler implícito

5.2.3 Crank-Nicolson

Podemos ver nas figuras, 14 e 15 como o erro se desenvolve nas simulações em diversos refinamentos de malha. Vale notar que nestes cenários, não existe um padrão para a distribuição desse erro.

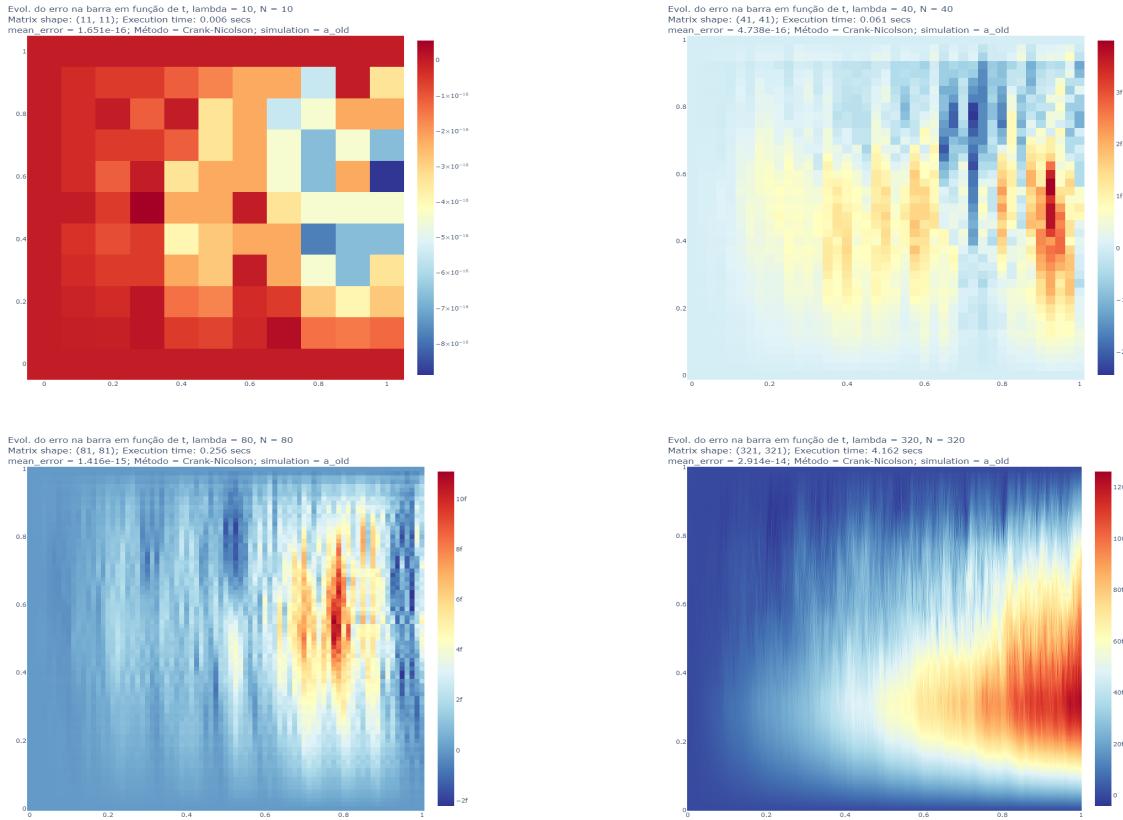
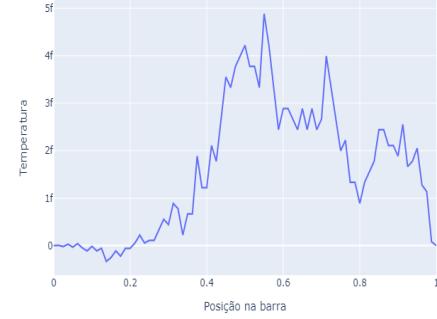


Figura 14: Distribuição do erro para diversos refinamentos de malha, método de Crank-Nicolson

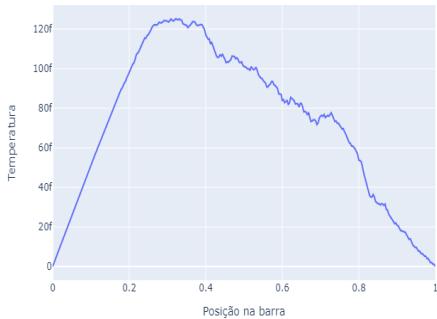
Erro da temperatura x posição na barra no instante T, lambda = 20, N = 20
mean_error = 1.918e-16; Método: Crank-Nicolson; simulation = a_old



Erro da temperatura x posição na barra no instante T, lambda = 80, N = 80
mean_error = 1.416e-15; Método: Crank-Nicolson; simulation = a_old



Erro da temperatura x posição na barra no instante T, lambda = 320, N = 320
mean_error = 2.914e-14; Método: Crank-Nicolson; simulation = a_old



Temperatura x posição na barra no instante T, lambda = 320, N = 320
mean_error = 2.914e-14; Método: Crank-Nicolson; simulation = a_old

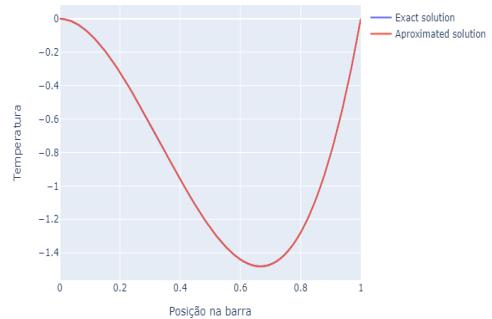


Figura 15: Distribuição do erro, no instante final, para diversos refinamentos de malha, método de Crank-Nicolson

5.3 Conclusões finais e análises

Neste cenário em específico, não se pode chegar a nenhuma conclusão muito significativa sobre os erros para os diversos métodos, principalmente pelo fato, de a maior fonte dos erros não depender nos métodos, e sim dos *float64*. Entretanto, pode-se notar as limitações do método de Euler quanto aos refinamentos no Δt .

5.3.1 Euler

Podemos ver, na figura 16, que esse método possui uma complexidade de $O(N^3)$, fazendo o número de pontos a serem determinados a crescer de forma muito rápida, e por conseguinte, o tempo e os recursos necessários.

Number of calculations (matrix size) in function of N for each lambda
Método: Euler; simulation = a_old

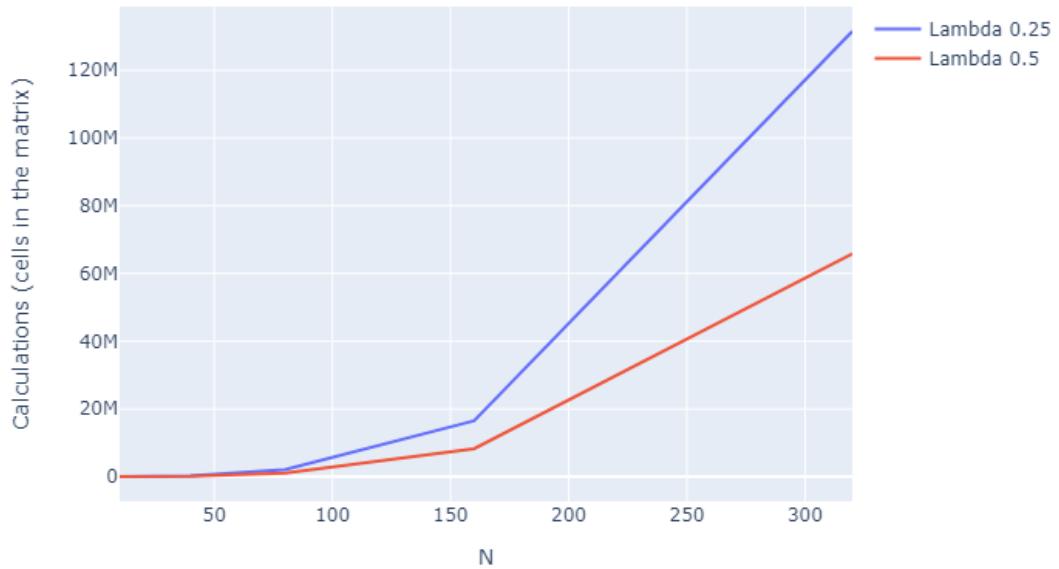


Figura 16: Número de cálculos em função de N

Analizando o erro, nas figuras, 17 e 18, podemos notar também, que o mesmo aumenta com o refinamento da malha, e aumenta com a diminuição de λ . Novamente, esse comportamento deve-se ao *float64*, e em ambas condições descritas acima, o número de cálculos aumenta, e portanto a propagação dos erros de aproximação também.

Error in function of lambda for each N
Método: Euler; simulation = a_old

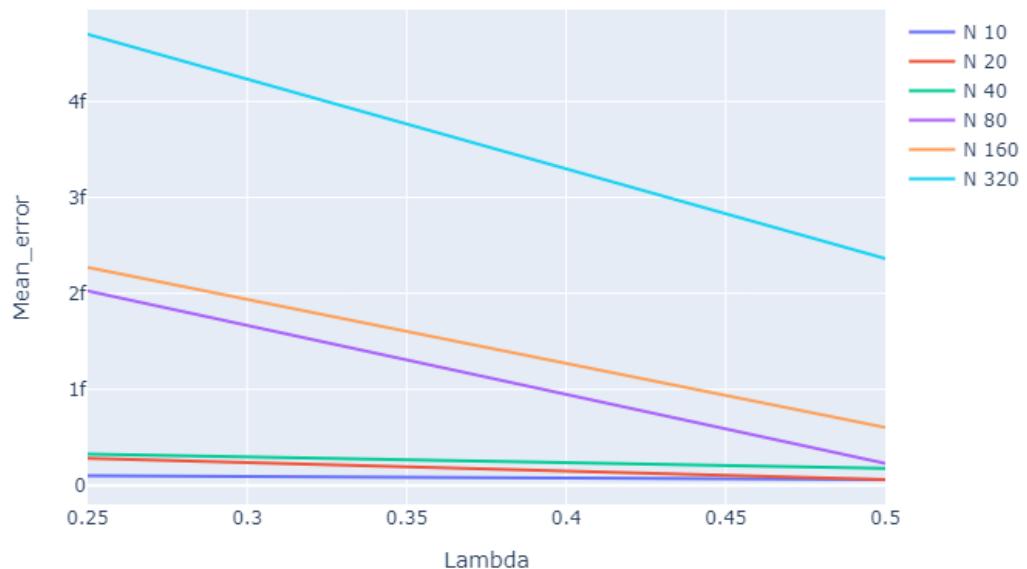


Figura 17: Erro em função de λ , para cada N

Error in function of N for each lambda
Método: Euler; simulation = a_old

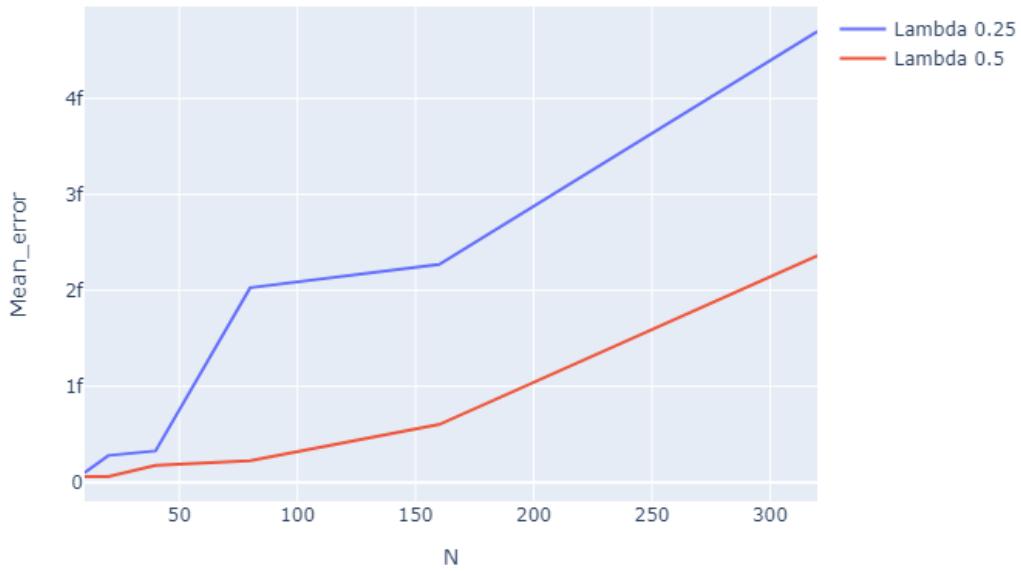


Figura 18: Erro em função de N, para cada λ

5.3.2 Euler implícito

Podemos ver, na figura 19, que esse método possui uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método anterior, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

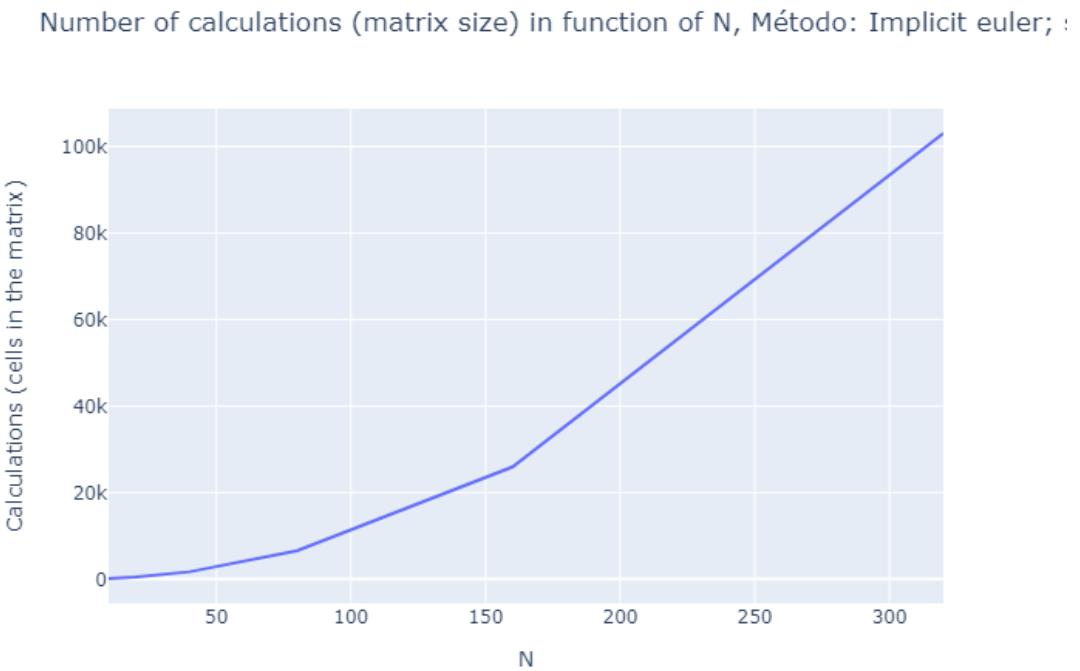


Figura 19: Número de cálculos em função de N

Analizando o erro, na figura 20, podemos notar também, que o mesmo aumenta com o refinamento da malha. Novamente, esse comportamento deve-se ao *float64*, e na condição supracitada, o número de cálculos aumenta, e portanto a propagação dos erros de aproximação também.

Error in function of N, Método: Implicit euler; simulation = a_old

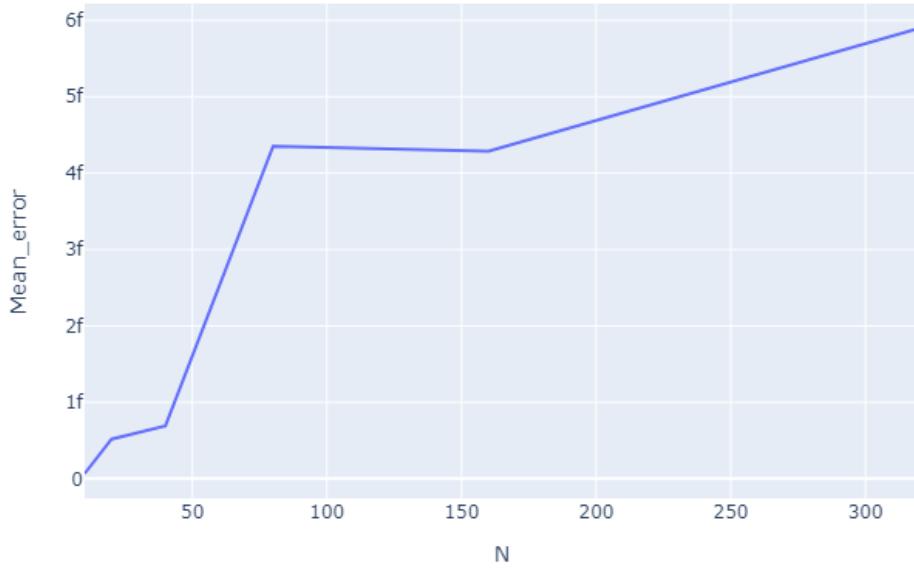


Figura 20: Erro em função de função de N

5.3.3 Crank-Nicolson

Podemos ver, na figura 21, que esse método possui também uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método de Euler, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

Number of calculations (matrix size) in function of N, Método: Crank-Nicolson

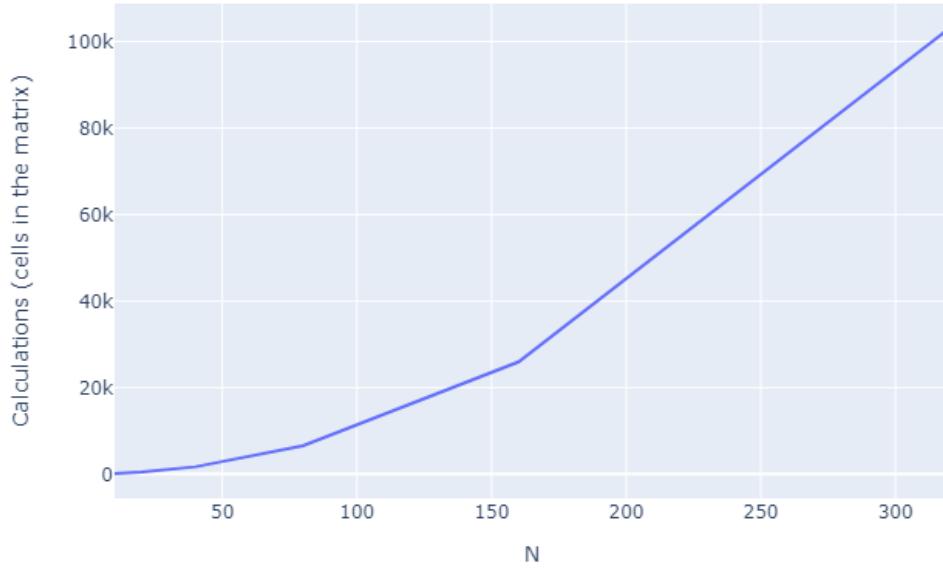


Figura 21: Número de cálculos em função de N

Analizando o erro, na figura 22, podemos notar também, que o mesmo aumenta com o refinamento da malha. Novamente, esse comportamento deve-se ao *float64*, e na condição supracitada, o número de cálculos aumenta, e portanto a propagação dos erros de aproximação também.

Error in function of N, Método: Crank-Nicolson; simulation = a_old

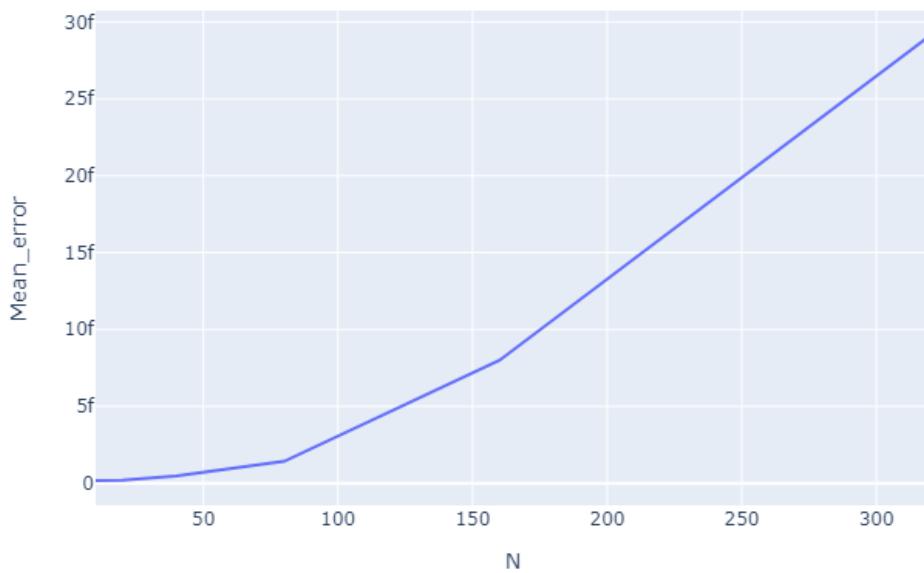


Figura 22: Erro em função de N

6 Problema A

As equações que descrevem esse problema são:

$$g_1(t) = 0$$

$$g_2(t) = 0$$

$$f(t,x) = 10\cos(10t)x^2(1-x)^2 - (\sin(10t) + 1)(12x^2 - 12x + 2)$$

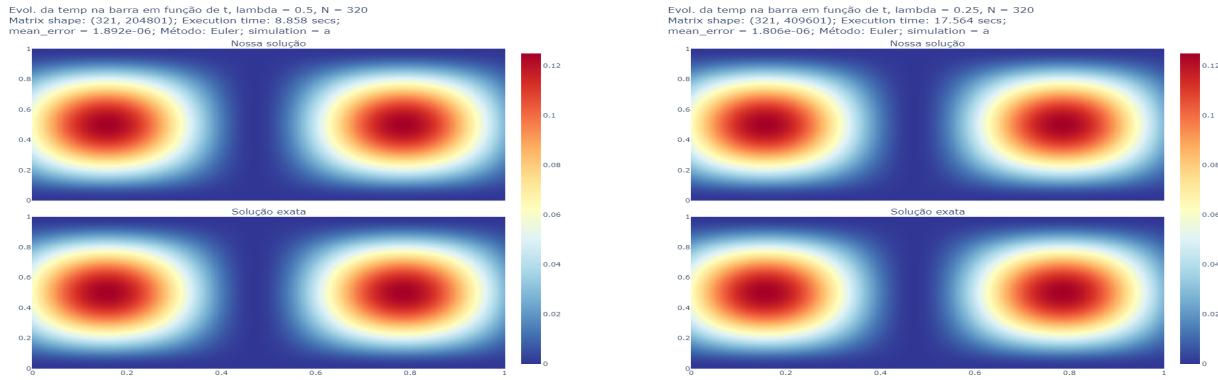
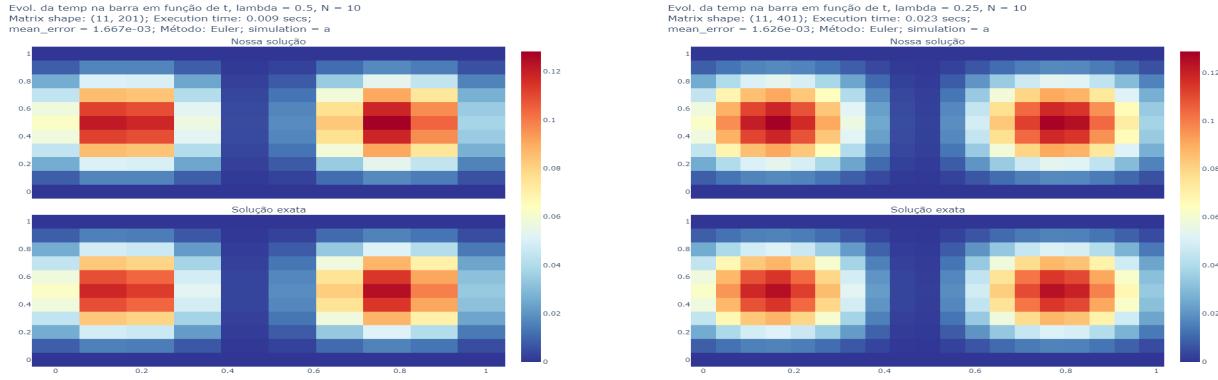
$$u_0(x) = x^2(1-x)^2$$

$$u(t,x) = (1+\sin(10t))x^2(1-x)^2$$

6.1 Analise de resultados

6.1.1 Euler

Podemos ver, na figura 23 uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções para ambos λ .



Evol. da temp na barra em função de t, lambda = 0.5, N = 320
Método: Euler, simulation = a

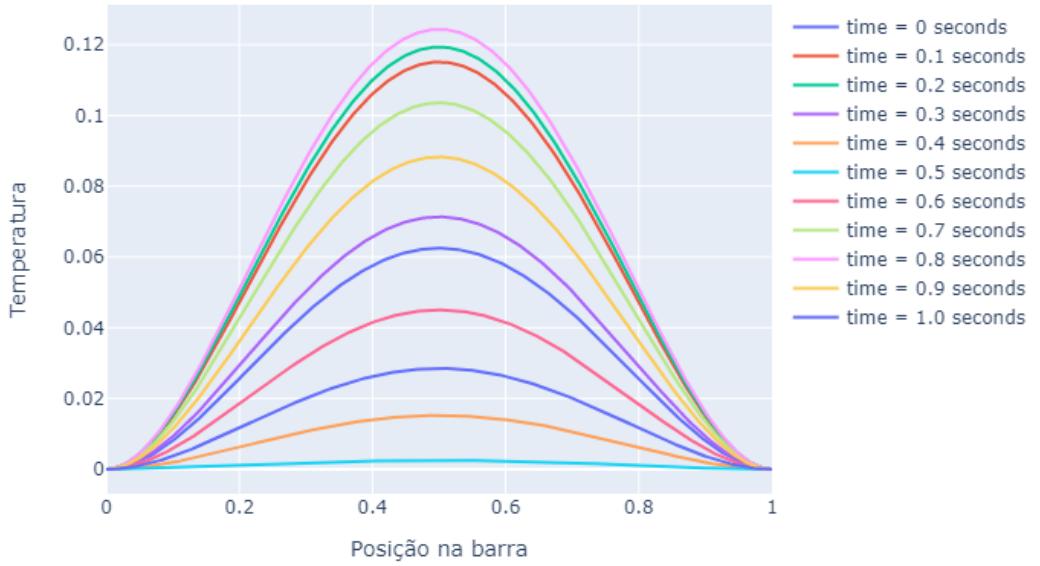


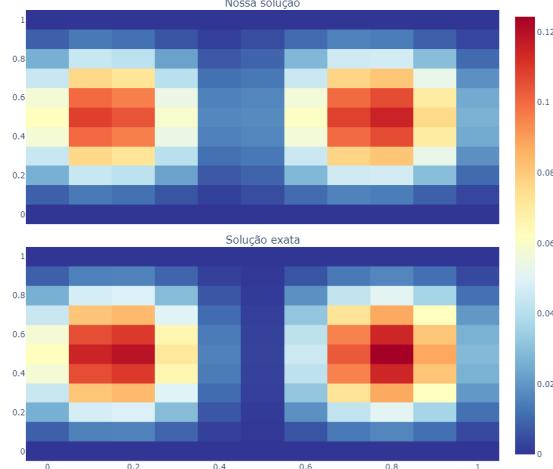
Figura 23: Podemos ver a evolução da temperatura na barra em diversos cenários, usando o método de Euler, com $\lambda = 0.5$ a esquerda, e $\lambda = 0.25$ a direita.

6.1.2 Euler implícito

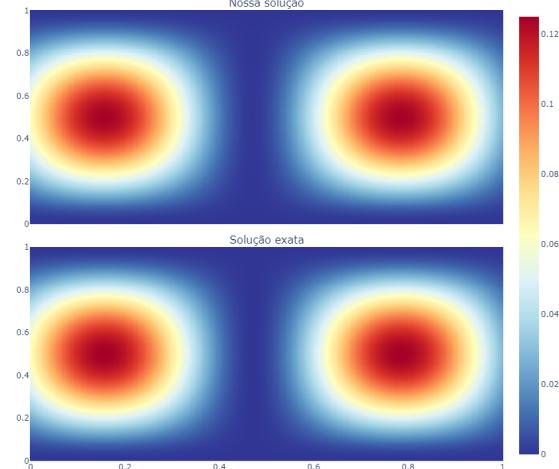
Podemos ver, na figura 24, uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções, vale notar, que embora tenhamos mantido a mesma precisão nesse cenário, o tempo de execução e o custo computacional para tal solução, foi extremamente reduzido, se comparado com o método de Euler. Vale também ressaltar a ausência da

dependência em λ

Evol. da temp na barra em função de t, lambda = 10, N = 10
 Matrix shape: (11, 11); Execution time: 0.005 secs;
 mean_error = 4.1706e-03; Método: Implicit euler; simulation = a



Evol. da temp na barra em função de t, lambda = 320, N = 320
 Matrix shape: (321, 321); Execution time: 4.226 secs;
 mean_error = 2.455e-04; Método: Implicit euler; simulation = a



Evol. da temp na barra em função de t, lambda = 320, N = 320
 Método: Implicit euler, simulation = a

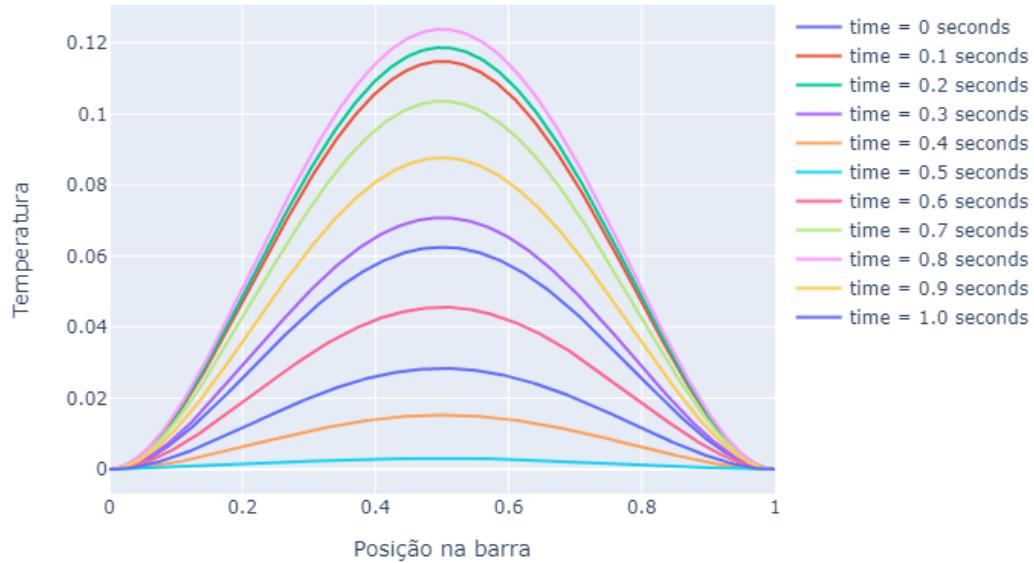
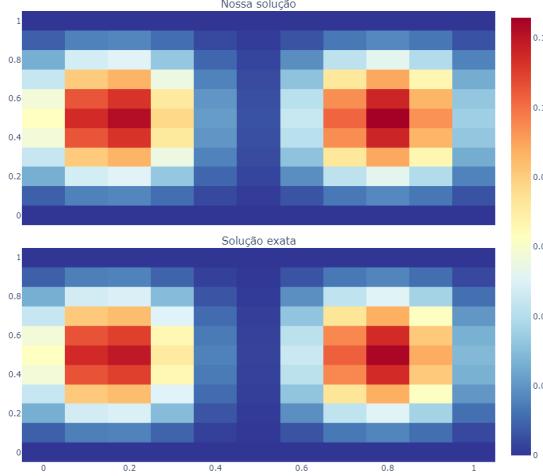


Figura 24: Resultados pelo método de Euler implícito

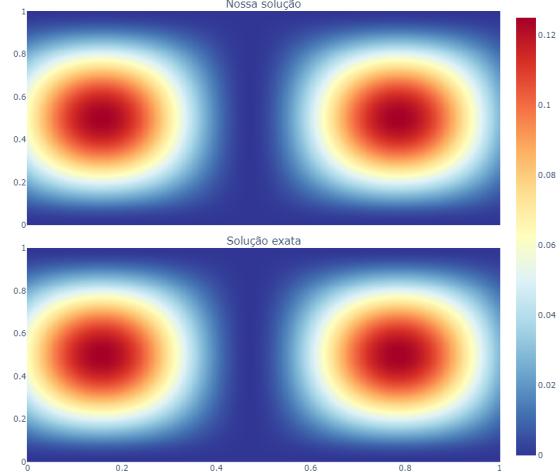
6.1.3 Crank-Nicolson

Podemos ver, na figura 25 uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções, o desempenho deste método neste cenário, mostrou-se muito semelhante ao do método implícito de Euler.

Evol. da temp na barra em função de t, lambda = 10, N = 10
Matrix shape: (11, 11); Execution time: 0.005 secs;
mean_error = 1.738e-03; Método: Crank-Nicolson; simulation = a



Evol. da temp na barra em função de t, lambda = 320, N = 320
Matrix shape: (321, 321); Execution time: 4.246 secs;
mean_error = 1.871e-06; Método: Crank-Nicolson; simulation = a
Nossa solução



Evol. da temp na barra em função de t, lambda = 320, N = 320
Método: Crank-Nicolson, simulation = a

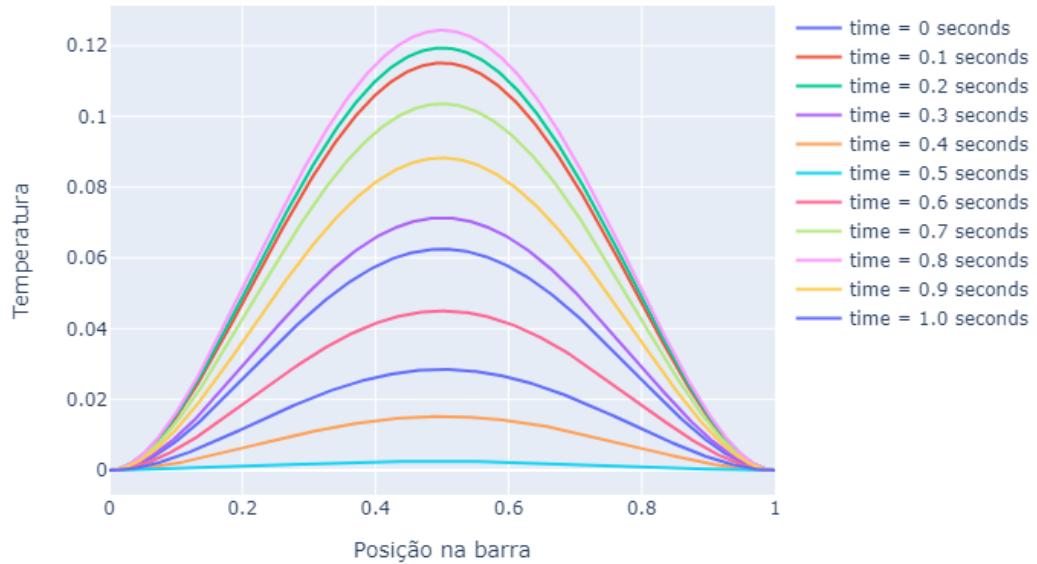


Figura 25: Resultados pelo método de Crank-Nicolson

6.2 Analise de erro

Para este problema, veremos o esperado ocorrer, conforme aumentarmos o refinamento da malha, menor será o nosso erro, em todos os métodos. Isso ocorre, pois estes métodos são derivados a partir de séries de Taylor, e tendem a ser uma boa representação da solução original apenas nas redondezas dos pontos onde estão centradas. Logo, é necessário que os pontos estejam próximos o suficiente, para que a solução venha a convergir.

6.2.1 Euler

Podemos ver nas figuras, 26, 27 e 28, como o erro se desenvolve nas simulações em diversos refinamentos de malha. Vale notar que nestes cenários, a influência do parâmetro λ , é mínima, e que existe um padrão para a distribuição deste erro. Isso ocorre, pois, a aproximação pelos

polinômios de Taylor, segue a solução exata, porém com atraso.

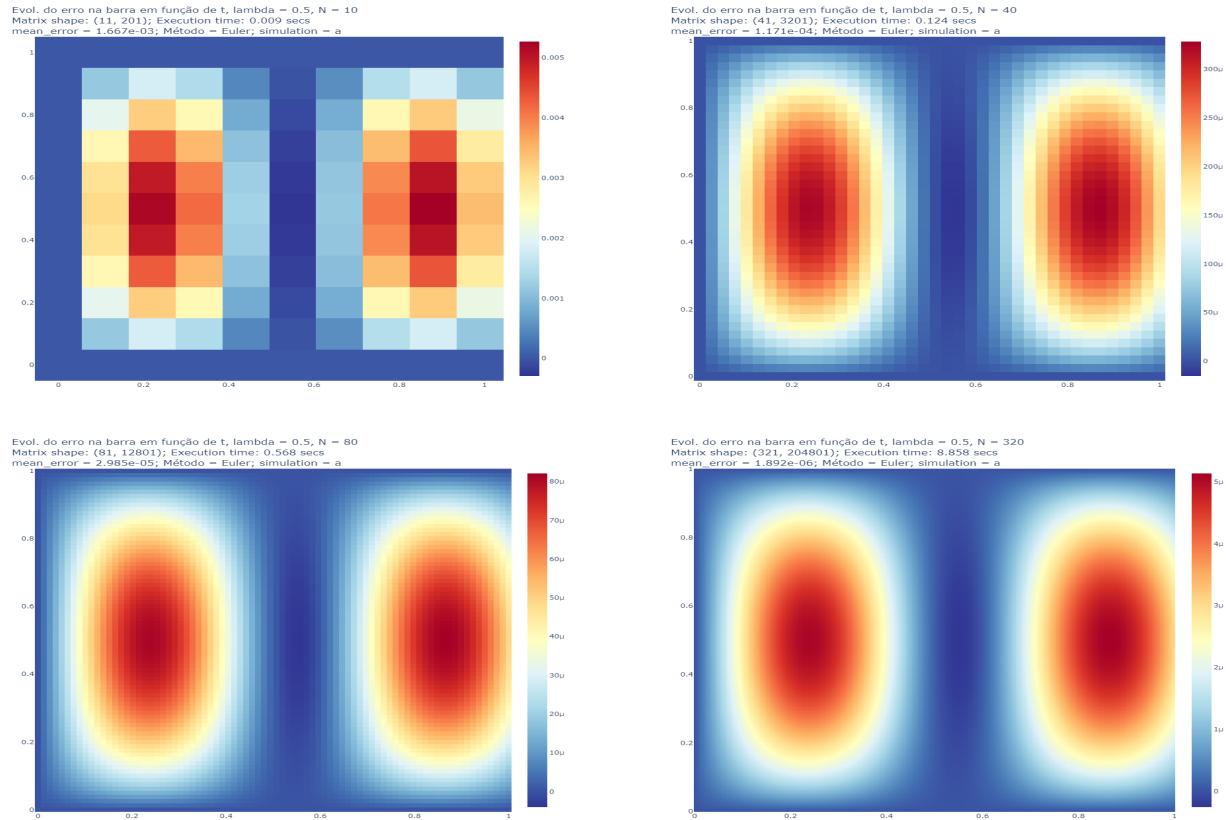


Figura 26: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.5$, método de Euler

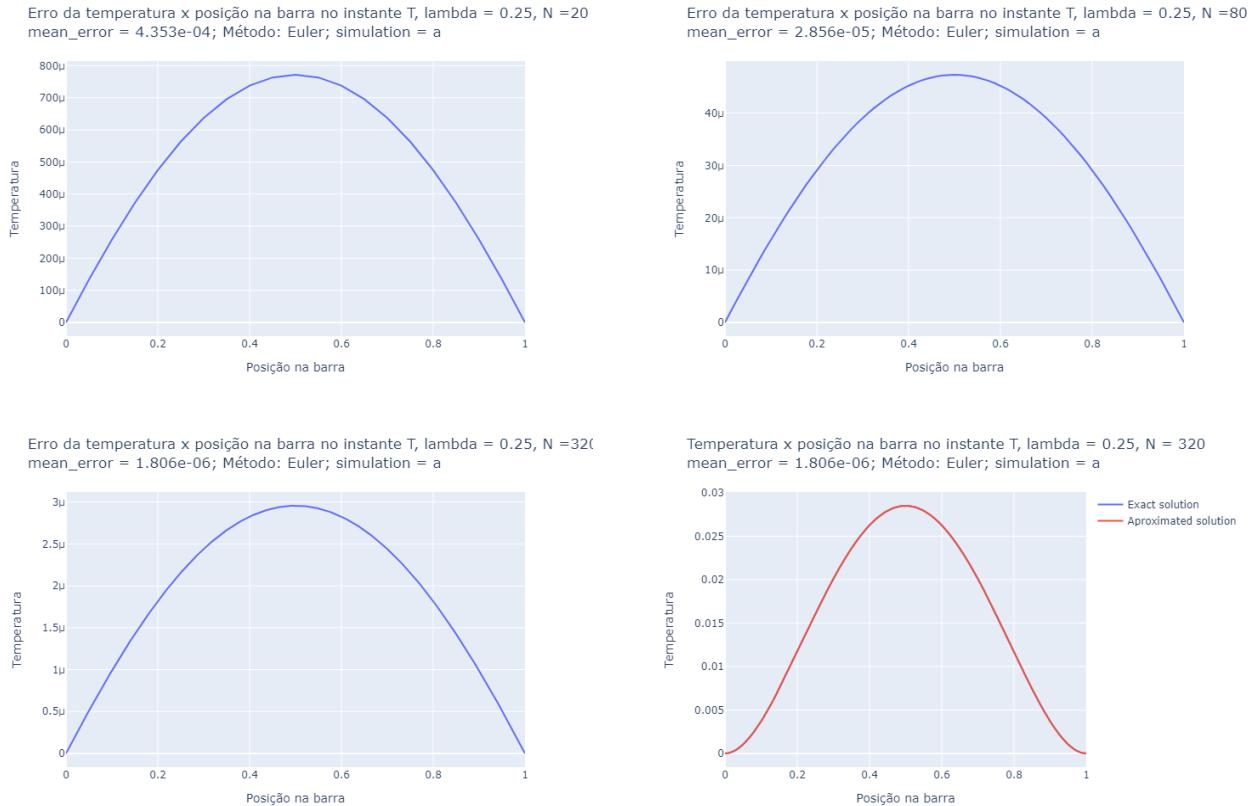
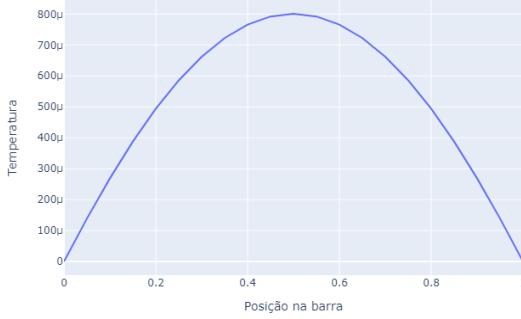
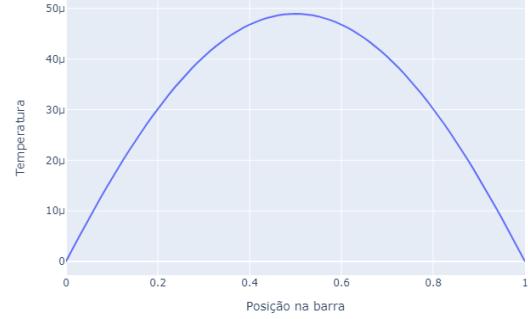


Figura 27: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.25$, método de Euler

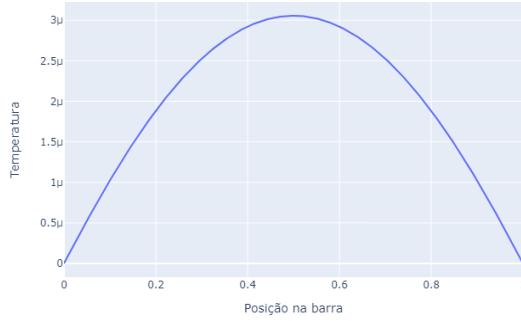
Erro da temperatura x posição na barra no instante T, lambda = 0.5, N = 20
mean_error = 4.504e-04; Método: Euler; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 0.5, N = 80
mean_error = 2.985e-05; Método: Euler; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 0.5, N = 320
mean_error = 1.892e-06; Método: Euler; simulation = a



Temperatura x posição na barra no instante T, lambda = 0.5, N = 320
mean_error = 1.892e-06; Método: Euler; simulation = a

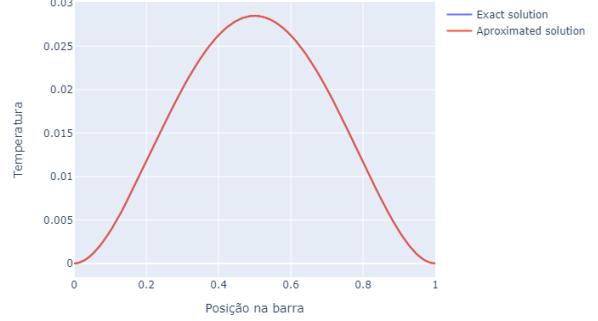
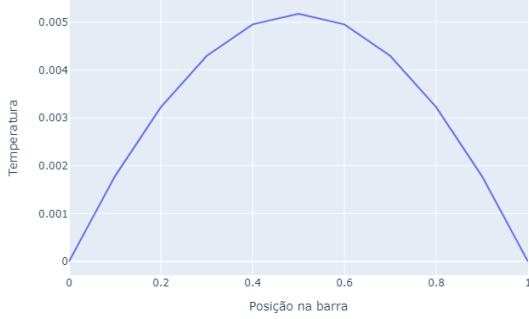


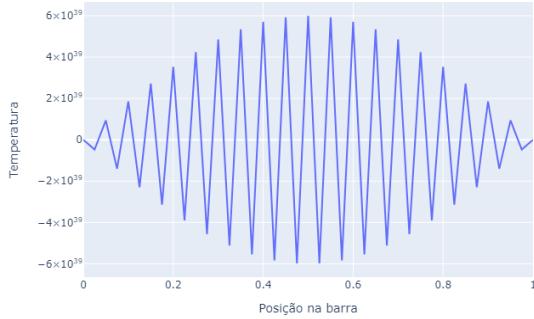
Figura 28: Distribuição do erro, no instante final, para diversos refinamentos de malha, com $\lambda = 0.5$, método de Euler

Nas figuras, 29, 30 e 31, podemos começar a ver as limitações deste método, onde para $\lambda > 0.50$, ele não converge para a solução. Para N pequenos, a solução não se distancia de forma significativa da solução correta, uma vez que não existem passos suficientes para que as instabilidades cresçam.

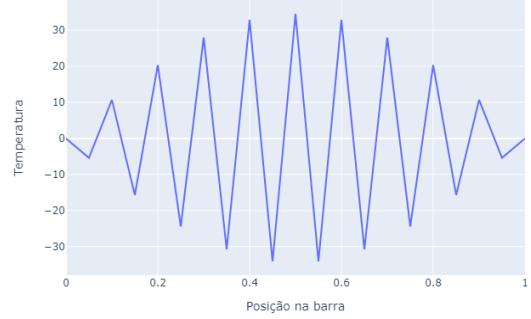
Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 10
mean_error = 1.651e-03; Método: Euler; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 40
mean_error = 1.006e+38; Método: Euler; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 20
mean_error = 1.596e+00; Método: Euler; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 0.51, N = 80
mean_error = 1.195e+195; Método: Euler; simulation = a

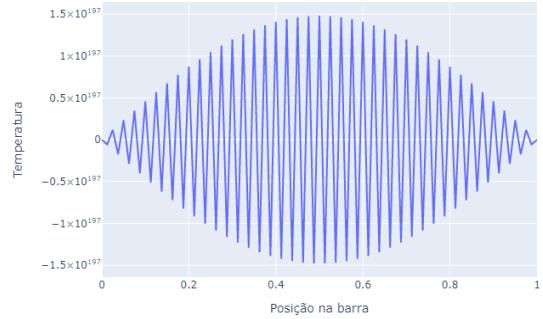
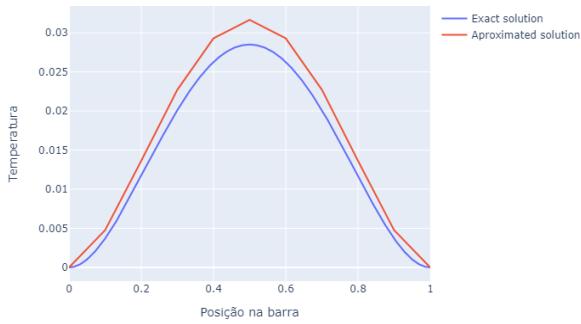
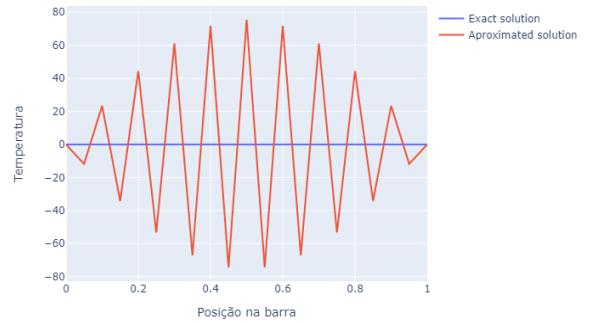


Figura 29: Distribuição do erro, no instante final, para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

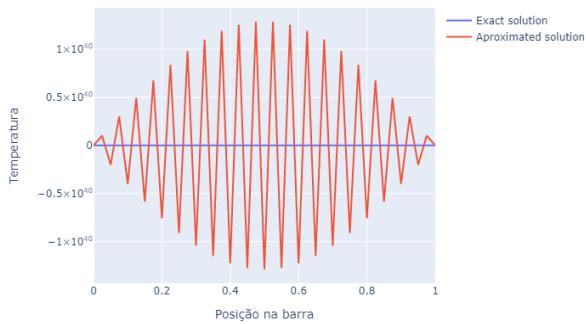
Temperatura x posição na barra no instante T, lambda = 0.51, N = 10
mean_error = 1.651e-03; Método: Euler; simulation = a



Temperatura x posição na barra no instante T, lambda = 0.51, N = 20
mean_error = 1.596e+00; Método: Euler; simulation = a



Temperatura x posição na barra no instante T, lambda = 0.51, N = 40
mean_error = 1.006e+38; Método: Euler; simulation = a



Temperatura x posição na barra no instante T, lambda = 0.51, N = 80
mean_error = 1.195e+195; Método: Euler; simulation = a

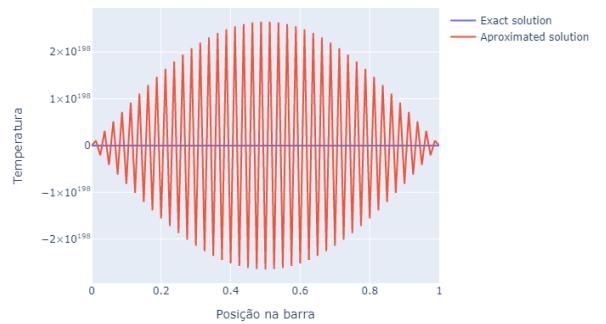


Figura 30: Estado final para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

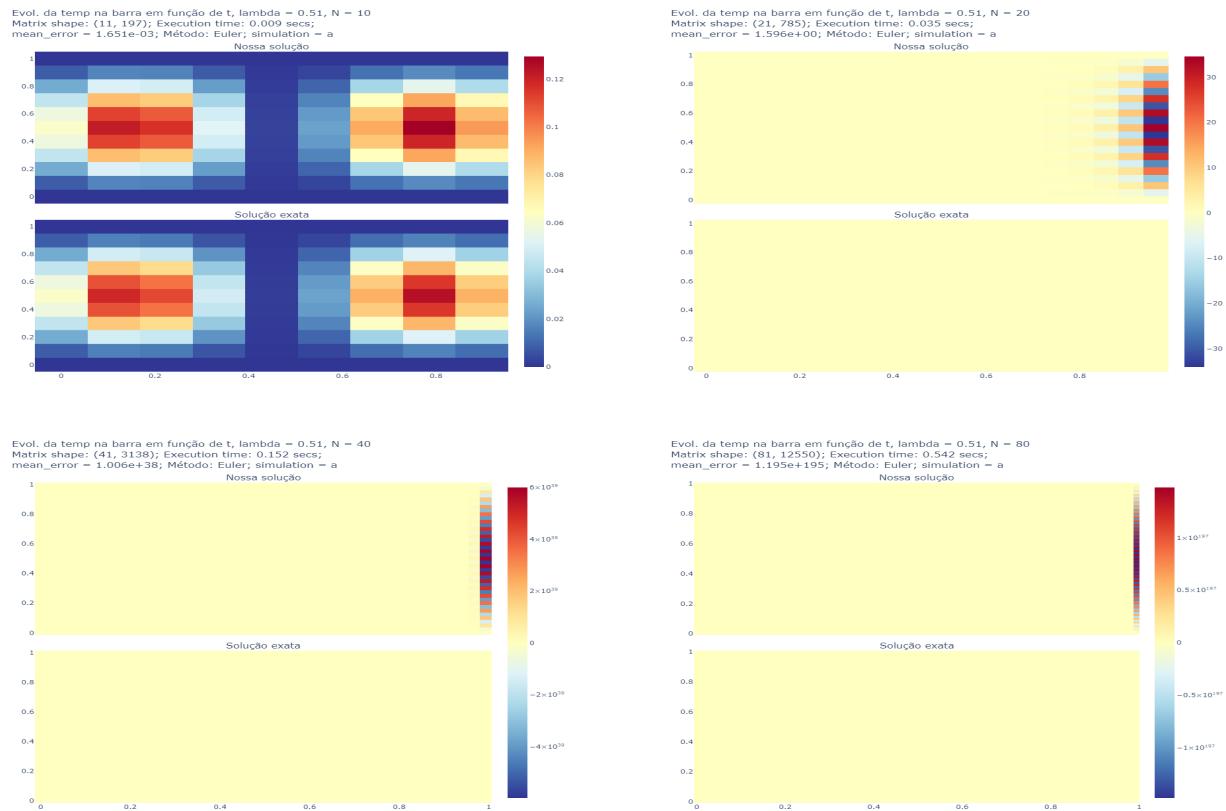


Figura 31: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

6.2.2 Euler implícito

Podemos ver nas figuras, 32 e 33 como o erro se desenvolve nas simulações em diversos refinamentos de malha.

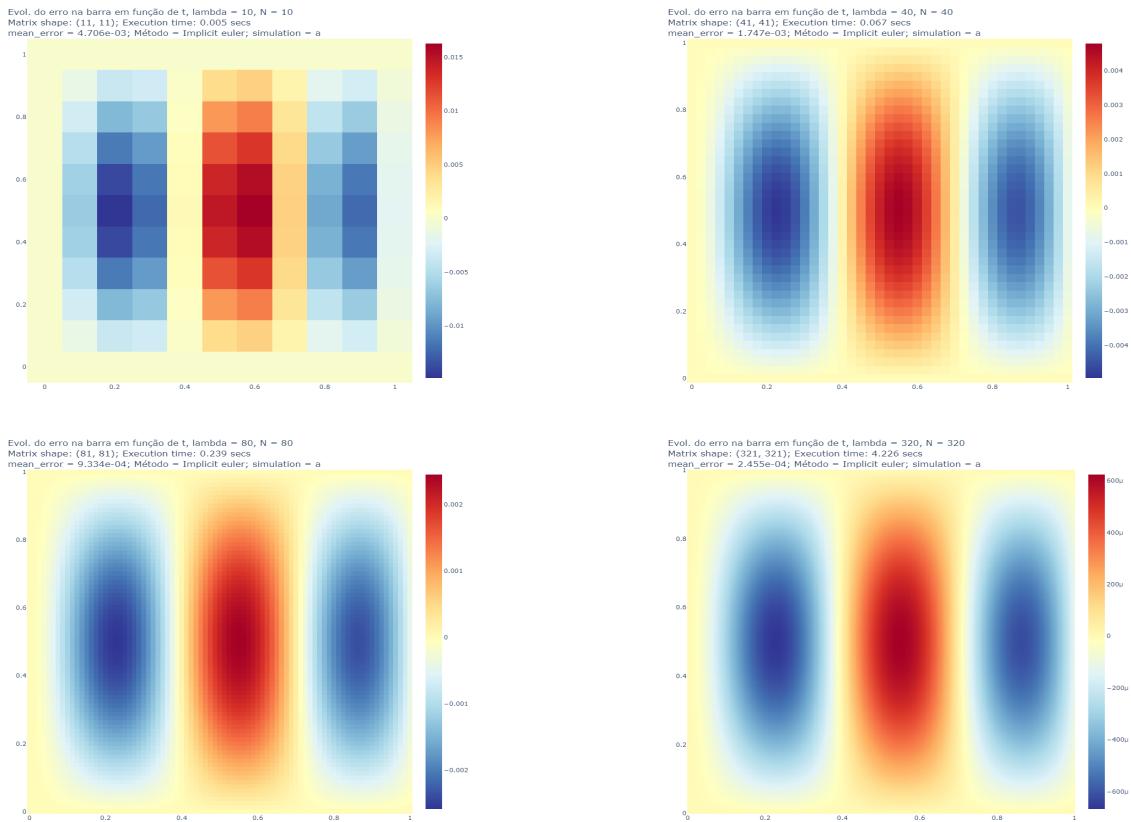


Figura 32: Distribuição do erro para diversos refinamentos de malha, método de Euler implícito

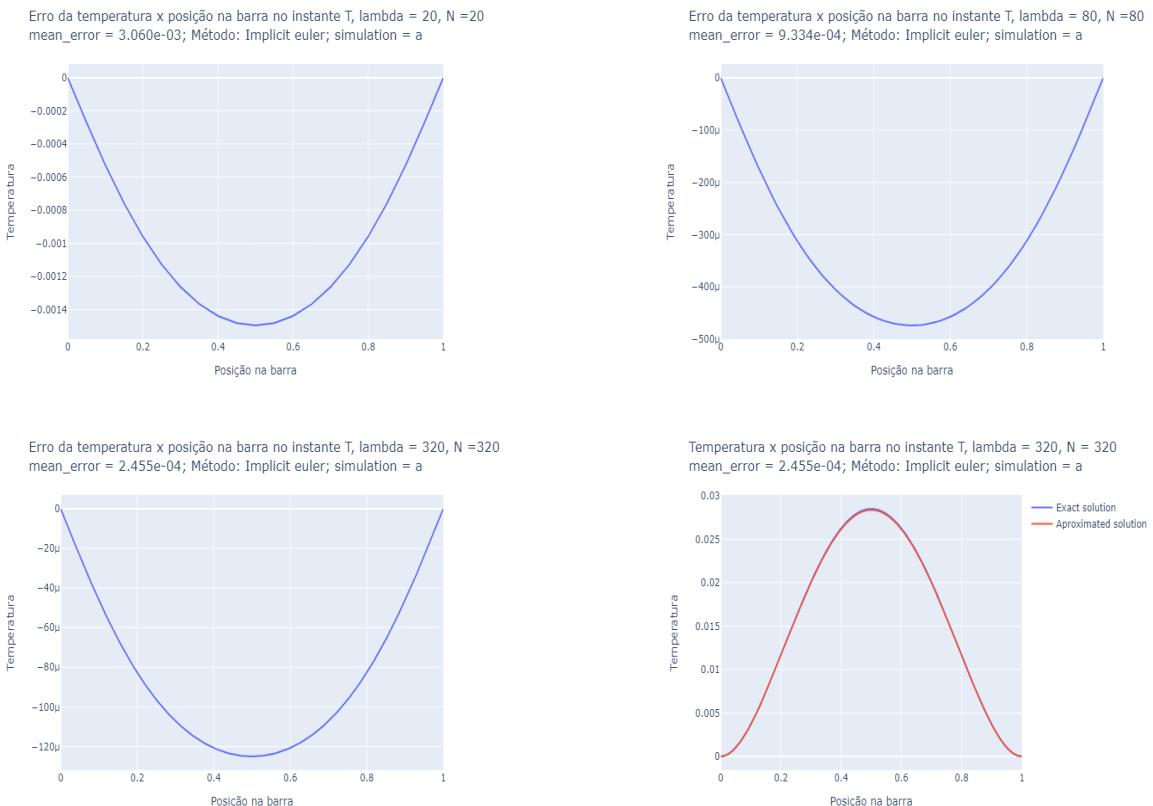


Figura 33: Distribuição do erro, no instante final, para diversos refinamentos de malha, método de Euler implícito

6.2.3 Crank-Nicolson

Podemos ver nas figuras, 34 e 35 como o erro se desenvolve nas simulações em diversos refinamentos de malha.

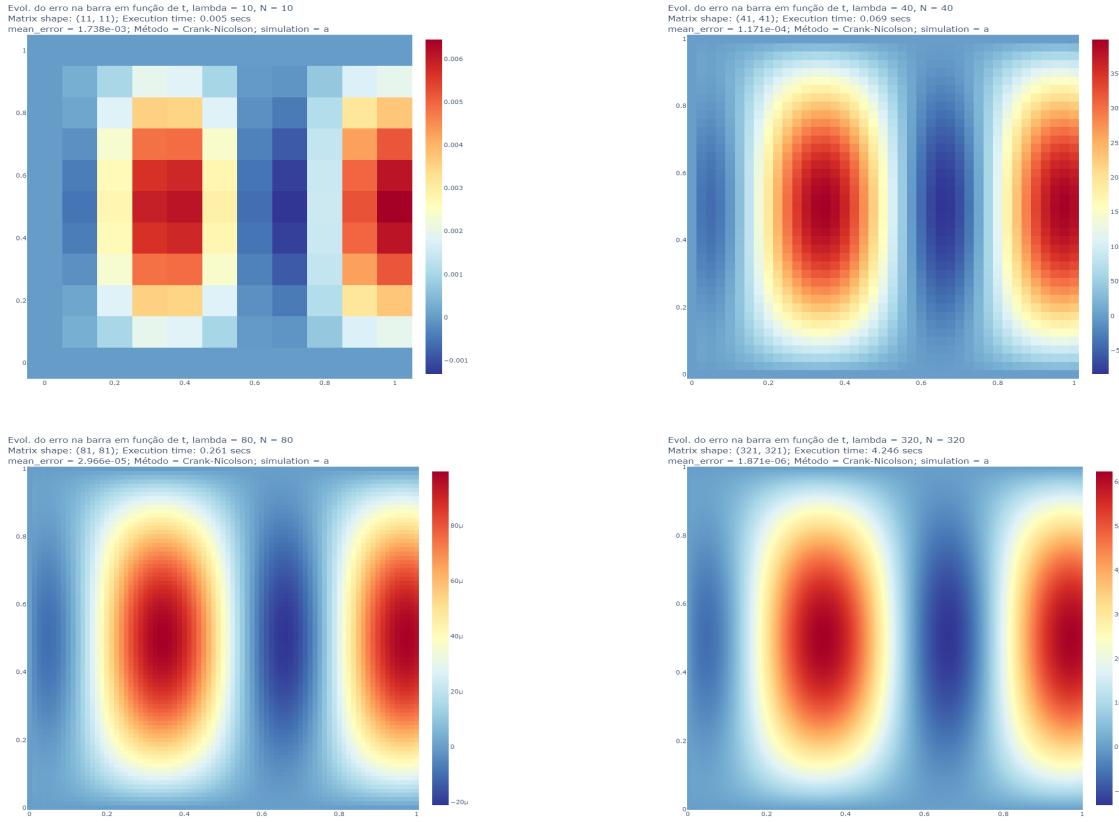
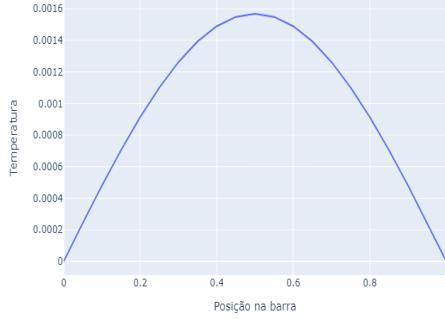
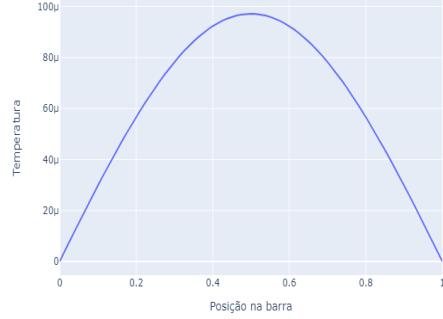


Figura 34: Distribuição do erro para diversos refinamentos de malha, método de Crank-Nicolson

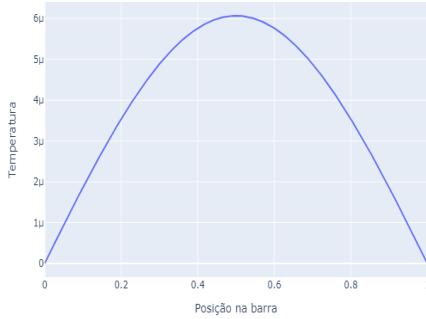
Erro da temperatura x posição na barra no instante T, lambda = 20, N = 20
mean_error = 4.562e-04; Método: Crank-Nicolson; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 80, N = 80
mean_error = 2.966e-05; Método: Crank-Nicolson; simulation = a



Erro da temperatura x posição na barra no instante T, lambda = 320, N = 320
mean_error = 1.871e-06; Método: Crank-Nicolson; simulation = a



Temperatura x posição na barra no instante T, lambda = 320, N = 320
mean_error = 1.871e-06; Método: Crank-Nicolson; simulation = a

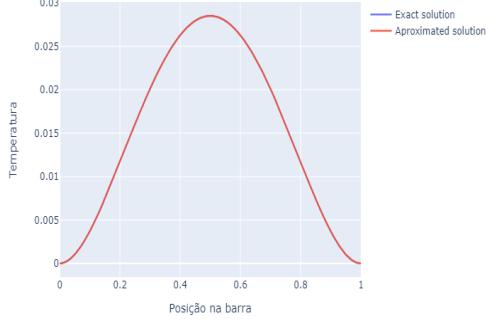


Figura 35: Distribuição do erro, no instante final, para diversos refinamentos de malha, método de Crank-Nicolson

6.3 Conclusões finais e análises

Neste cenário, podemos chegar a algumas conclusões muito importantes. Embora o método de Euler tenha boas aproximações, ele é muito custoso computacionalmente. Entre os métodos implícitos, ambos apresentam custos similares, e inferiores ao método de Euler. Vale notar, que o método de Implícito de Euler, para a mesma discretização, apresenta resultados inferiores ao método de Crank-Nicolson. E o de Crank-Nicolson, resultados semelhantes ao de Euler.

6.3.1 Euler

Podemos ver, na figura 36, que esse método possui uma complexidade de $O(N^3)$, fazendo o número de pontos a serem determinados a crescer de forma muito rápida, e por conseguinte, o tempo e os recursos necessários.

Number of calculations (matrix size) in function of N for each lambda
Método: Euler; simulation = a

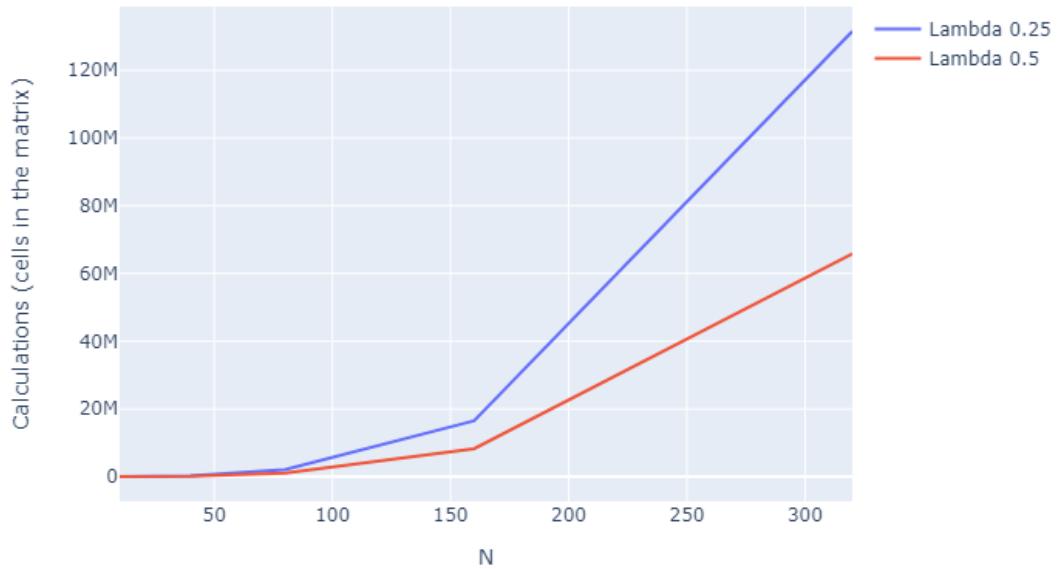


Figura 36: Número de cálculos em função de função de N

Analizando o erro, nas figuras, 37 e 38, podemos notar também, que, neste método, o λ não influência na precisão do método. Apenas a discretização no domínio do espaço, desde que a restrição de $\lambda \leq 0.5$, seja mantida. Logo, discretizações maiores que isso, são desperdício computacional. Para ambos os λ 's, o fator de redução é: 0.26.

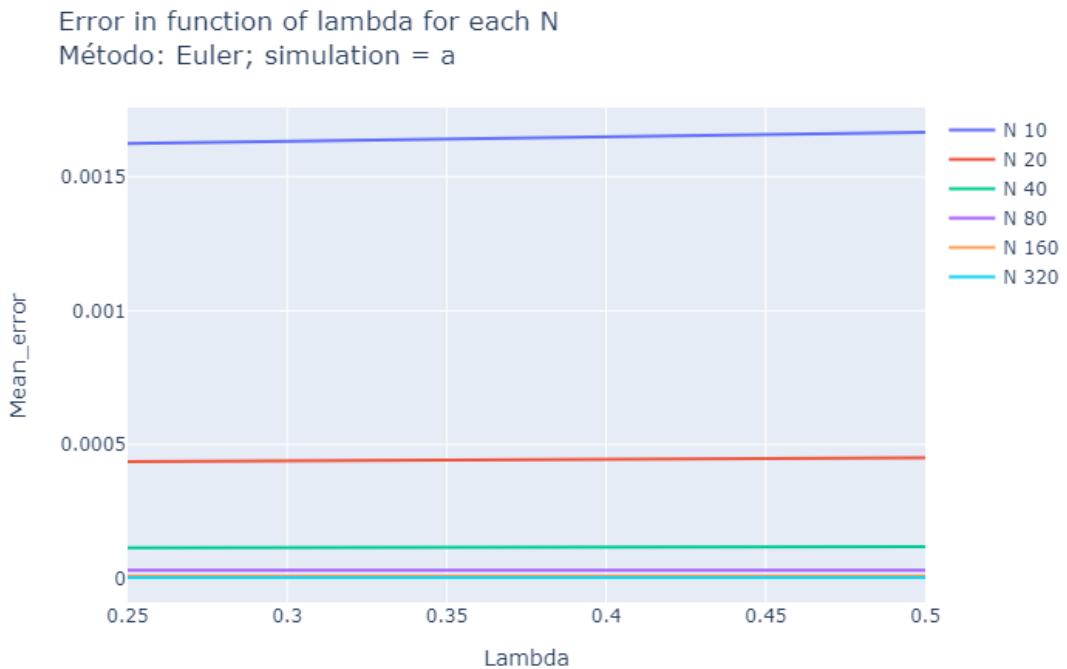


Figura 37: Erro em função de função de λ , para cada N

Error in function of N for each lambda
Método: Euler; simulation = a

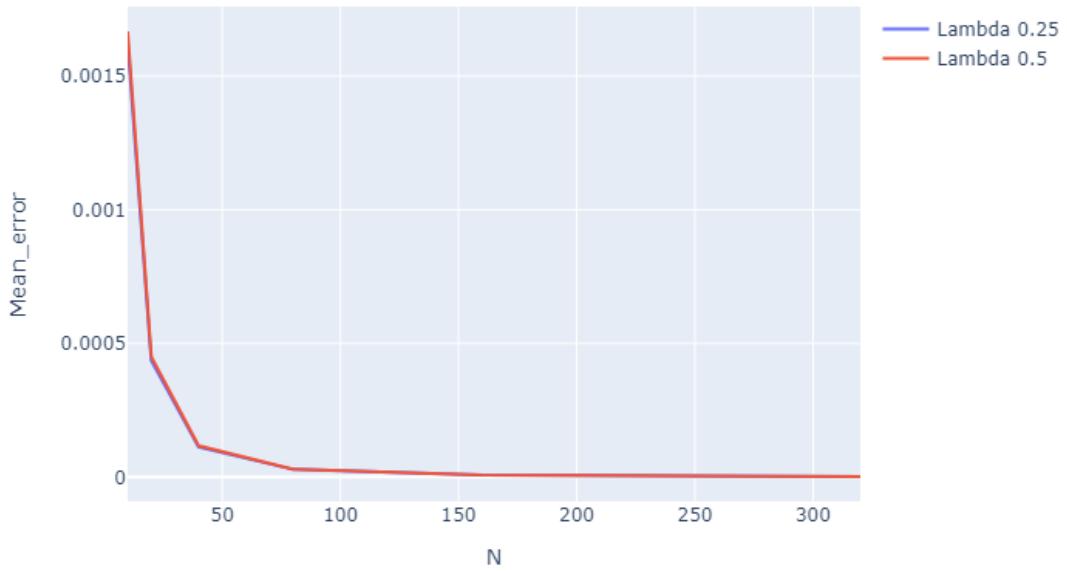


Figura 38: Erro em função de N, para cada λ

6.3.2 Euler implícito

Podemos ver, na figura 39, que esse método possui uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método anterior, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

Number of calculations (matrix size) in function of N, Método: Implicit euler; :

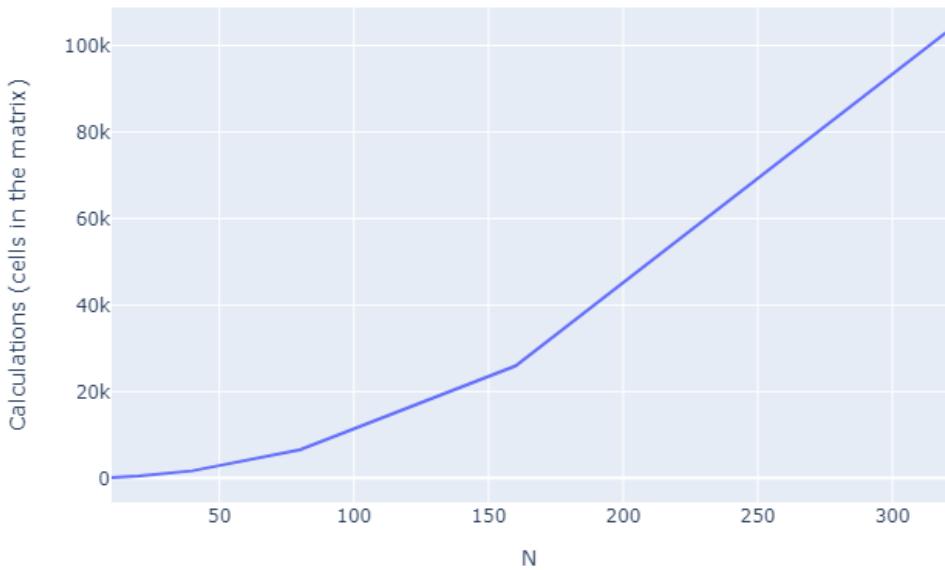


Figura 39: Número de cálculos em função de N

Analizando o erro, na figura 40, podemos notar também, que o mesmo diminui com o refinamento da malha. Para este método, o fator de redução é: 0.56.

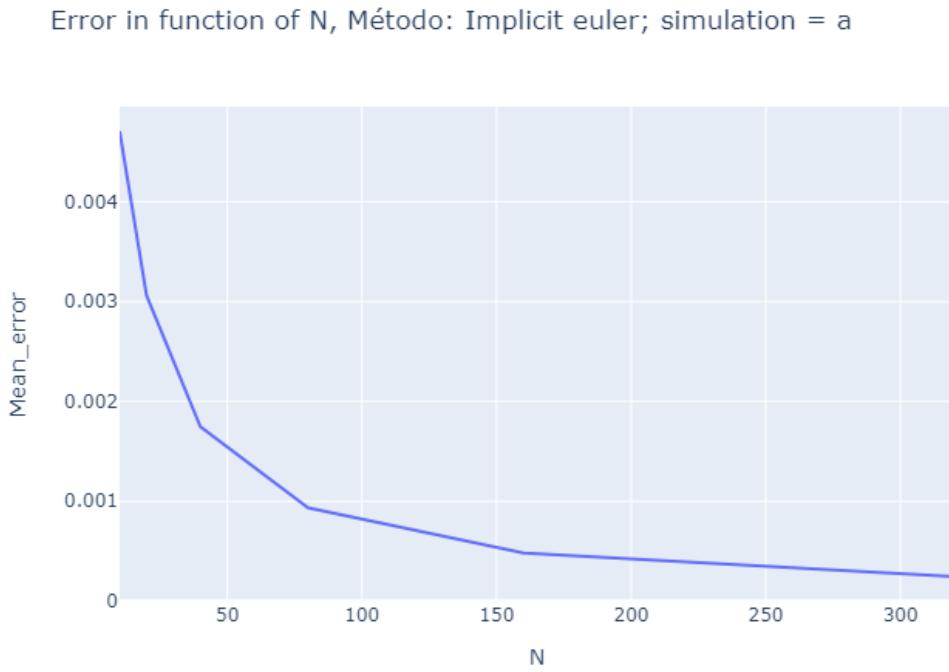


Figura 40: Erro em função de função de N

6.3.3 Crank-Nicolson

Podemos ver, na figura 41, que esse método possui também uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método de Euler, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

Number of calculations (matrix size) in function of N, Método: Crank-Nicolson

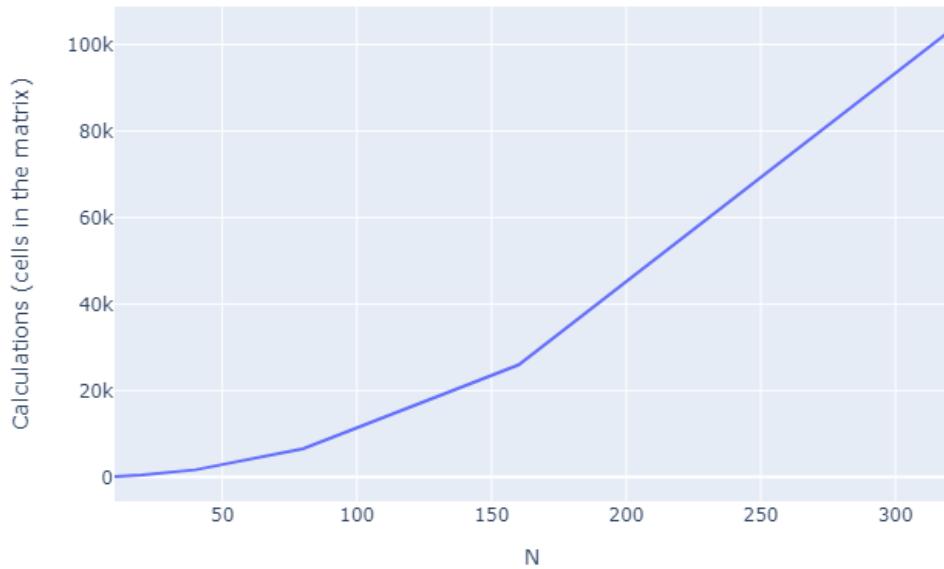


Figura 41: Número de cálculos em função de função de N

Analizando o erro, na figura 42, podemos notar também, que o mesmo aumenta com o refinamento da malha. Para este método, **o fator de redução é: 0.25**.

Error in function of N, Método: Crank-Nicolson; simulation = a

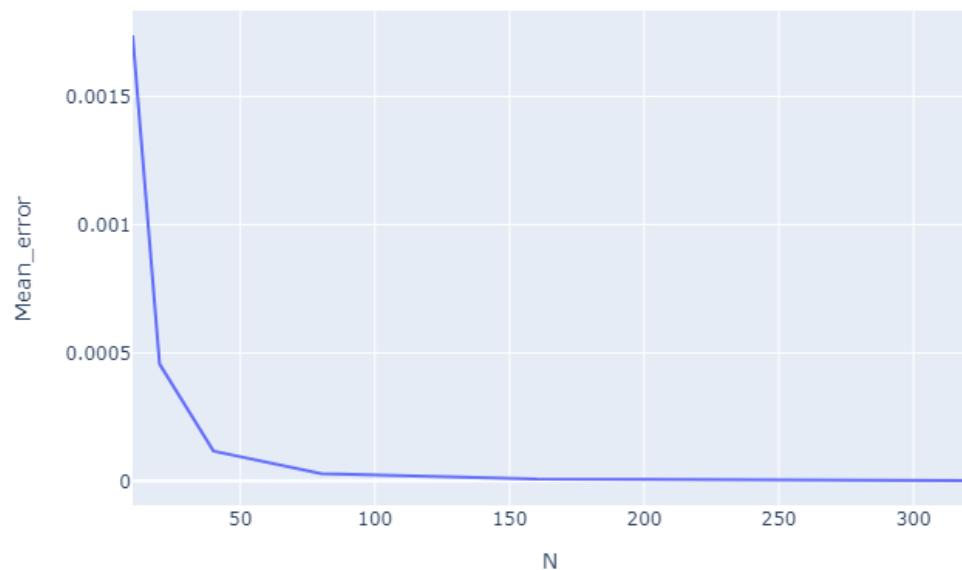


Figura 42: Erro em função de função de N

7 Problema B

As equações que descrevem esse problema são:

$$g_1(t) = e^t$$

$$g_2(t) = e^{t-1} \cos(5t)$$

$$f(t,x) = -5e^{t-x}((x+2t)\sin(5tx) - 5t^2\cos(5tx))$$

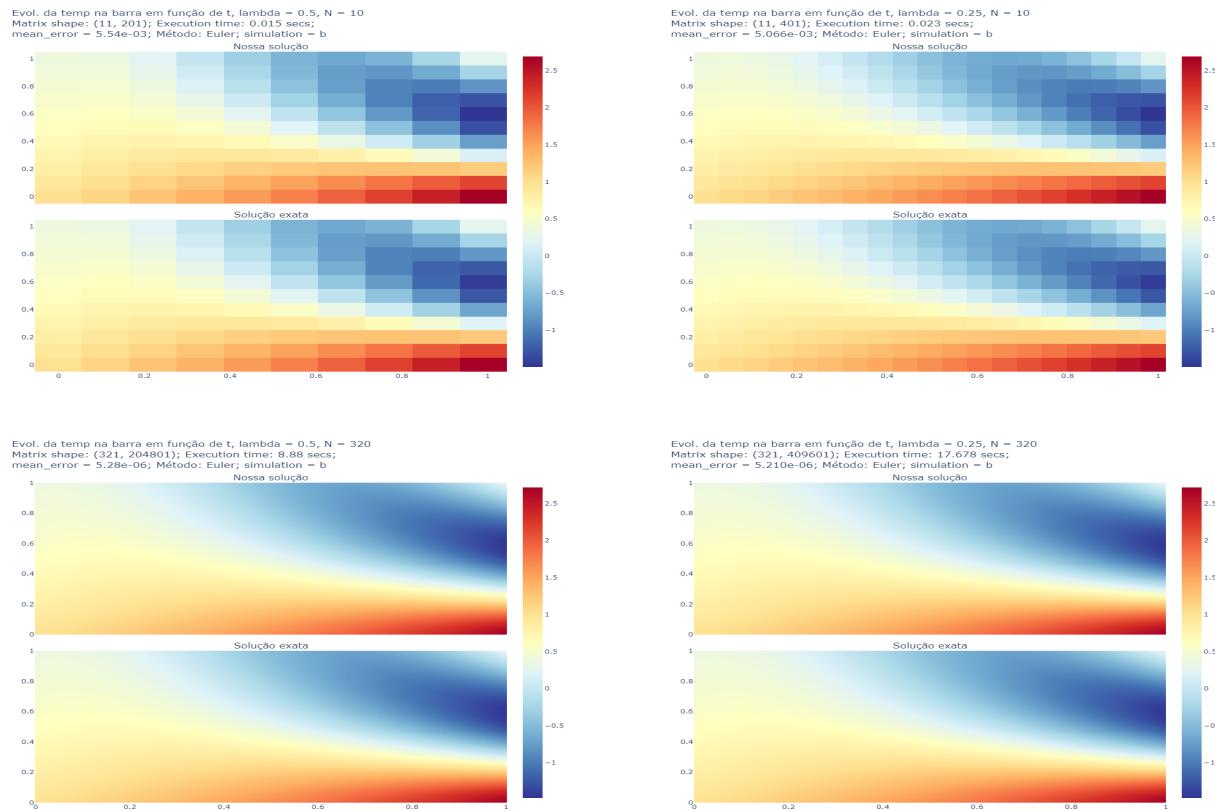
$$u_0(x) = e^{-x}$$

$$u(t,x) = e^{t-x}\cos(5tx)$$

7.1 Analise de resultados

7.1.1 Euler

Podemos ver, na figura 43 uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções para ambos λ .



Evol. da temp na barra em função de t, lambda = 0.5, N = 320
Método: Euler; simulation = b

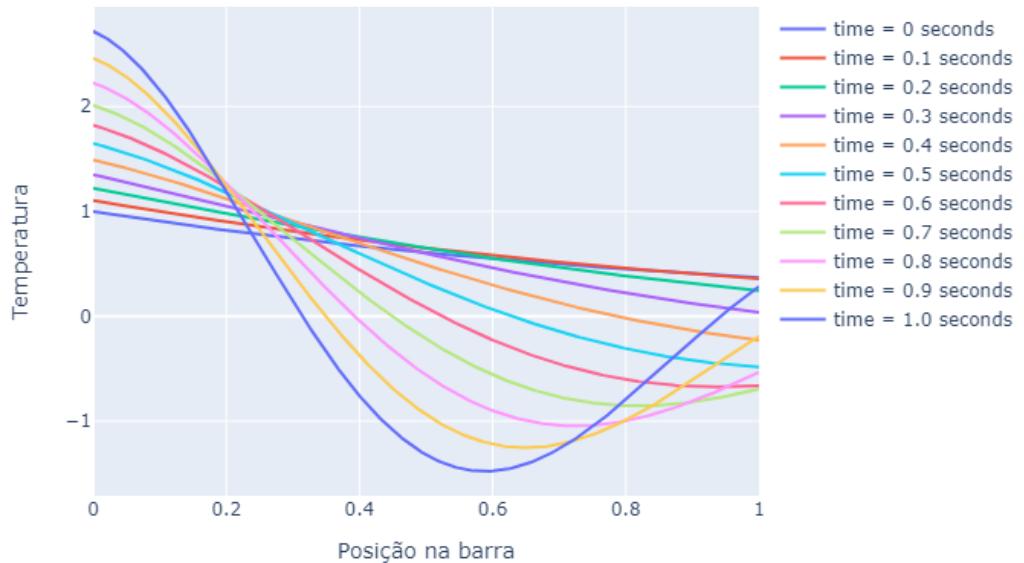
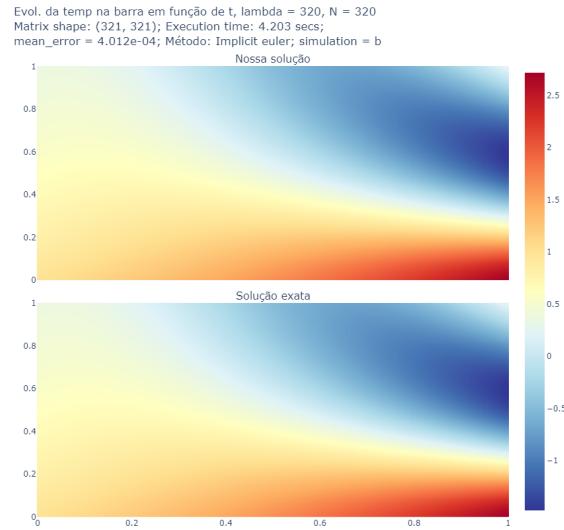
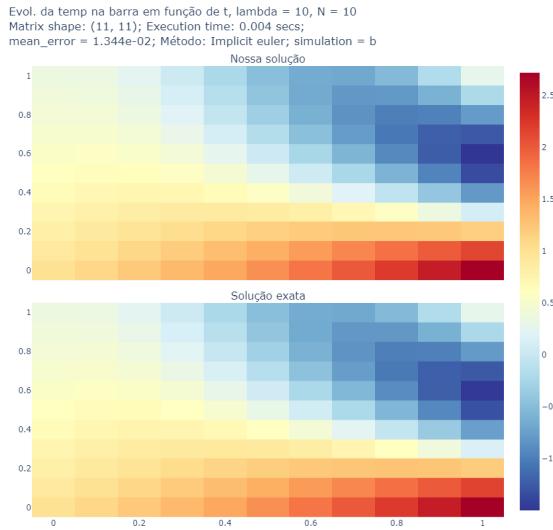


Figura 43: Podemos ver a evolução da temperatura na barra em diversos cenários, usando o método de Euler, com $\lambda = 0.5$ a esquerda, e $\lambda = 0.25$ a direita.

7.1.2 Euler implícito

Podemos ver, na figura 44, uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções, vale notar, que embora tenhamos mantido a mesma precisão nesse cenário, o tempo de execução e o custo computacional para tal solução, foi extremamente reduzido, se comparado com o método de Euler. Vale também ressaltar a ausência da

dependência em λ



Evol. da temp na barra em função de t, lambda = 320, N = 320
 Método: Implicit euler, simulation = b

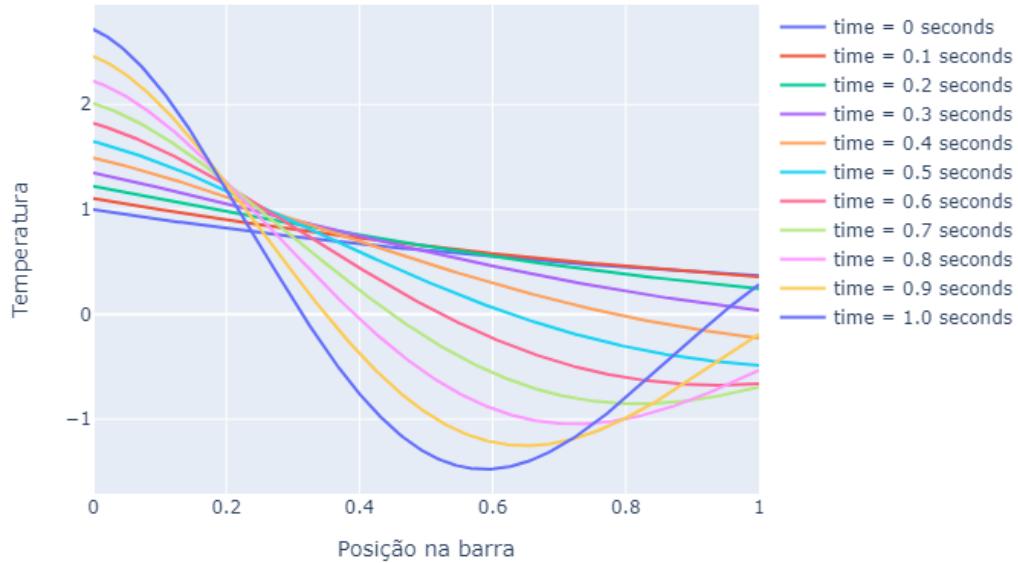
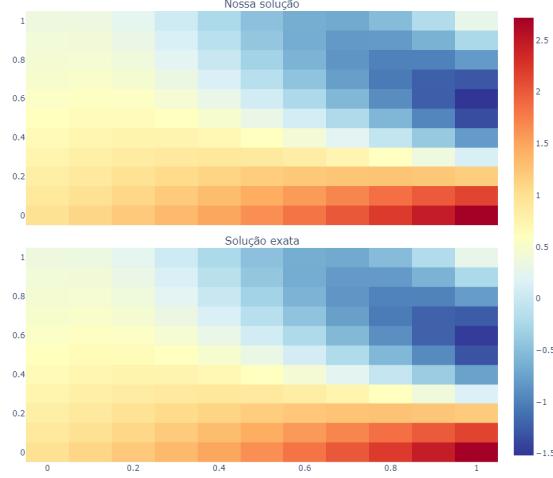


Figura 44: Resultados pelo método de Euler implícito

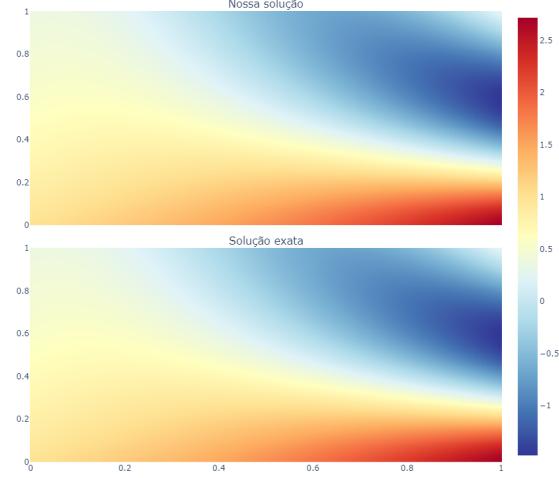
7.1.3 Crank-Nicolson

Podemos ver, na figura 45 uma grande semelhança entre a solução exata e a solução calculada numericamente em ambas as resoluções, o desempenho deste método neste cenário, mostrou-se muito semelhante ao do método implícito de Euler.

Evol. da temp na barra em função de t, lambda = 10, N = 10
Matrix shape: (11, 11); Execution time: 0.007 secs;
mean_error = 5.297e-03; Método: Crank-Nicolson; simulation = b



Evol. da temp na barra em função de t, lambda = 320, N = 320
Matrix shape: (321, 321); Execution time: 4.275 secs;
mean_error = 4.824e-06; Método: Crank-Nicolson; simulation = b



Evol. da temp na barra em função de t, lambda = 320, N = 320
Método: Crank-Nicolson, simulation = b

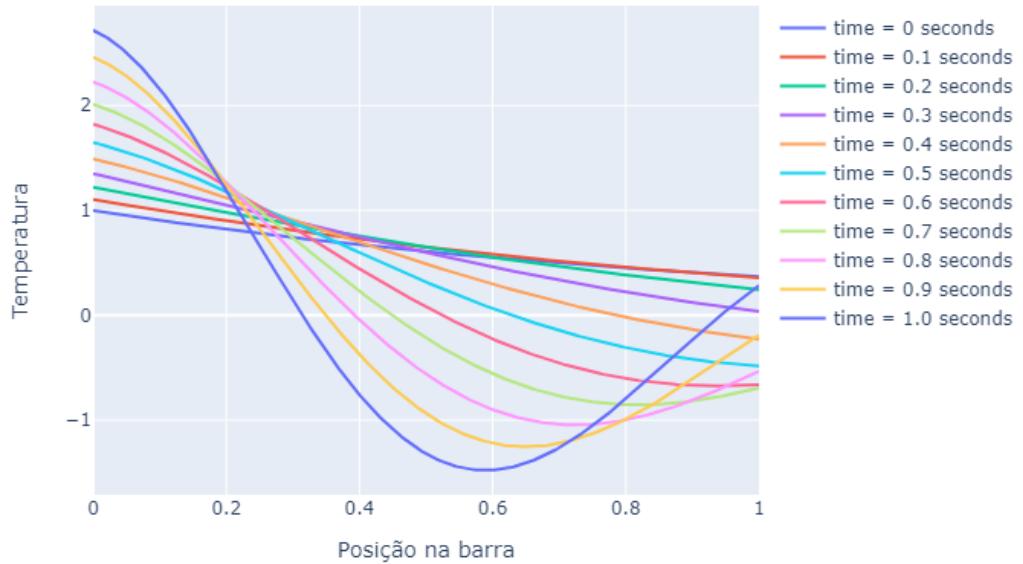


Figura 45: Resultados pelo método de Crank-Nicolson

7.2 Analise de erro

Para este problema, veremos o esperado ocorrer, conforme aumentarmos o refinamento da malha, menor será o nosso erro, em todos os métodos. Isso ocorre, pois estes métodos são derivados a partir de séries de Taylor, e tendem a ser uma boa representação da solução original apenas nas redondezas dos pontos onde estão centradas. Logo, é necessário que os pontos estejam próximos o suficiente, para que a solução venha a convergir.

7.2.1 Euler

Podemos ver nas figuras, 46, 47 e 48, como o erro se desenvolve nas simulações em diversos refinamentos de malha. Vale notar que nestes cenários, a influência do parâmetro λ , é mínima, e que existe um padrão para a distribuição deste erro. Isso ocorre, pois, a aproximação pelos

polinômios de Taylor, segue a solução exata, porém com atraso.

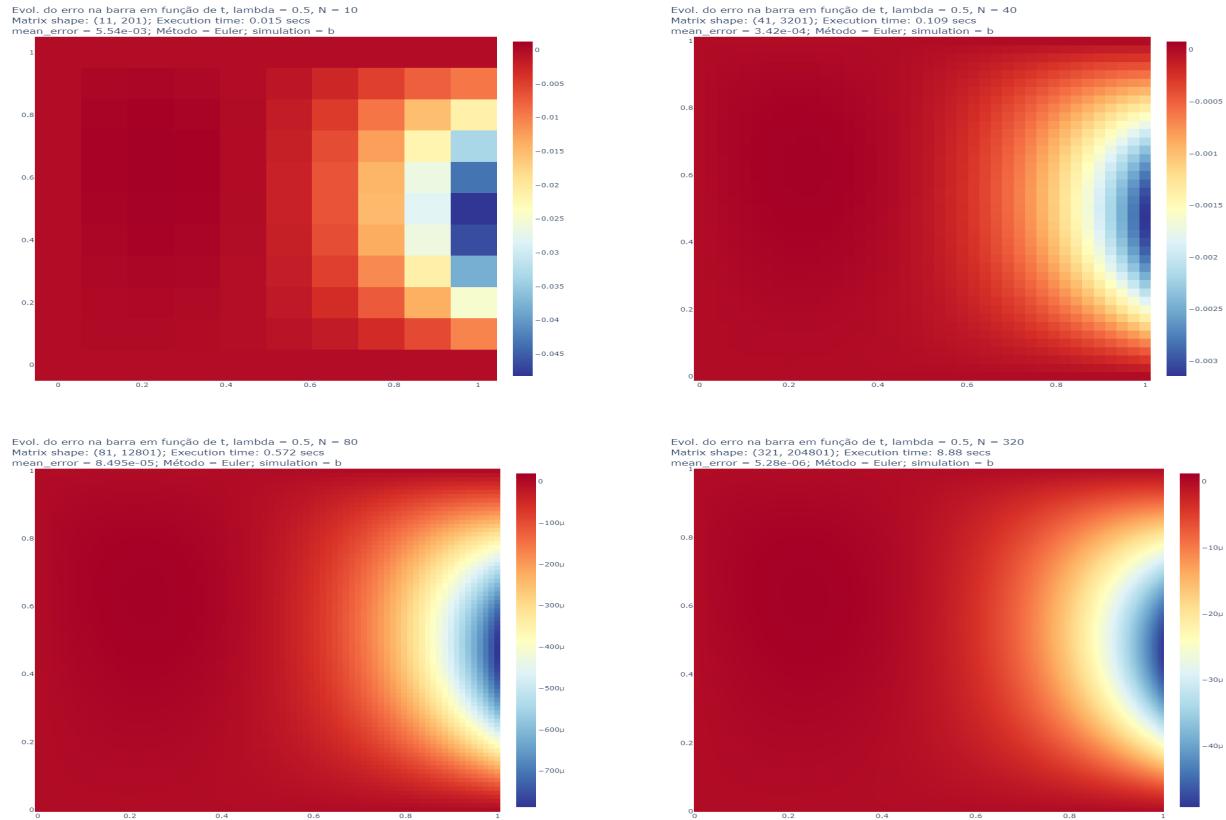


Figura 46: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.5$, método de Euler

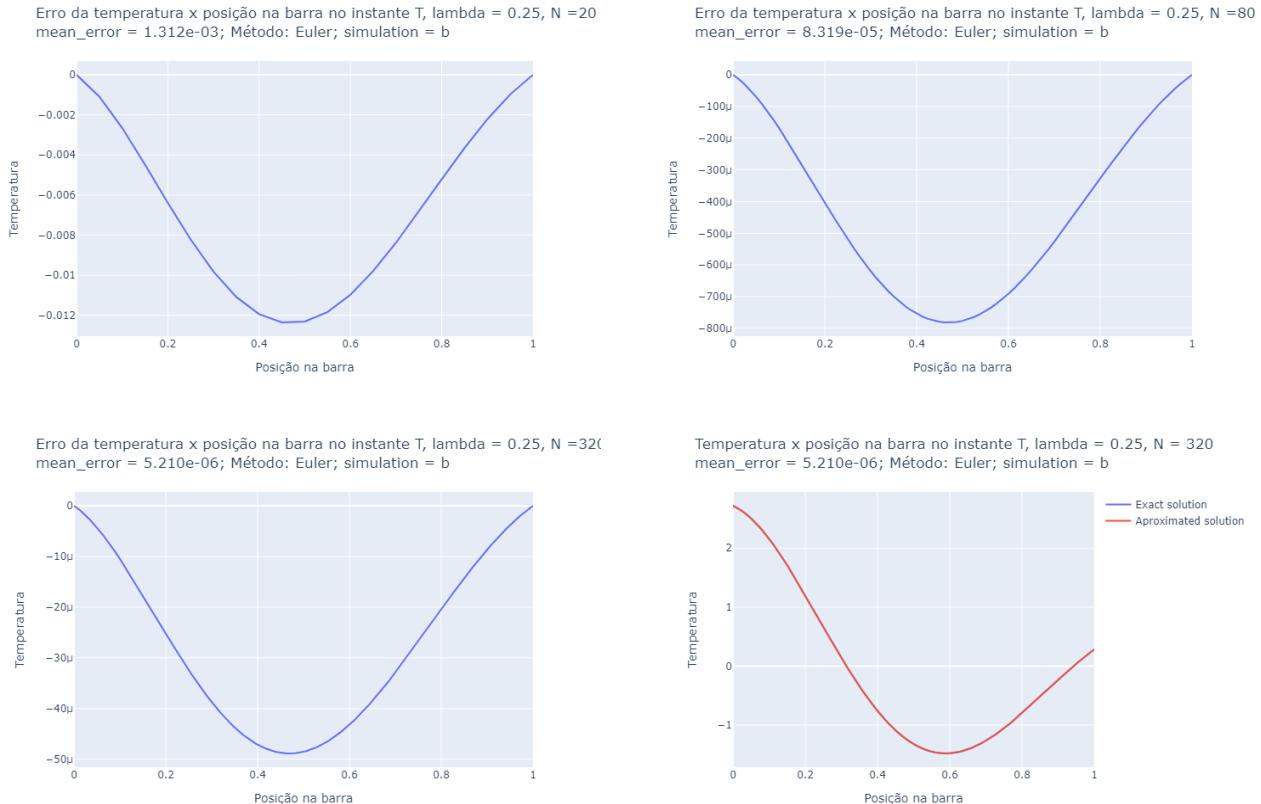
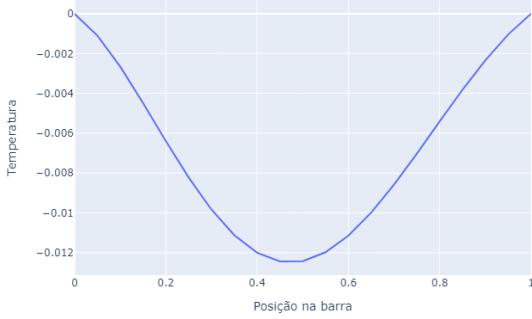
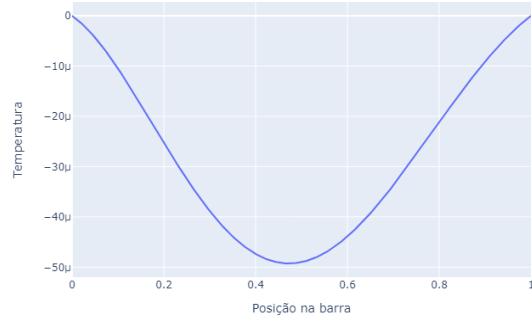


Figura 47: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.25$, método de Euler

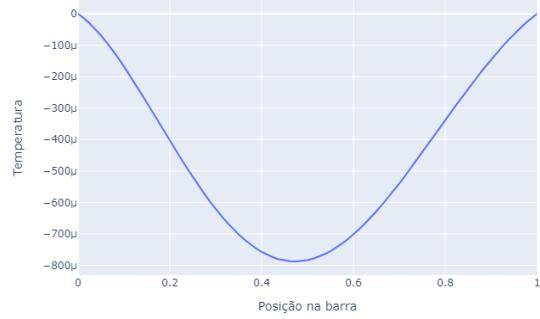
Erro da temperatura x posição na barra no instante T, lambda = 0.5, N =20
mean_error = 1.381e-03; Método: Euler; simulation = b



Erro da temperatura x posição na barra no instante T, lambda = 0.5, N =320
mean_error = 5.28e-06; Método: Euler; simulation = b



Erro da temperatura x posição na barra no instante T, lambda = 0.5, N =80
mean_error = 8.495e-05; Método: Euler; simulation = b



Temperatura x posição na barra no instante T, lambda = 0.5, N = 320
mean_error = 5.28e-06; Método: Euler; simulation = b

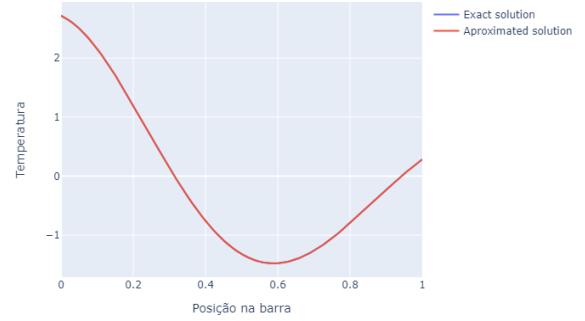


Figura 48: Distribuição do erro, no instante final, para diversos refinamentos de malha, com $\lambda = 0.5$, método de Euler

Nas figuras, 49, 50 e 51, podemos começar a ver as limitações deste método, onde para $\lambda > 0.50$, ele não converge para a solução. Para N pequenos, a solução não se distancia de forma significativa da solução correta, uma vez que não existem passos suficientes para que as instabilidades cresçam.

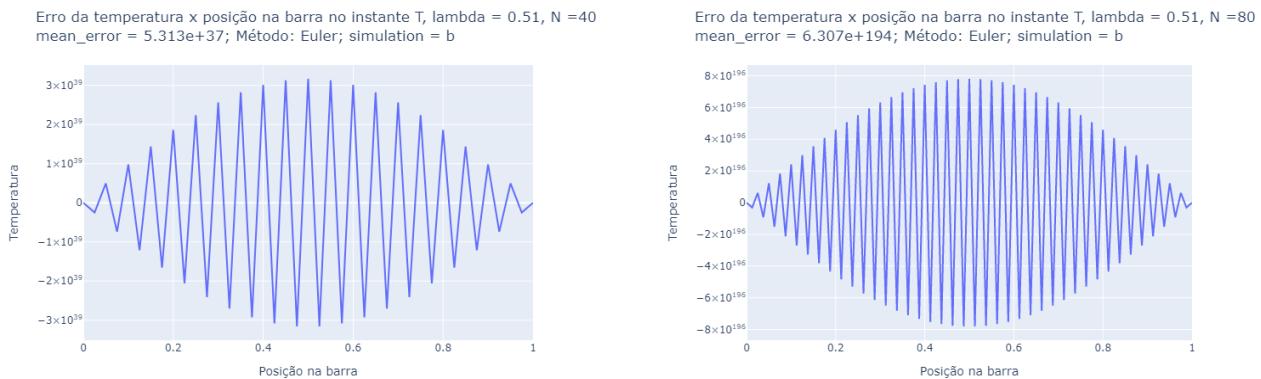
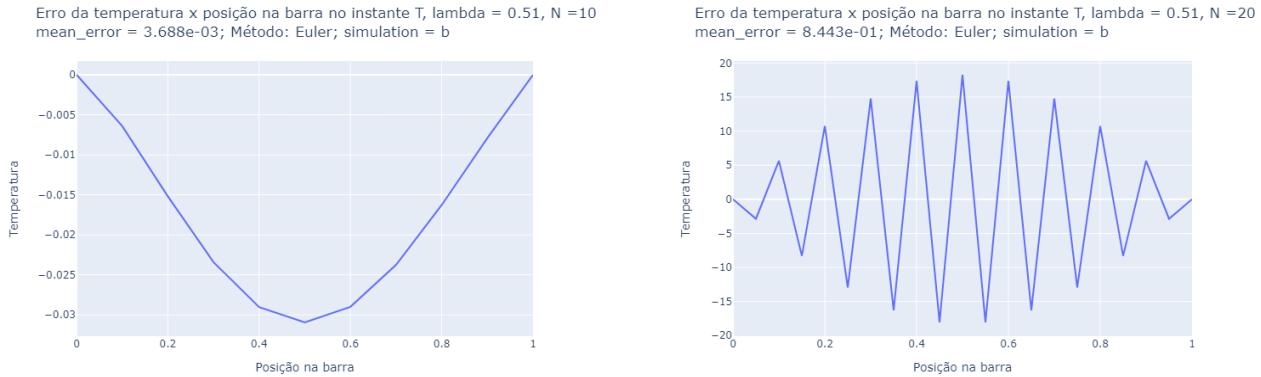


Figura 49: Distribuição do erro, no instante final, para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

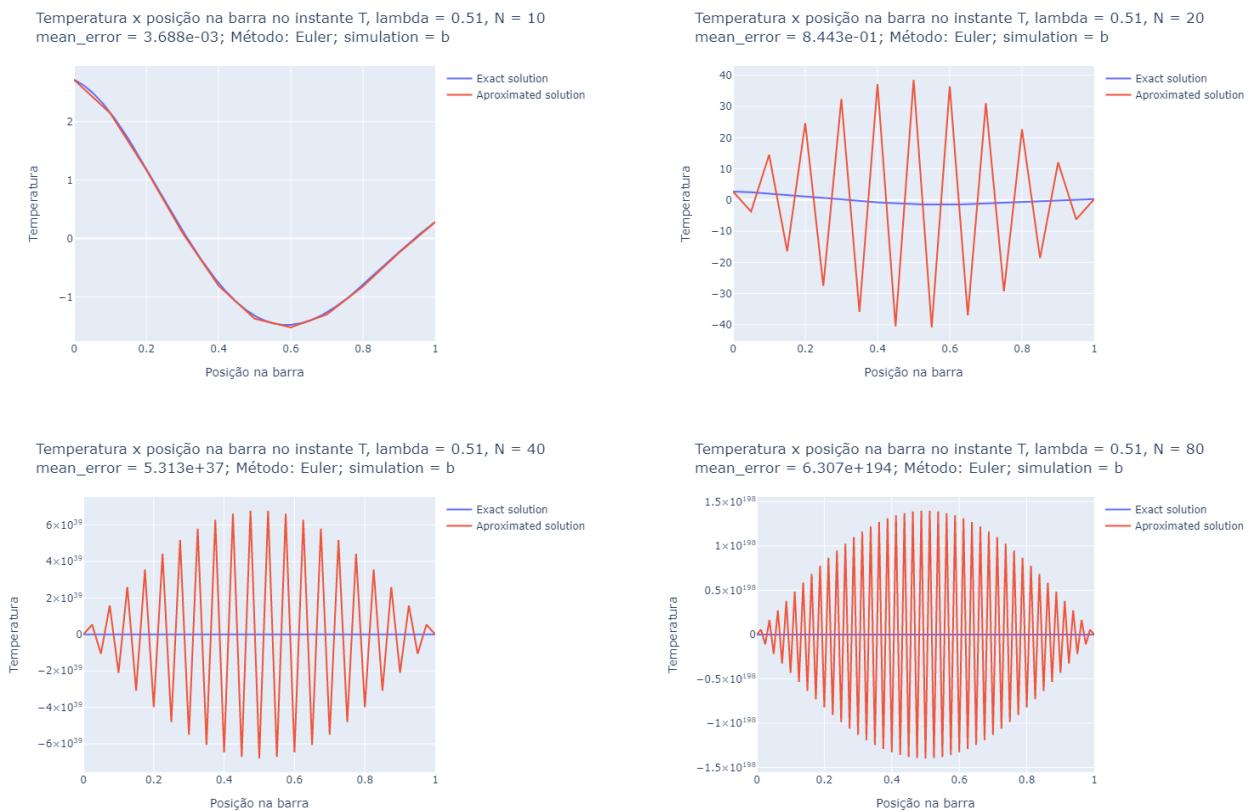


Figura 50: Estado final para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

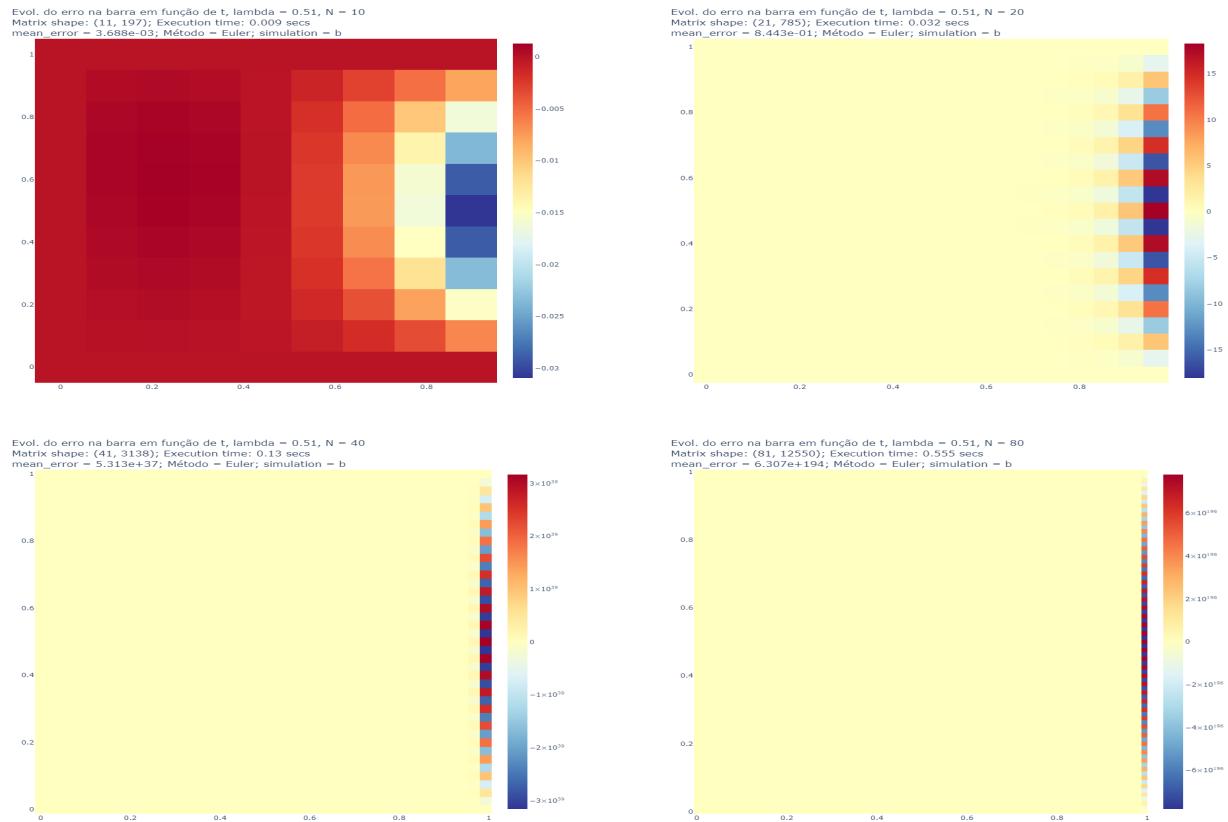


Figura 51: Distribuição do erro para diversos refinamentos de malha, com $\lambda = 0.51$, método de Euler

7.2.2 Euler implícito

Podemos ver nas figuras, 52 e 53 como o erro se desenvolve nas simulações em diversos refinamentos de malha.

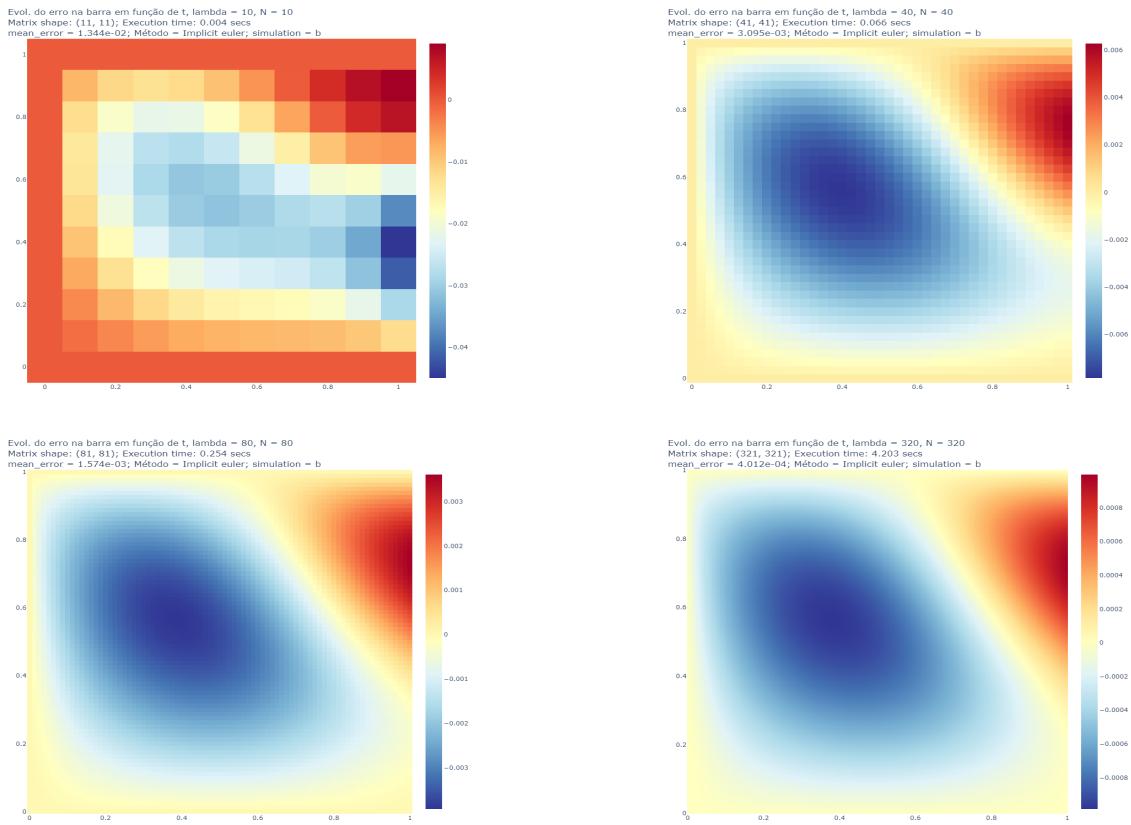


Figura 52: Distribuição do erro para diversos refinamentos de malha, método de Euler implícito

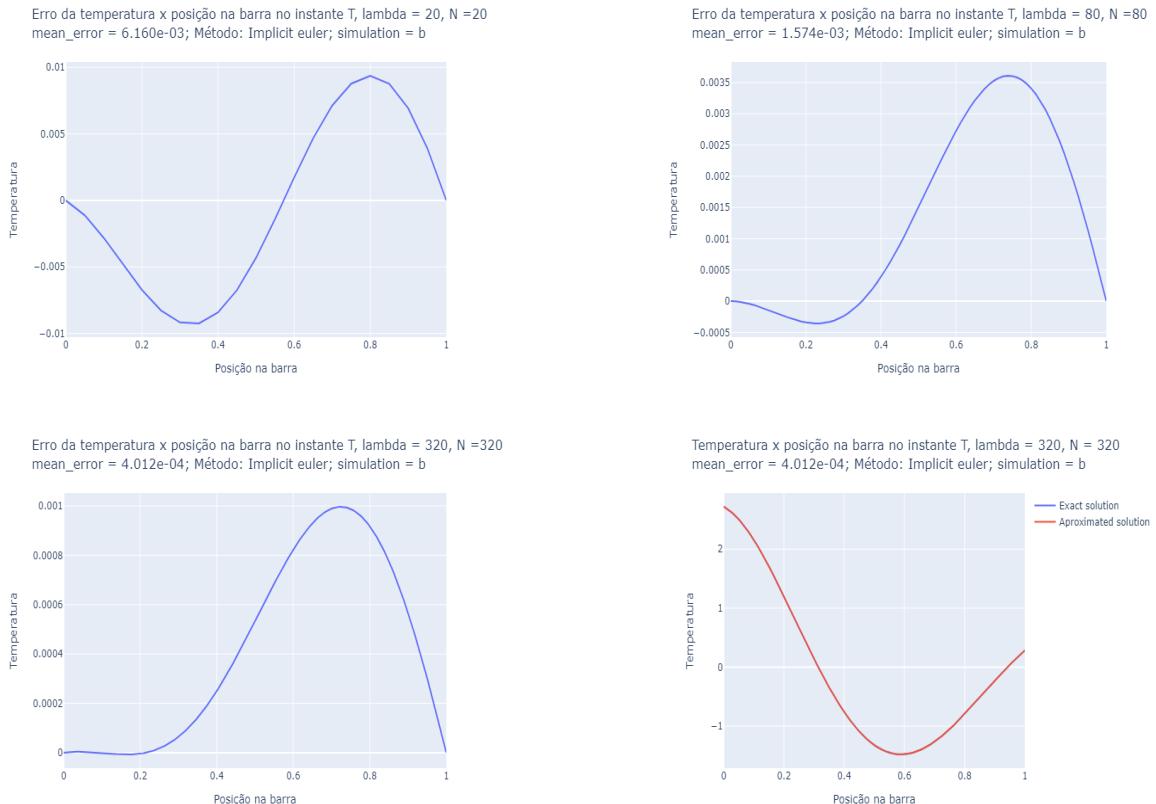


Figura 53: Distribuição do erro, no instante final, para diversos refinamentos de malha, método de Euler implícito

7.2.3 Crank-Nicolson

Podemos ver nas figuras, 54 e 55 como o erro se desenvolve nas simulações em diversos refinamentos de malha.

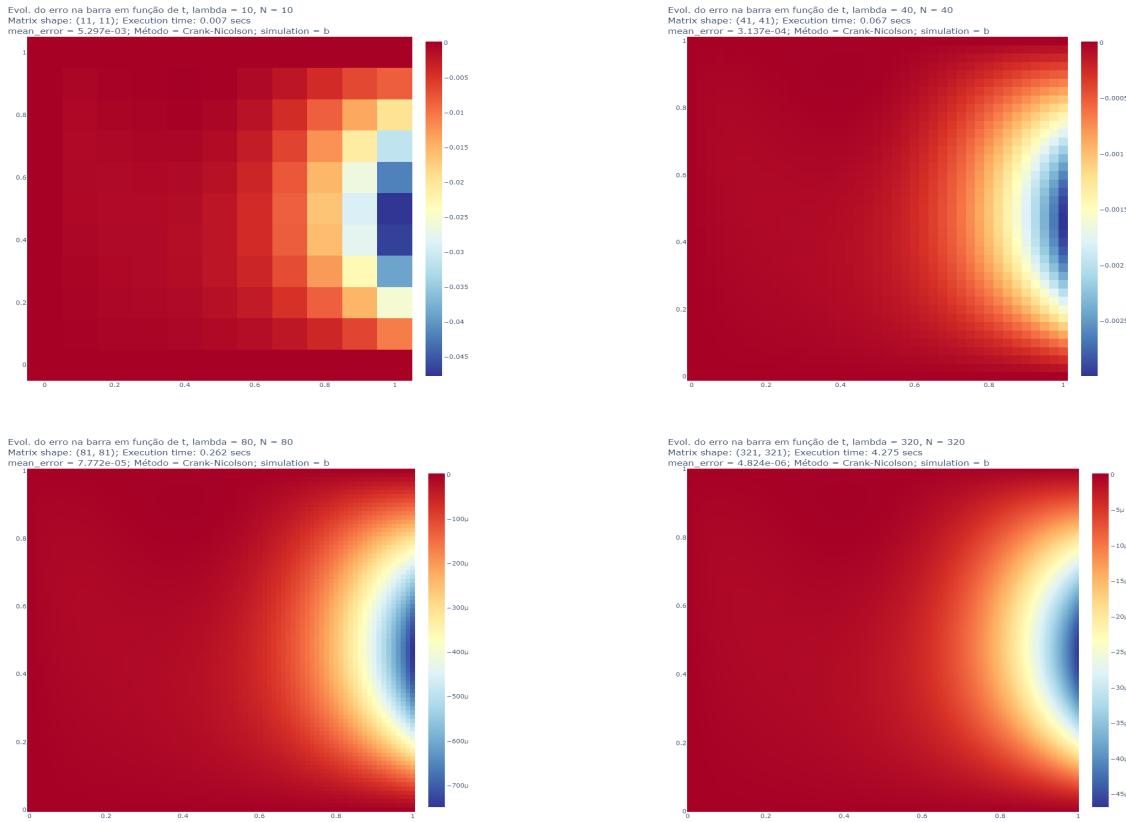
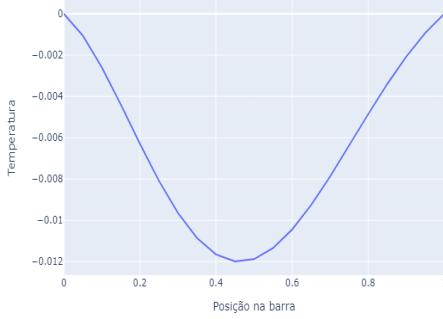
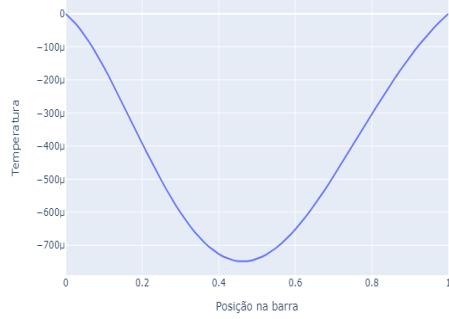


Figura 54: Distribuição do erro para diversos refinamentos de malha, método de Crank-Nicolson

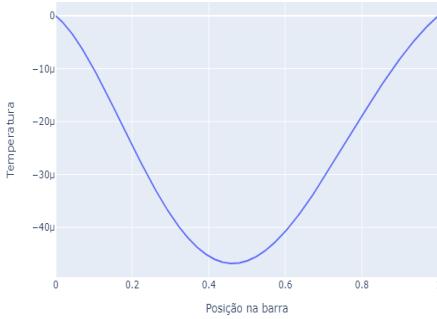
Erro da temperatura x posição na barra no instante T, lambda = 20, N =20
mean_error = 1.278e-03; Método: Crank-Nicolson; simulation = b



Erro da temperatura x posição na barra no instante T, lambda = 80, N =80
mean_error = 7.772e-05; Método: Crank-Nicolson; simulation = b



Erro da temperatura x posição na barra no instante T, lambda = 320, N =320
mean_error = 4.824e-06; Método: Crank-Nicolson; simulation = b



Temperatura x posição na barra no instante T, lambda = 320, N = 320
mean_error = 4.824e-06; Método: Crank-Nicolson; simulation = b

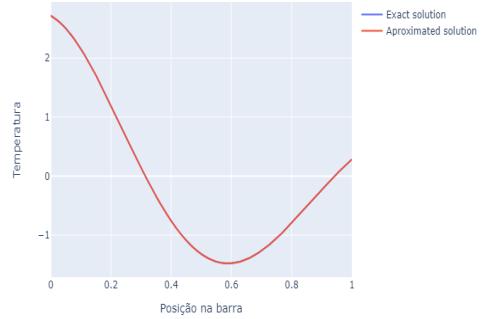


Figura 55: Distribuição do erro, no instante final, para diversos refinamentos de malha, método de Crank-Nicolson

7.3 Conclusões finais e análises

Neste cenário, podemos chegar a algumas conclusões muito importantes. Embora o método de Euler tenha boas aproximações, ele é muito custoso computacionalmente. Entre os métodos implícitos, ambos apresentam custos similares, e inferiores ao método de Euler. Vale notar, que o método de Implícito de Euler, para a mesma discretização, apresenta resultados inferiores ao método de Crank-Nicolson. E o de Crank-Nicolson, resultados semelhantes ao de Euler.

7.3.1 Euler

Podemos ver, na figura 56, que esse método possui uma complexidade de $O(N^3)$, fazendo o número de pontos a serem determinados a crescer de forma muito rápida, e por conseguinte, o tempo e os recursos necessários.

Number of calculations (matrix size) in function of N for each lambda
Método: Euler; simulation = b

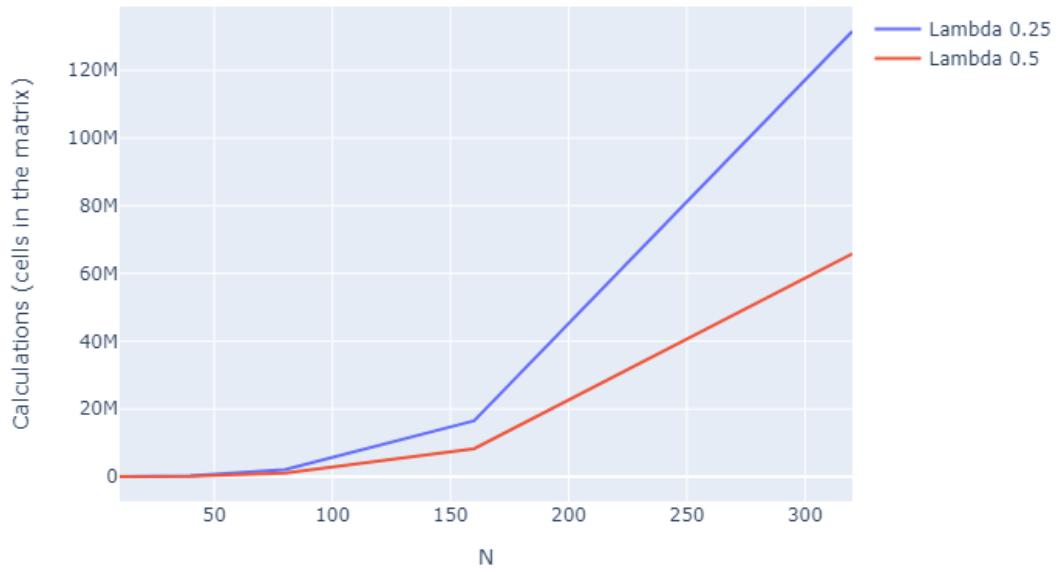


Figura 56: Número de cálculos em função de N

Analizando o erro, nas figuras, 57 e 58, podemos notar também, que, neste método, o λ não influência na precisão do método. Apenas a discretização no domínio do espaço, desde que a restrição de $\lambda \leq 0.5$, seja mantida. Logo, discretizações maiores que isso, são desperdício computacional. Para ambos os λ 's, o fator de redução é: 0.25.

Error in function of lambda for each N
Método: Euler; simulation = b

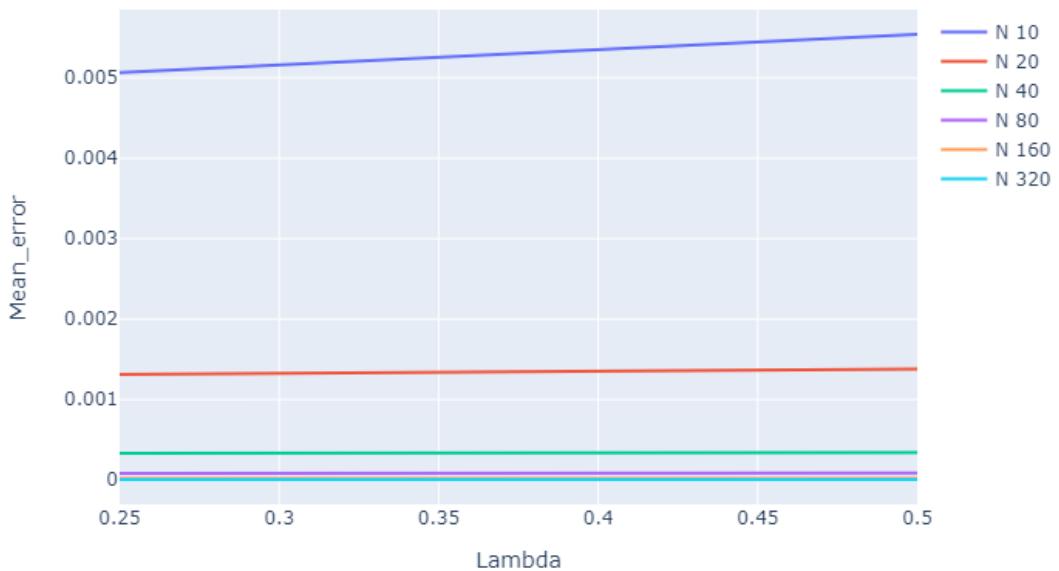


Figura 57: Erro em função de λ , para cada N

Error in function of N for each lambda
Método: Euler; simulation = b

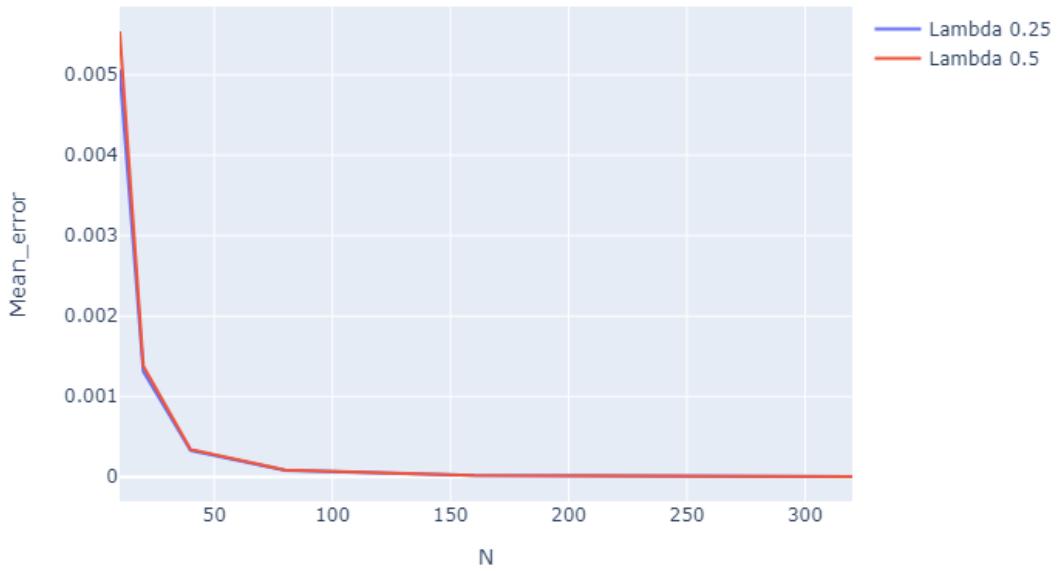


Figura 58: Erro em função de N, para cada λ

7.3.2 Euler implícito

Podemos ver, na figura 59, que esse método possui uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método anterior, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

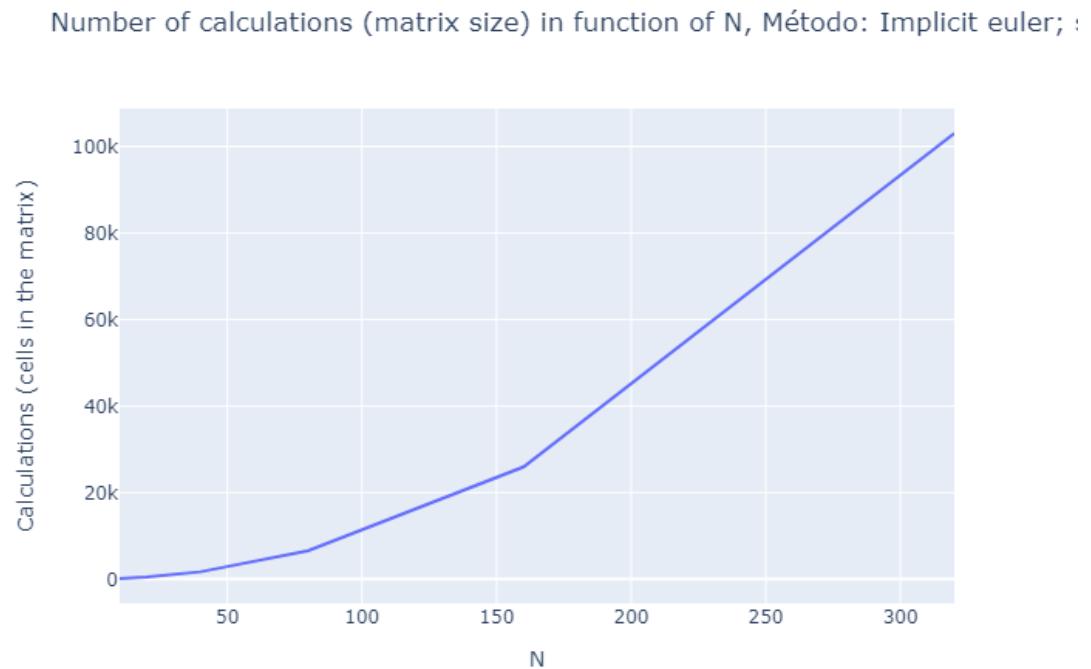


Figura 59: Número de cálculos em função de N

Analizando o erro, na figura 60, podemos notar também, que o mesmo diminui com o refinamento da malha. Para este método, o fator de redução é: 0.50.

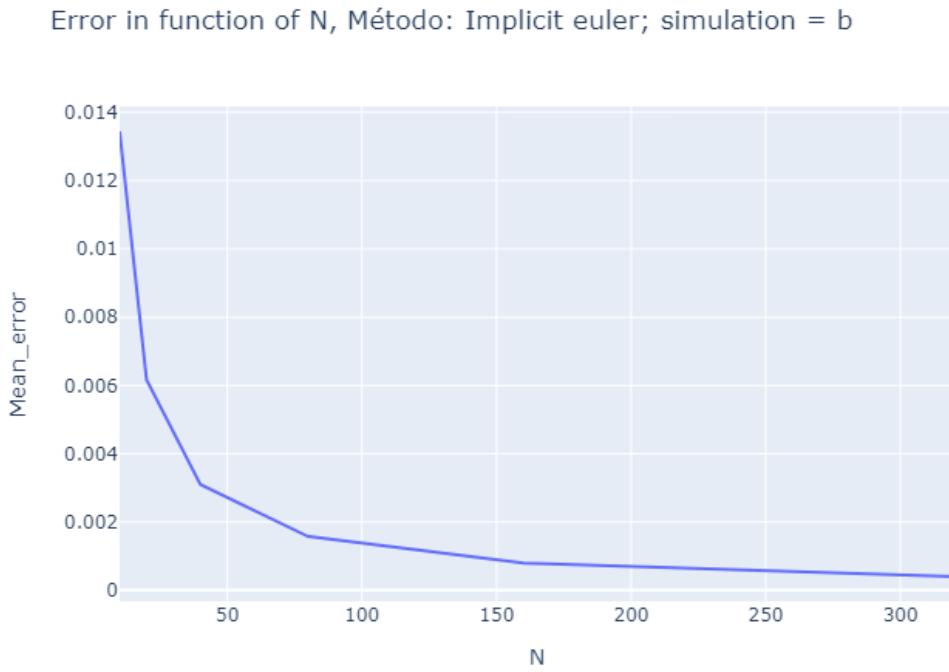


Figura 60: Erro em função de função de N

7.3.3 Crank-Nicolson

Podemos ver, na figura 61, que esse método possui também uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método de Euler, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

Number of calculations (matrix size) in function of N, Método: Crank-Nicolson

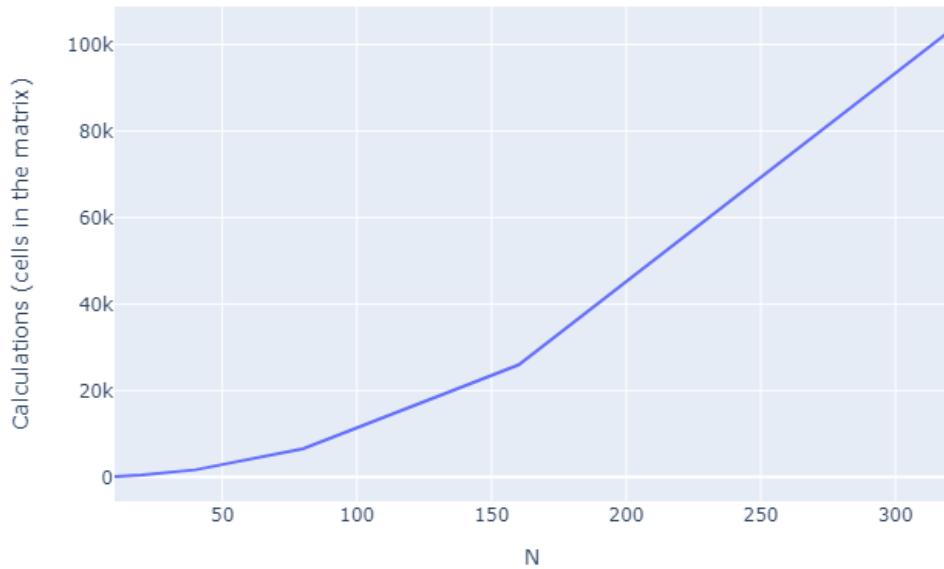


Figura 61: Número de cálculos em função de N

Analizando o erro, na figura 62, podemos notar também, que o mesmo aumenta com o refinamento da malha. Para este método, o fator de redução é: 0.25.

Error in function of N, Método: Crank-Nicolson; simulation = b

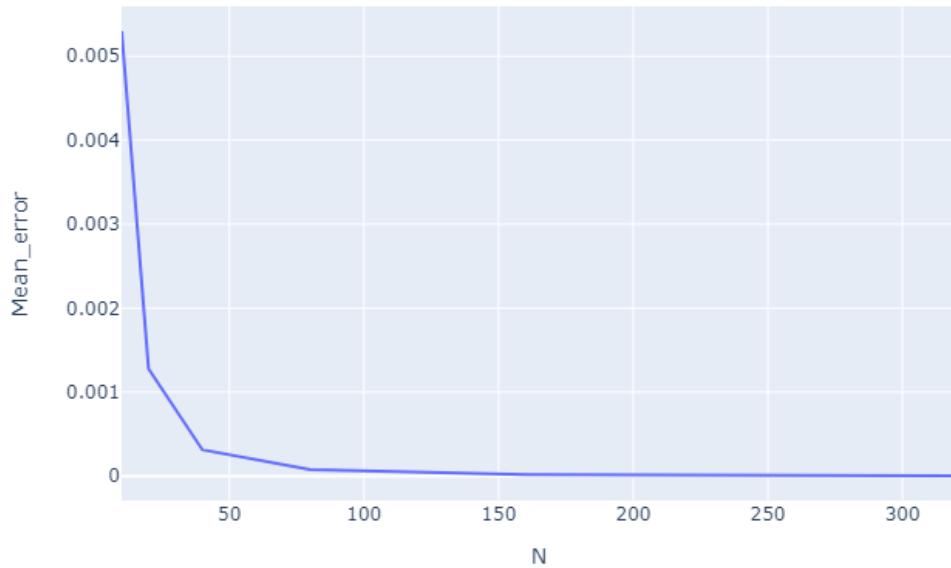


Figura 62: Erro em função de N

8 Problema C

As equações que descrevem esse problema são:

$$g_1(t) = 0$$

$$g_2(t) = 0$$

$$f(t, x, \Delta x) = 10000(1 - 2t^2) * gh(x, \Delta x)$$

$$gh(x, \Delta x) = \begin{cases} \frac{1}{\Delta x}, & \text{se } x \in [0.25 - \frac{\Delta x}{2} - 10^{-7}, 0.25 + \frac{\Delta x}{2} + 10^{-7}] \\ 0, & \text{caso contrário} \end{cases}$$

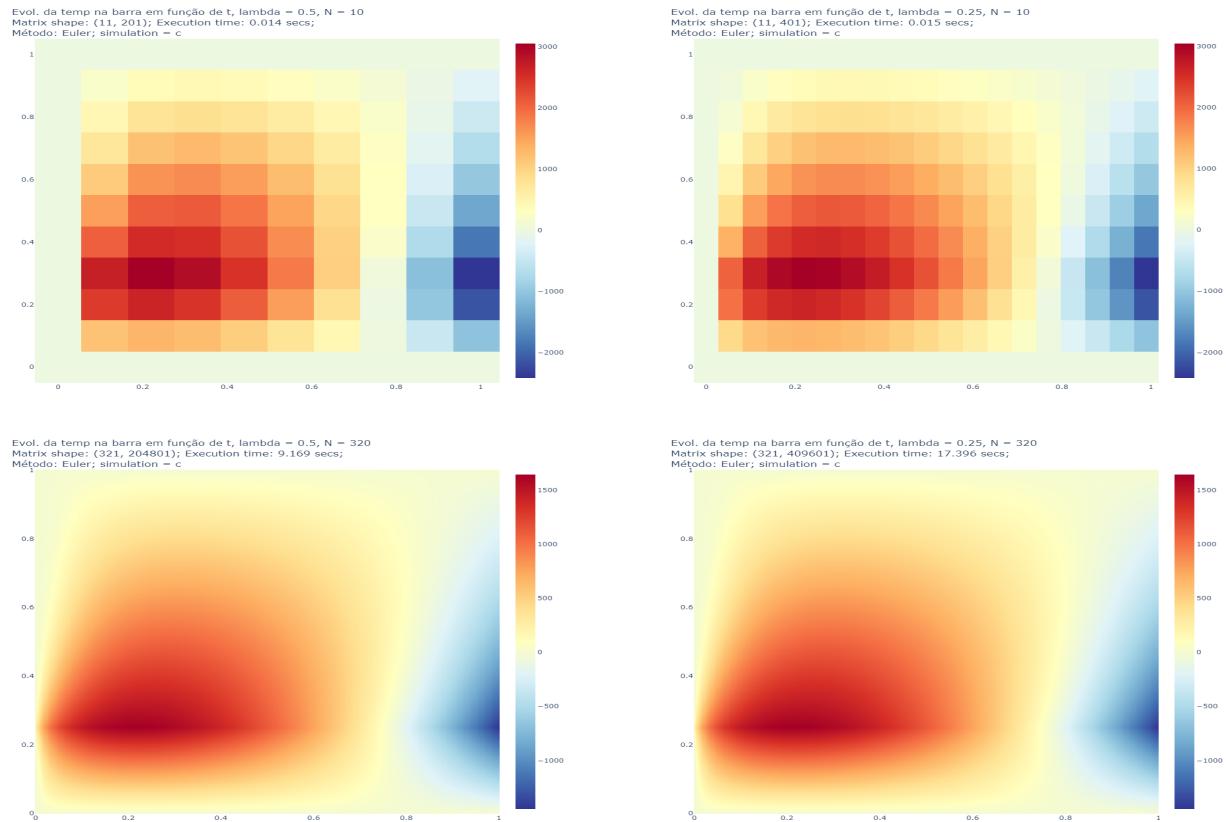
$$u_0(x) = e^{-x}$$

8.1 Analise de resultados

8.1.1 Euler

Podemos ver, na figura 63 a solução calculada numericamente em ambas as resoluções para ambos λ .

¹O alargamento no intervalo em $2 * 10^{-7}$ é necessário, pois no cenário em que x cai exatamente na fronteira do intervalo, devido as limitações de um *float 64*, esse ponto é considerado fora do intervalo. O número 10^{-7} foi escolhido pois é menor que Δx , porém é maior que o erro de truncamento de números do tipo *float 64*



Evol. da temp na barra em função de t, lambda = 0.5, N = 320
Método: Euler, simulation = c

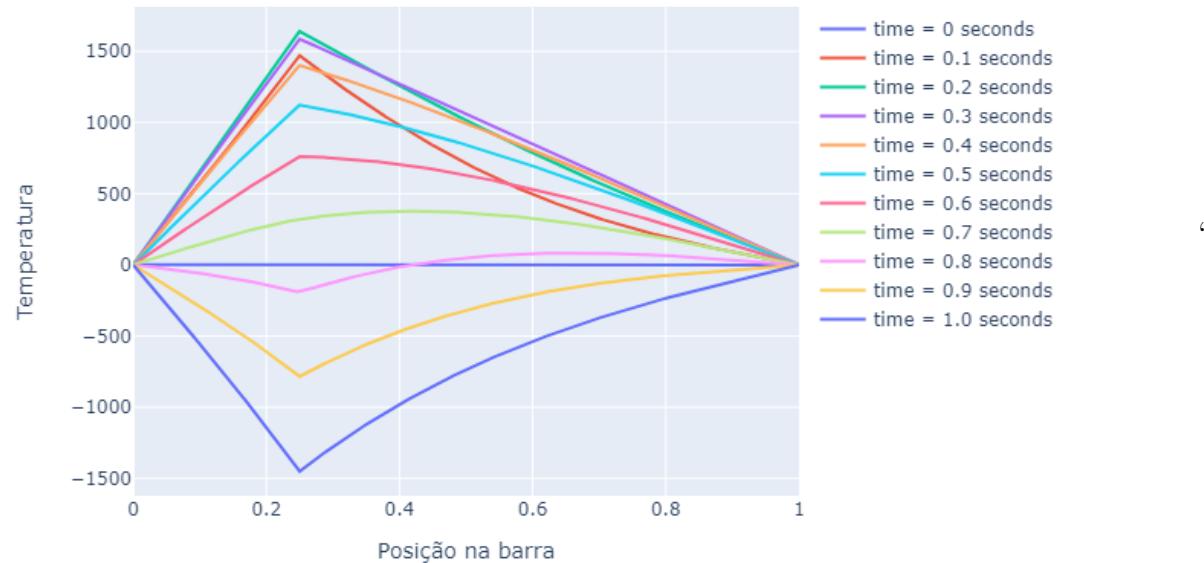
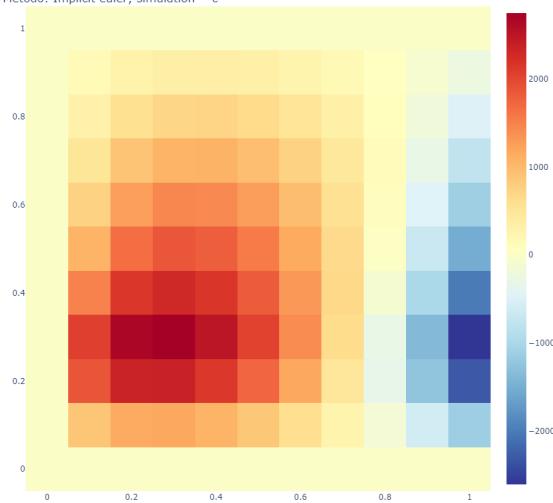


Figura 63: Podemos ver a evolução da temperatura na barra em diversos cenários, usando o método de Euler, com $\lambda = 0.5$ a esquerda, e $\lambda = 0.25$ a direita.

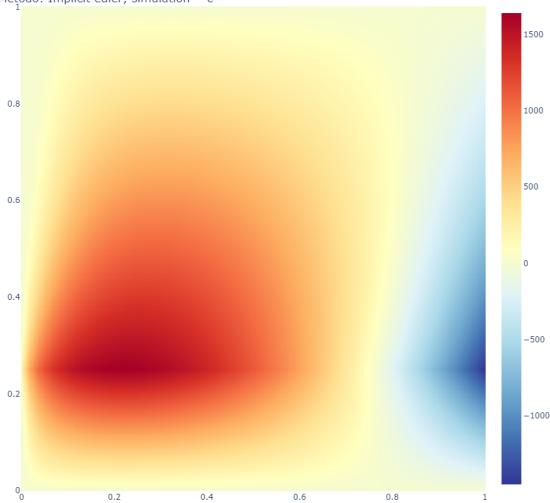
8.1.2 Euler implícito

Podemos ver, na figura 64, a solução calculada numericamente em ambas as resoluções, vale notar, que o tempo de execução e o custo computacional para tal solução, foi extremamente reduzido, se comparado com o método de Euler.

Evol. da temp na barra em função de t, lambda = 10, N = 10
Matrix shape: (11, 11); Execution time: 0.006 secs;
Método: Implicit euler; simulation = c



Evol. da temp na barra em função de t, lambda = 320, N = 320
Matrix shape: (321, 321); Execution time: 4.3 secs;
Método: Implicit euler; simulation = c



Evol. da temp na barra em função de t, lambda = 320, N = 320
Método: Implicit euler, simulation = c

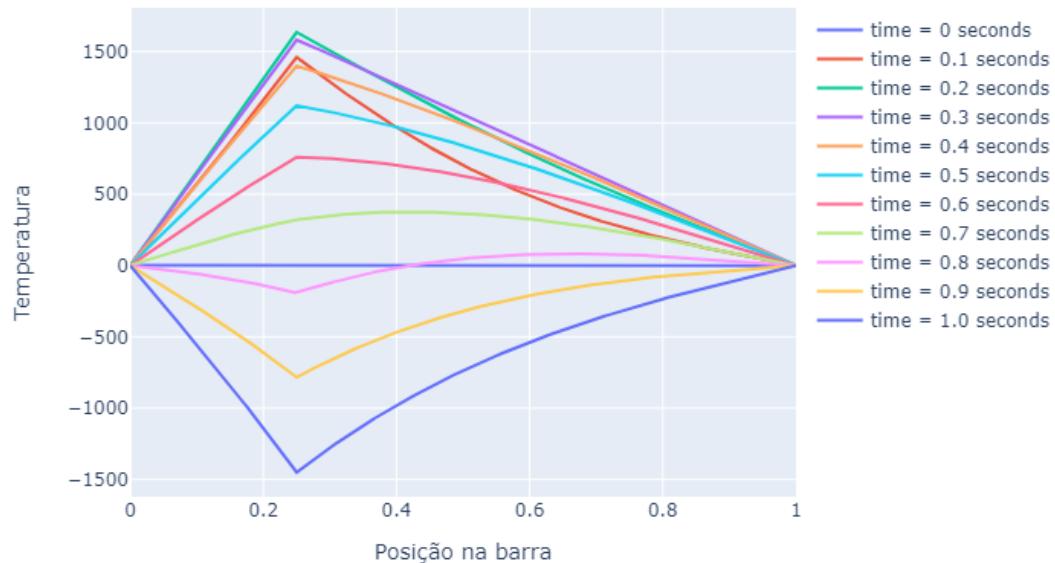
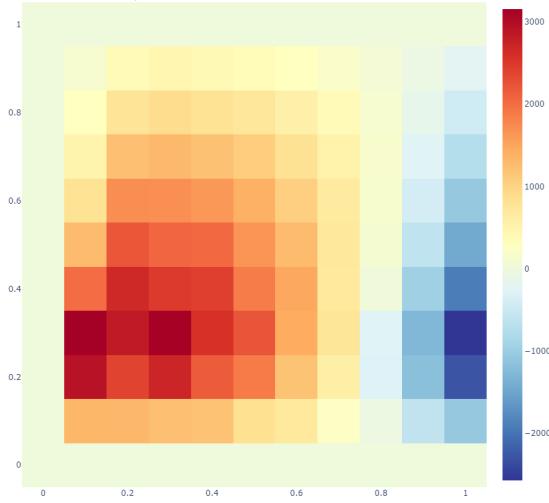


Figura 64: Resultados pelo método de Euler implícito

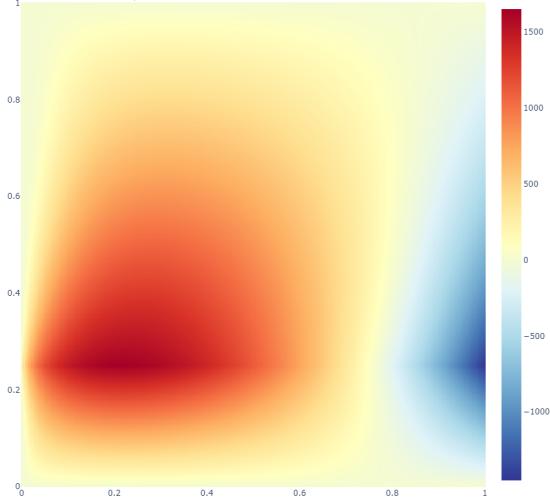
8.1.3 Crank-Nicolson

Podemos ver, na figura 65 a solução calculada numericamente em ambas as resoluções, o desempenho deste método neste cenário, mostrou-se muito semelhante ao do método implícito de Euler.

Evol. da temp na barra em função de t, lambda = 10, N = 10
Matrix shape: (11, 11); Execution time: 0.005 secs;
Método: Crank-Nicolson; simulation = c



Evol. da temp na barra em função de t, lambda = 320, N = 320
Matrix shape: (321, 321); Execution time: 4.063 secs;
Método: Crank-Nicolson; simulation = c



Evol. da temp na barra em função de t, lambda = 320, N = 320
Método: Crank-Nicolson, simulation = c

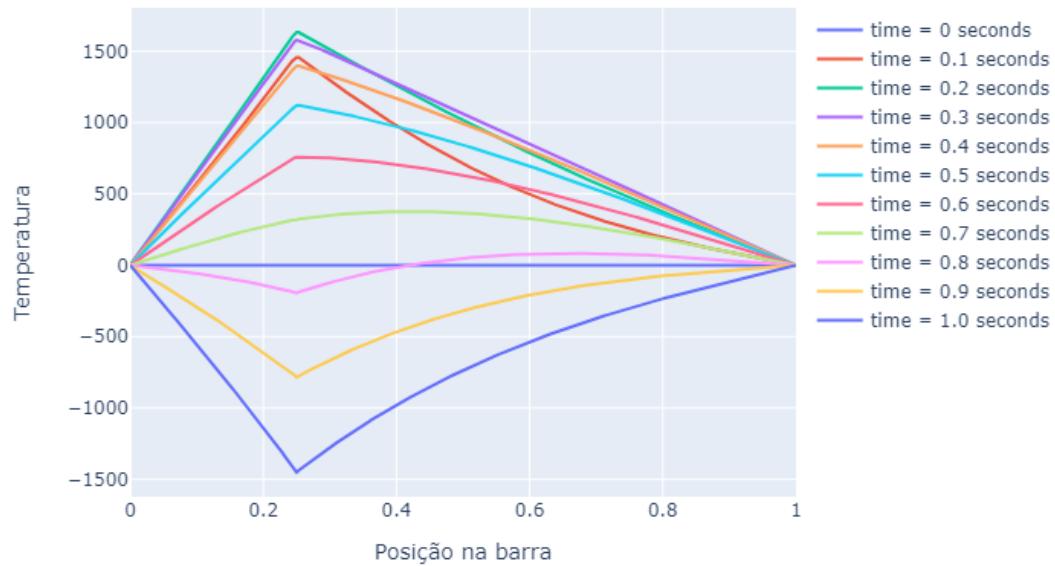


Figura 65: Resultados pelo método de Crank-Nicolson

8.2 Analise de erro

Para este problema, não teremos analise de erro, uma vez que não possuímos a solução exata para tal comparação.

8.3 Conclusões finais e analyses

Neste cenário, podemos chegar a algumas conclusões muito importantes. O método de Euler é muito custoso computacionalmente. Entre os métodos implícitos, ambos apresentam custos similares, e inferiores ao método de Euler.

8.3.1 Euler

Podemos ver, na figura 66, que esse método possui uma complexidade de $O(N^3)$, fazendo o número de pontos a serem determinados a crescer de forma muito rápida, e por conseguinte, o tempo e os recursos necessários.

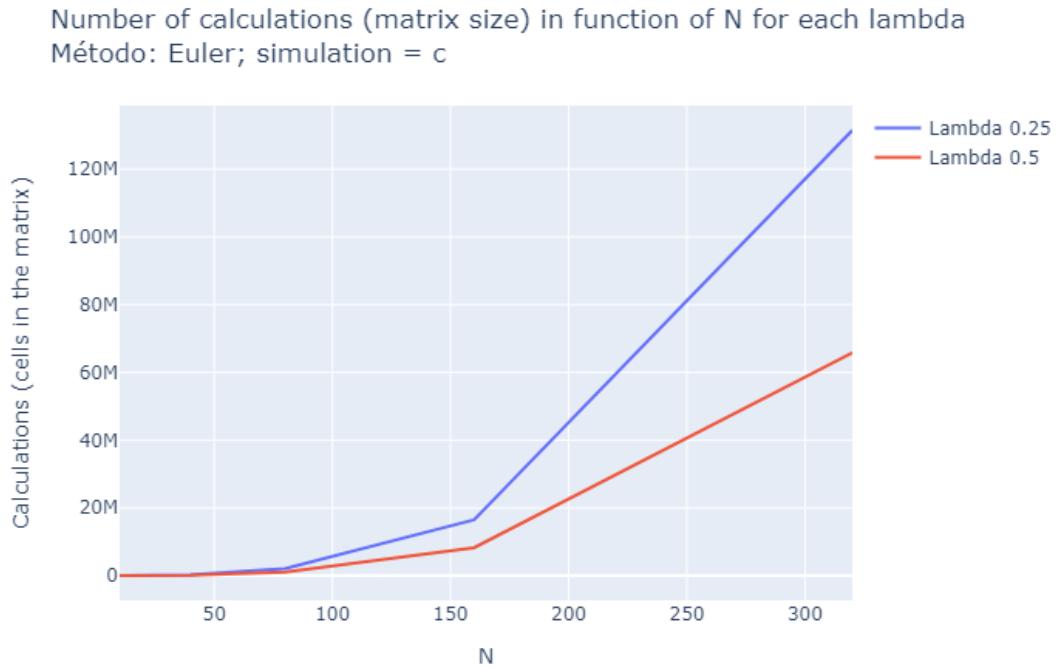


Figura 66: Número de cálculos em função de N

8.3.2 Euler implícito

Podemos ver, na figura 67, que esse método possui uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método anterior, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

Number of calculations (matrix size) in function of N, Método: Implicit euler; :

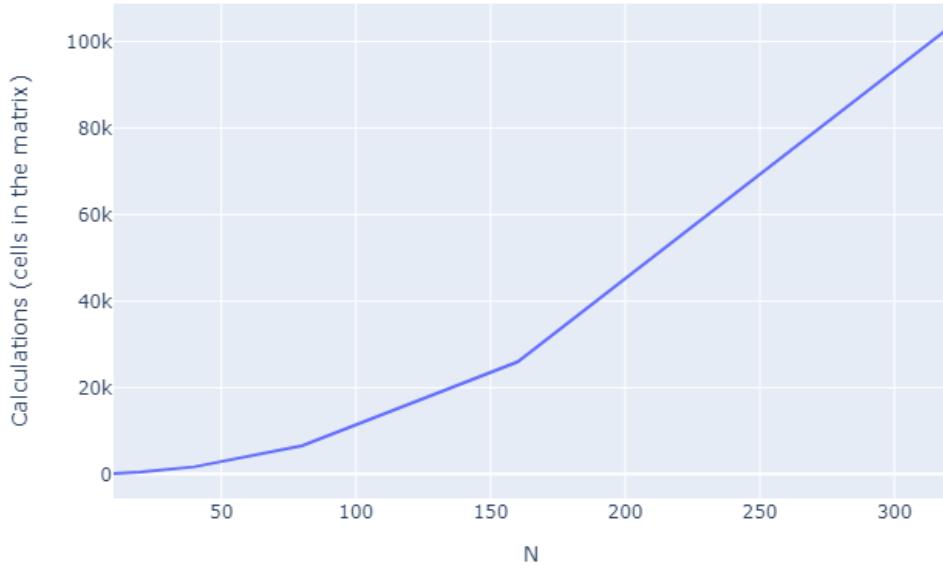


Figura 67: Número de cálculos em função de função de N

8.3.3 Crank-Nicolson

Podemos ver, na figura 68, que esse método possui também uma complexidade de $O(N^2)$, fazendo o número de pontos a serem determinados a crescer de forma muito mais lenta que no cenário do método de Euler, e por conseguinte, o tempo e os recursos necessários também são diminuídos. Nos cenários mais extremos, essa é uma diminuição por um fator de 1000.

Number of calculations (matrix size) in function of N, Método: Crank-Nicolson

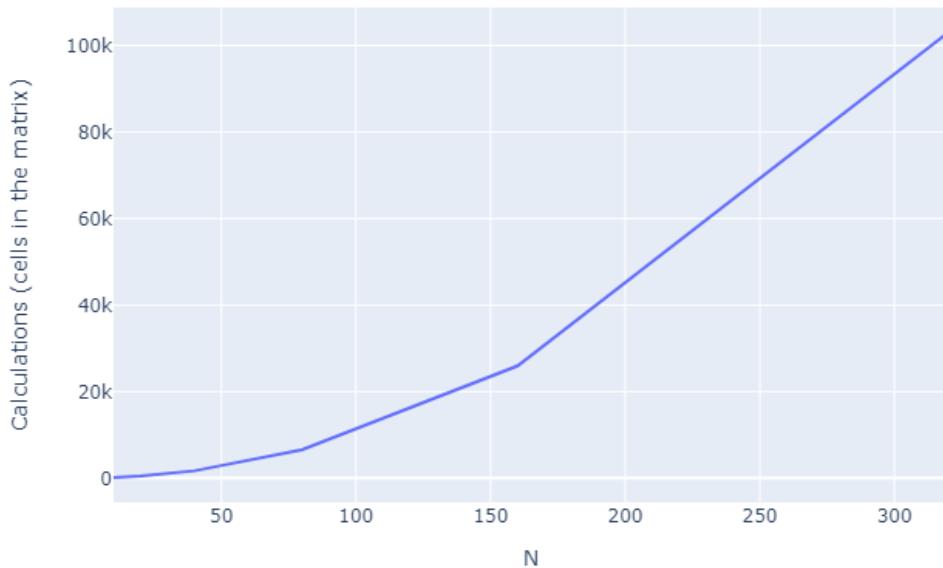


Figura 68: Número de cálculos em função de função de N

9 Conclusões finais

Ao implementarmos tais métodos numéricos, pudemos aprender que o melhor e mais apropriado é o método de Crank-Nicolson. Que devido a transformações LDL^t pode ser muito eficiente computacionalmente, e apresenta soluções mais precisas que os outros métodos, sem ser mais custoso que nenhum deles.

Os métodos de Euler e Euler implícito, esbarram ou em custo computacional ou em precisão, não sendo capazes de aliar ambos, como o método de Crank-Nicolson é.

Vale ressaltar também a grande evolução da dupla, em relação ao conhecimento sobre como computadores e programação funcionam. Na busca por eficiência, esses tópicos surgiram, e a evolução da dupla foi enorme. Como prova disso, tivemos um ganho de eficiência de cerca de 225x. Processos que demoravam em torno de uma hora, passaram a demorar meros 16 segundos. Sem mudança alguma ao hardware.

10 Apêndice

10.1 Gráficos não utilizados

Todos os 634 gráficos e tabelas de dados, inclusive os não utilizados na confecção deste relatório, estão na pasta 'Data', junto nesse mesmo .zip.

10.2 Código fonte

O código fonte encontrasse no arquivo *ep.py*, junto deste .zip.

10.3 Dados consolidados

Embora, os dados relativos a cada método em cada cenário possam ser encontrados individualmente dentro da pasta */Data*, um arquivo consolidado, chamado *consolidado.xlsx*, encontrasse junto nesse mesmo .zip.