

Cálculo Numérico - Relatório EP1

João Rodrigo Windisch Olenscki
NUSP 10773224

Luca Rodrigues Miguel
NUSP 10705655

Maio, 2020

Conteúdo

1	Introdução	2
2	Metodologia	2
2.1	Primeira tarefa	2
2.2	Segunda Tarefa	2
3	Desenvolvimento matemático	3
3.1	Primeira Tarefa	3
3.2	Segunda Tarefa	4
3.2.1	Método de Euler Implícito	5
3.2.2	Método de Crank-Nicolson	5
3.3	Fator de Redução e Ordem de Convergência	6
4	Estruturação do Código	7
5	Tarefas	8
5.1	1-a)	8
5.2	1-b)	9
6	Resultados	9
6.1	Função a)	9
6.2	Função b)	24
6.3	Função c)	36
6.4	Fator de redução e Ordem de Convergência	40
6.4.1	Método de Euler	40
6.4.2	Método de Euler Implícito	42
6.4.3	Método de Crank-Nicolson	44
7	Conclusões e considerações finais	45

1 Introdução

Este exercício consistiu na resolução da equação discreta do calor utilizando os métodos de Euler, Euler Implícito e Crank-Nicolson. Para tal, utilizando o **Python**, elaborou e escreveu-se um algoritmo que realizava as tarefas determinadas pelo enunciado.

O programa, resultado final do estudo dos alunos sobre o tema encontra-se em anexo no arquivo **.zip** no qual está contido este relatório.

Quanto ao relatório em si, o mesmo encontra-se dividido em algumas partes: a Seção 2 discorre sobre a metodologia adotada para a resolução do problema; já a Seção 3 apresenta os fundamentos matemáticos aplicados e que auxiliaram na sintetização do código; a Seção 4 apresenta uma breve explicação sobre a organização do código, bem como um pequeno exemplo do caminho que uma chamada da função de testes faz até que atinja o método iterativo; a Seção 5 resolve tarefas requisitadas pelo enunciado; a Seção 6 apresenta os gráficos dos diferentes métodos e testes e discute seus comportamentos; a Seção 7 finaliza o trabalho, revendo alguns conceitos-chave e tomando conclusões sobre os testes feitos.

2 Metodologia

2.1 Primeira tarefa

A primeira tarefa consiste na implementação e análise da equação referenciada como (11) no enunciado do EP: a equação da difusão de calor aproximada pela fórmula de diferenças finitas.

Como parâmetros de entrada para o programa para esta parte têm-se T (constante de tempo, representa o tempo total da análise), N (constante relacionada com o número de divisões que realizamos do espaço, $\Delta x = \frac{1}{N}$) e λ (constante que relaciona os passos de espaço Δx com os passos de tempo Δt , com relação igual a $\lambda = \frac{\Delta t}{\Delta x^2}$).

Dadas estas entradas, pode-se, a partir de λ , encontrar o parâmetro M . A partir deste parâmetro, cria-se uma matriz nula de ordem $(M + 1, N + 1)$, utilizada para armazenar os valores de temperatura calculados pela equação (11), a qual referencia-se neste relatório como **matriz de temperaturas**. Nesta, cada linha k representa o instante $\frac{kT}{M}$ de tempo e cada coluna i , a posição $\frac{i}{N}$ na barra.

Em seguida, calcula-se os vetores das condições iniciais e aplicou-se na **matriz de temperaturas**. A partir do desenvolvimento matemático aplicado a equação (11), é possível transformar a operação, antes escalar, em vetorial, o que possibilita a execução do código em tempo hábil.

Para o cálculo da *array* de erro para o instante $T = 1$ retira-se da **matriz de temperaturas** sua última linha e da **matriz exata** o mesmo. Em seguida, subtrai-se uma da outra, calculando-se assim a **array de erro** associada ao método.

2.2 Segunda Tarefa

A segunda tarefa consiste em aplicar a mesma rotina anteriormente descrita para os métodos implícitos de Euler e Crank-Nicolson. A diferença primordial entre esta e a tarefa anterior é que agora o valor de λ não mais tem efeito, ou seja, M depende somente de N pela relação $M = N$, de forma que os instantes discretizados passam a ser $\frac{kT}{N}$ para uma linha k . Isso também significa que, para o caso em que $T = 1$, a matriz de temperaturas é quadrada, de dimensões $(N + 1, N + 1)$.

3 Desenvolvimento matemático

3.1 Primeira Tarefa

Para o desenvolvimento matemático, é interessante definir com antecedência a **matriz de temperaturas**, a **matriz exata**, a **matriz erro**

$$M = \begin{bmatrix} g_1(0) = u_0(0) & u_0(1) & u_0(2) & \cdots & u_0(N-1) & u_0(N) = g_2(0) \\ g_1(1) & u_1^1 & u_2^1 & \cdots & u_{N-1}^1 & g_2(1) \\ g_1(2) & u_1^2 & u_2^2 & \cdots & u_{N-1}^2 & g_2(2) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ g_1(M) & u_1^M & u_2^M & \cdots & u_{N-1}^M & g_2(M) \end{bmatrix}$$

$$E = \begin{bmatrix} e(0,0) & e(0,1) & e(0,2) & \cdots & e(0,N-1) & e(0,N) \\ e(1,0) & e(1,1) & e(1,2) & \cdots & e(1,N-1) & e(1,N) \\ e(2,0) & e(2,1) & e(2,2) & \cdots & e(2,N-1) & e(2,N) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ e(M,0) & e(M,1) & e(M,2) & \cdots & e(M,N-1) & e(M,N) \end{bmatrix}$$

Onde M é a **matriz de temperaturas**, em que cada linha representa um tempo e cada coluna, uma posição, e E a **matriz exata**, sendo $e(k, i)$ o valor do resultado da função exata de distribuição. A **matriz erro** é definida como $\Delta = M - E$. Podemos também escrever (e renumerar) a equação (11) contida no enunciado:

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

Reorganizando esta equação e substituindo $\lambda = \frac{\Delta t}{\Delta x^2}$, chegamos em:

$$u_i^{k+1} = (1 - 2\lambda)u_i^k + \lambda(u_{i-1}^k + u_{i+1}^k) + \Delta t f(t_k, x_i) \quad (2)$$

Assim, é possível observar que as temperaturas de qualquer linha $k + 1$ somente dependem do valor registrado na linha anterior k (mesmo que em três posições diferentes) e do resultado da função f calculado nesta linha, de forma que isto torna possível vetorizar esta operação. Obtendo-se assim que:

$$\begin{bmatrix} 0 & u_{1\dots N-1}^{k+1} & 0 \end{bmatrix} = \begin{bmatrix} u_0^k & \dots & u_N^k \end{bmatrix} \cdot \begin{bmatrix} 0 & \lambda & 0 & \cdots & 0 & 0 \\ 0 & 1 - 2\lambda & \lambda & \cdots & 0 & 0 \\ 0 & \lambda & 1 - 2\lambda & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 - 2\lambda & 0 \\ 0 & 0 & 0 & \cdots & \lambda & 0 \end{bmatrix} \quad (3)$$

$$+ \Delta t [0 \quad f_{1\dots N-1}^k \quad 0]$$

Na equação (3), as matrizes da forma $[x_{1\dots N-1}^i]$ representam matrizes-linha que referenciam os elementos de 1 a $N - 1$ da i -ésima linha da matriz u . A matriz tridiagonal de índices $a_i = c_i = \lambda$ e $b_i = 1 - 2\lambda$, a qual nos referenciaremos como **A**, tem ordem $(N + 1) \times (N + 1)$. A primeira e a última colunas são nulas devido ao fato de que u_0^k e u_N^k serem iguais a, respectivamente, $g_1(k)$ e $g_2(k)$. Além disso, a matriz $[0 \ f_{1\dots N-1}^K \ 0]$ representa uma array de saídas da função $f(t_k, x_i)$ com k fixado e i variando no intervalo $1\dots N - 1$.

3.2 Segunda Tarefa

Nesta segunda tarefa, a primeira atividade requisitada é realizar a seguinte decomposição $\mathbf{A} = \mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^t$. Um adendo que esta matriz \mathbf{A} não é a mesma apresentada na primeira parte, sendo diferente em sinal e em dimensões. Assim, é conveniente aplicar a multiplicação em etapas, que podem ser representadas pelos parênteses na seguinte equação: $\mathbf{A} = (\mathbf{L} \cdot (\mathbf{D} \cdot \mathbf{L}^t))$.

$$\begin{aligned}
D \cdot L^t &= \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & d_3 & & \\ & & & \ddots & \\ & & & & d_{N-2} & d_{N-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & l_2 & & & \\ & 1 & l_3 & & \\ & & 1 & l_4 & \\ & & & \ddots & \ddots \\ & & & & 1 & l_{N-1} \\ & & & & & 1 \end{bmatrix} \\
&= \begin{bmatrix} d_1 & d_1 \cdot l_2 & & & \\ & d_2 & d_2 \cdot l_3 & & \\ & & d_3 & d_3 \cdot l_4 & \\ & & & \ddots & \ddots \\ & & & & d_{N-2} & d_{N-2} \cdot l_{N-1} \\ & & & & & d_{N-1} \end{bmatrix} \\
L \cdot D \cdot L^t &= \begin{bmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & l_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & l_{N-2} & 1 \\ & & & & l_{N-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} d_1 & d_1 \cdot l_2 & & & \\ & d_2 & d_2 \cdot l_3 & & \\ & & d_3 & d_3 \cdot l_4 & \\ & & & \ddots & \ddots \\ & & & & d_{N-2} & d_{N-2} \cdot l_{N-1} \\ & & & & & d_{N-1} \end{bmatrix} \\
&= \begin{bmatrix} d_1 & d_1 \cdot l_2 & & & \\ d_1 \cdot l_2 & d_1 \cdot l_2^2 + d_2 & d_2 \cdot l_3 & & \\ & d_2 \cdot l_3 & d_2 \cdot l_3^2 + d_3 & d_3 \cdot l_4 & \\ & & \ddots & \ddots & \ddots \\ & & & d_{N-2} \cdot l_{N-1} & d_{N-3} \cdot l_{N-2}^2 + d_{N-2} \\ & & & & d_{N-2} \cdot l_{N-1} & d_{N-2} \cdot l_{N-1}^2 + d_{N-1} \end{bmatrix}
\end{aligned}$$

Dado que \mathbf{A} , definido por uma *array* \mathbf{a} dos valores da diagonal principal e por outra \mathbf{b} da subdiagonal, sendo ela igual a $\mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^t$, compara-se ambas termo a termo, gerando o seguinte sistema:

$$\begin{cases} d_1 &= a_1 \\ d_i \cdot l_{i+1} &= b_{i+1} \quad \forall i = 1, \dots, N-2 \\ b_{i+1} \cdot l_{i+1} + d_{i+1} &= a_{i+1} \quad \forall i = 1, \dots, N-2 \end{cases}$$

Esta equação se resolve de forma recursiva: a partir de $d_1 = a_1$, é possível calcular $l_2 = \frac{b_2}{a_1}$. Com l_2 obtém-se d_2 a partir de $d_2 = a_2 - b_2 \cdot l_2$ e assim sucessivamente.

Desta forma, define-se propriamente as matrizes \mathbf{L} e \mathbf{D} e possibilita-se realizar a resolução do sistema linear $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ na forma de $\mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^t \cdot \mathbf{x} = \mathbf{b}$. Sendo que este deve ser desenvolvido por partes $\mathbf{L} \cdot (\mathbf{D} \cdot (\mathbf{L}^t \cdot \mathbf{x})) = \mathbf{b}$.

$$\mathbf{L}^t \cdot \mathbf{x} = \begin{bmatrix} 1 & l_2 & & & \\ & 1 & l_3 & & \\ & & 1 & l_4 & \\ & & & \ddots & \ddots \\ & & & & 1 & l_{N-1} \\ & & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} x_1 + l_2 \cdot x_2 \\ x_2 + l_3 \cdot x_3 \\ x_3 + l_4 \cdot x_4 \\ \vdots \\ x_{N-2} + l_{N-1} \cdot x_{N-1} \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{bmatrix}$$

$$\mathbf{D} \cdot \mathbf{L}^t \cdot \mathbf{x} = \begin{bmatrix} d_1 & & & & & \\ & d_2 & & & & \\ & & d_3 & & & \\ & & & \ddots & & \\ & & & & d_{N-2} & \\ & & & & & d_{N-1} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} d_1 \cdot y_1 \\ d_2 \cdot y_2 \\ d_3 \cdot y_3 \\ \vdots \\ d_{N-2} \cdot y_{N-2} \\ d_{N-1} \cdot y_{N-1} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{N-2} \\ z_{N-1} \end{bmatrix}$$

$$\mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^t \cdot \mathbf{x} = \begin{bmatrix} 1 & & & & & \\ l_2 & 1 & & & & \\ l_3 & & 1 & & & \\ \ddots & \ddots & & & & \\ & & l_{N-2} & 1 & & \\ & & & l_{N-1} & 1 & \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{N-2} \\ z_{N-1} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 + z_1 \cdot l_2 \\ z_3 + z_2 \cdot l_3 \\ \vdots \\ z_{N-2} + z_{N-3} \cdot l_{N-2} \\ z_{N-1} + z_{N-2} \cdot l_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{N-2} \\ b_{N-1} \end{bmatrix} \quad (4)$$

A resolução deste sistema é implementada calculando os vetores \mathbf{z} , \mathbf{y} e \mathbf{x} nesta ordem. Ou seja, dado que possui-se os vetores \mathbf{b} e \mathbf{l} de antemão, é fácil definir um sistema linear que relate \mathbf{z} a ambos. Em seguida, realiza-se o mesmo procedimento para achar \mathbf{y} a partir do recém encontrado \mathbf{z} e \mathbf{d} . Por fim, aplica-se o mesmo raciocínio para encontrar-se \mathbf{x} , o que resolve o problema.

3.2.1 Método de Euler Implícito

O método de Euler implícito é definido segundo a seguinte equação matricial:

$$\begin{bmatrix} 1+2\lambda & -\lambda & 0 & \dots & 0 & 0 \\ -\lambda & 1+2\lambda & -\lambda & \dots & 0 & 0 \\ 0 & -\lambda & 1+2\lambda & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1+2\lambda & -\lambda \\ 0 & 0 & 0 & \dots & -\lambda & 1+2\lambda \end{bmatrix} \cdot \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ u_3^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \begin{bmatrix} u_1^k + \Delta t f_1^{k+1} + \lambda g_1(t_{k+1}) \\ u_2^k + \Delta t f_2^{k+1} \\ u_3^k + \Delta t f_3^{k+1} \\ \vdots \\ u_{N-2}^k + \Delta t f_{N-2}^{k+1} \\ u_{N-1}^k + \Delta t f_{N-1}^{k+1} + \lambda g_2(t_{k+1}) \end{bmatrix} \quad (5)$$

Dado o resultado obtido na equação 4, este sistema iterativo é facilmente resolvível via o método de Cholesky. Outra informação importante e passível de análise é a de que o enunciado pede que $\Delta t = \Delta x$. Ou seja, $\frac{M}{T} = N$ (o que implica que $M = N$, uma vez que $T = 1$) e $\lambda = \frac{\Delta t}{\Delta x \Delta x} = \frac{1}{\Delta x} = N$

3.2.2 Método de Crank-Nicolson

O método de Crank-Nicolson é, também, um método **implícito**. Isto quer dizer que a estabilidade dele é alcançada por iterações seguidas de estimativas das variáveis locais da matriz. Sendo assim, ele é incondicionalmente estável [1], o que permite o uso de grandes passos nas iterações.

A equação de Crank-Nicolson é a seguinte:

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

A mesma pode ser re-escrita da seguinte forma:

$$-\frac{\lambda}{2} u_{i-1}^{k+1} + (1 + \lambda) u_i^{k+1} - \frac{\lambda}{2} u_{i+1}^{k+1} = \frac{\lambda}{2} u_{i-1}^k + (1 - \lambda) u_i^k + \frac{\lambda}{2} u_{i+1}^k + \frac{\Delta t}{2} (f(x_i, t_k) + f(x_i, t_{k+1})) \quad (7)$$

E, com esta equação (7), é possível observar o caráter matricial do método. Assim, segue que:

$$\begin{bmatrix}
1 + \lambda & -\frac{\lambda}{2} & 0 & \dots & 0 & 0 \\
-\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} & \dots & 0 & 0 \\
0 & -\frac{\lambda}{2} & 1 + \lambda & \dots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \dots & 1 + \lambda & -\frac{\lambda}{2} \\
0 & 0 & 0 & \dots & -\frac{\lambda}{2} & 1 + \lambda
\end{bmatrix} \cdot \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ u_3^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \\
\begin{bmatrix}
(1 - \lambda)u_1^k + \frac{\lambda}{2}u_2^k + \frac{\lambda}{2}(g_1(t_k) + g_1(t_{k+1})) + \frac{\Delta t}{2}(f(x_1, t_k) + f(x_1, t_{k+1})) \\
\frac{\lambda}{2}u_1^k + (1 - \lambda)u_2^k + \frac{\lambda}{2}u_3^k + \frac{\Delta t}{2}(f(x_2, t_k) + f(x_2, t_{k+1})) \\
\frac{\lambda}{2}u_2^k + (1 - \lambda)u_3^k + \frac{\lambda}{2}u_4^k + \frac{\Delta t}{2}(f(x_3, t_k) + f(x_3, t_{k+1})) \\
\vdots \\
\frac{\lambda}{2}u_{N-3}^k + (1 - \lambda)u_{N-2}^k + \frac{\lambda}{2}u_{N-1}^k + \frac{\Delta t}{2}(f(x_{N-2}, t_k) + f(x_{N-2}, t_{k+1})) \\
\frac{\lambda}{2}u_{N-2}^k + (1 - \lambda)u_{N-1}^k + \frac{\lambda}{2}(g_2(t_k) + g_2(t_{k+1})) + \frac{\Delta t}{2}(f(x_{N-1}, t_k) + f(x_{N-1}, t_{k+1}))
\end{bmatrix} \quad (8)$$

3.3 Fator de Redução e Ordem de Convergência

A ordem de convergência de um método está diretamente relacionada a seu erro de truncamento. No Método de Euler, sendo C_1 e C_2 as constantes definidas no enunciado, têm-se que:

$$\|e^{k+1}\| \leq T(C_1\Delta t + C_2\Delta x^2)$$

Como $T = 1$ e $\Delta t = \lambda\Delta x^2$, vale que:

$$\|e^{k+1}\| \leq (C_1 + \lambda C_2)\Delta x^2$$

$$\|e^{k+1}\| \leq C\Delta x^2$$

Ou seja, o erro máximo é limitado por Δx^2 . O expoente de x é a ordem de convergência da solução, em Δx . No caso, o Método de Euler possui ordem 2 em Δx (assumindo $\lambda \leq 0.5$, ou seja, que o método é estável).

Para calcular a ordem de convergência μ do método a partir dos valores obtidos para o erro, por definição, faz-se a seguinte operação:

$$\mu = \frac{\log(e_{N'}/e_N)}{\log(\Delta x'/\Delta x)}$$

Onde $\Delta x' = 1/N'$, e N' e N são valores possíveis do número de discretizações da barra. Discretizações maiores aumentam a precisão dos resultados, retornando valores precisos de μ .

Para calcular o fator de redução ϕ , adota-se a seguinte operação:

$$\phi = \frac{e_{2N}}{e_N}$$

Note a seguinte relação:

$$\mu = \frac{\log \phi}{\log(1/2)} \Rightarrow \log \phi = \mu \log 1/2$$

$$\phi = 2^{-\mu} \quad (9)$$

4 Estruturação do Código

O código deste exercício-programa, como já citado anteriormente, foi redigido em **Python**. Ele foi estruturado de forma a separar ao máximo pequenas tarefas em funções auxiliares e pensando sempre em modularizar estas funções, de forma que uma única função redigida para uma tarefa específica pudesse ser utilizada em todos os testes e métodos requisitados.

Todas as funções do código foram comentadas e explicadas quando definidas, de forma a tornar simples a tarefa de entender o que cada uma faz, quais são os valores de entrada e quais os de saída. Um exemplo de comentário de função encontra-se abaixo:

```
def run_vectorized(T, lambda_val, N, f_function, exact = False, method = 'euler'):
    """
    funcao que define a rotina de execucao do programa para determinados valores de T, lambda_val e N
    @parameters:
        - T: float, constante de tempo T
        - lambda_val: float, constante do problema
        - N: inteiro, numero de divisoes feitas na barra
        - f_function: function, funcao f(x,t) para o teste
        - exact: bool, indicador se calcularemos a equacao exata ou a aproximacao
        -- default_value: False
        - method: string, representa qual o metodo empregado na resolucao da integracao numerica
        -- default_value: 'euler'
    @output:
        -~ M: inteiro, numero de divisoes no tempo
        - delta_time: float, tempo total decorrido para a execucao do programa, em segundos
        -~ time_array: array (1x(M+1)), contem todos os instantes de tempo
        -~ space_array: array (1x(N+1)), contem todas as posicoes da barra
        -~ u: array ((M+1)x(N+1)), matriz de temperaturas
        -~ exact_matrix: array ((M+1)x(N+1)), matriz de temperaturas calculada a partir da funcao exata
        - last_line: array, (1x(N+1)), array da ultima linha de uma matriz (u ou exact_matrix)
    """

Neste exemplo pode-se explicar como padronizaram-se os comentários. Depois da definição de uma certa função com seus parâmetros de entrada iniciou-se uma seção de comentário. As primeiras linhas desta seção são usadas para explicar o qual tarefa a função executa. Após esta sucinta explicação, existe uma seção denominada @parameters, esta seção explica o que são as variáveis de entrada da função, da seguinte forma:
```

```
"""
- var_name: tipo (com dimensão, se necessário), explicação da variável
-- default_value: valor padrão assumido pela variável na definição da função
-- example: esta sintaxe também pode ser utilizada para provir um exemplo de uma variável de entrada
-> comentário adicional sobre a seção
"""

Em seguida têm-se uma seção @output, que explica como são cada uma das saídas da função, inclusive se esta saída ocorre sempre ou somente em casos específicos:
```

```
"""
- var_name: tipo (com dimensão, se necessário), explicação da variável
-- example: um exemplo de uma variável de saída
-~ var_name: quando o retorno desta variável é condicionado a algo dentro do código
-> comentário adicional sobre a seção
"""

Tendo isto explicado, pode-se explicar a estruturação do código em si. Para tanto é necessário mostrar o caminho que uma chamada da função run_set_of_tests. Esta função recebe um valor de  $T$ , uma lista de valores
```

de λ , uma lista de valores de N , uma string de método e uma string de teste (a, b ou c). Esta função então chama outra, `create_folders`, responsável por criar pastas dentro do diretório para organizar a saída das figuras. Então, para cada valor de λ (se o método for o de Euler explícito) e para cada valor de N , é chamada a função `generate_plots`, que recebe como parâmetros quase iguais aos da função anterior, porém ao invés de uma lista de λ e N , passam-se valores para estas variáveis, e também um diretório (recém criado) onde as figuras deverão ser salvas.

A função `generate_plots`, por sua vez, chama outra de nome `run_vectorized` (cuja expressão de definição foi usada de exemplo no início desta Seção). Os outputs desta função são então utilizados para realizar os *plots* de fato (tanto as figuras dos intervalos definidos de tempo quanto o *heatmap*) e para calcular o erro no instante $T = 1$.

O processo iterativo, em si, é executado pela função `apply_estimated_solution`, chamada na `run_vectorized`, que recebe como parâmetros a constante T , um valor de λ , uma matriz u já com as condições de contorno aplicadas, uma `space_array`, que é um vetor do numpy que contém todos os valores de x considerados, uma `f_function`, função que define a fonte de calor utilizada e uma string de método, utilizada para diferenciar a iteração.

As instruções sobre como rodar o código para todos os testes ou para um teste específico encontram-se no arquivo LEIAME.txt anexado junto a este relatório,

5 Tarefas

5.1 1-a)

No primeiro item deste exercício, é requisitado a integração numérica a partir do método contido na equação (1) para $N = 10, 20, 40, 80, 160$ e 320 e $\lambda = 0.25$ e 0.5 com função de transferência $f(t, x) = 10 \cos(10t)x^2(1 - x)^2 - (1 + \sin(10t))(12x^2 - 12x + 2)$.

Verifica-se que $u(t, x) = (1 + \sin(10t))x^2(1 - x)^2$ é solução exata da equação

$$u_t = u_{xx} + f(t, x)$$

Onde $f(t, x) = 10 \cos(10t)x^2(1 - x)^2 - (1 + \sin(10t))(12x^2 - 12x + 2)$. Calculando u_t , obtém-se:

$$u_t = (1 + 10 \cos(10t))x^2(1 - x)^2 = 10 \cos(10t)(x^2 - 2x^3 + x^4)$$

E u_{xx} :

$$u_x = (1 + \sin(10t))(2x - 6x^2 + 4x^3)$$

$$u_{xx} = (1 + \sin(10t))(2 - 12x + 12x^2)$$

Assim, fazendo $u_{xx} + f(t, x)$:

$$\begin{aligned} u_{xx} + f(t, x) &= (1 + \sin(10t))(2 - 12x + 12x^2) + 10 \cos(10t)x^2(1 - x)^2 - \\ &\quad (1 + \sin(10t))(12x^2 - 12x + 2) \end{aligned}$$

$$u_{xx} + f(t, x) = 10 \cos(10t)x^2(1 - x)^2$$

Mas $u_t = 10 \cos(10t)x^2(1 - x)^2$. Logo:

$$u_t = u_{xx} + f(t, x)$$

E, portanto, $u(t, x)$ é solução da equação de calor.

É requisitado também calcular o número de passos deste método iterativo. Dado que o Método de Euler é calculado a partir de uma iteração matricial, onde o iterador é o número da linha, é possível afirmar que o código tem complexidade $\mathcal{O}(M)$, ou seja, $\mathcal{O}(N^2/\lambda)$, de forma que o número de passos necessários para $N = 640$ e $\lambda = 0.25$ é 1 638 400. É fácil notar, também, que conforme dobra-se o valor de N , quadruplica-se o número de passos, uma vez que o expoente de N na complexidade é igual a 2.

Diferentemente do caso anterior, os métodos implícitos possuem complexidades muito menores. Ambos continuam, de fato, a ter complexidade $\mathcal{O}(M)$, porém nestes casos, o valor de M não é mais o mesmo. Para estes métodos, $M = N$, e portanto $\mathcal{O}(M) = \mathcal{O}(N)$, e assim dobrar o valor de N somente dobra o valor de passos necessários na iteração

5.2 1-b)

Para encontrar $f(t, x)$, substitui-se a solução $u(x)$ na equação da transferência de calor:

$$u_t = u_{xx} + f(t, x)$$

Onde $u = e^{t-x} \cos(5tx)$. Calculando u_t e u_{xx} :

$$u_t = e^{t-x} \cos(5tx) - 5xe^{t-x} \sin(5tx)$$

$$u_{xx} = (1 - 25t^2)e^{t-x} \cos(5tx) + 10te^{t-x} \sin(5tx)$$

Logo $f(t, x) = u_t - u_{xx}$ e:

$$f(t, x) = [25t^2 \cos(5tx) - 5(x + 2t) \sin(5tx)]e^{t-x}$$

Já para encontrar as condições de contorno, substituem-se os valores na solução. Para $u_0(x)$, toma-se $t = 0$ em $u(t, x)$. O resultado é:

$$u_0(x) = e^{0-x} \cos(5x \cdot 0) \Rightarrow u_0(x) = e^{-x}$$

Para $g_1(t)$, fazemos $x = 0$ e encontramos:

$$g_1(t) = e^{t-0} \cos(5t \cdot 0) \Rightarrow g_1(t) = e^t$$

E, por fim, para encontrar $g_2(t)$, toma-se $x = 1$:

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

6 Resultados

Para cada função, testou-se os métodos de Euler, Euler Implícito e Crank-Nicolson, de forma que é possível comparar a eficácia e os erros gerados por cada método em cada situação específica.

6.1 Função a)

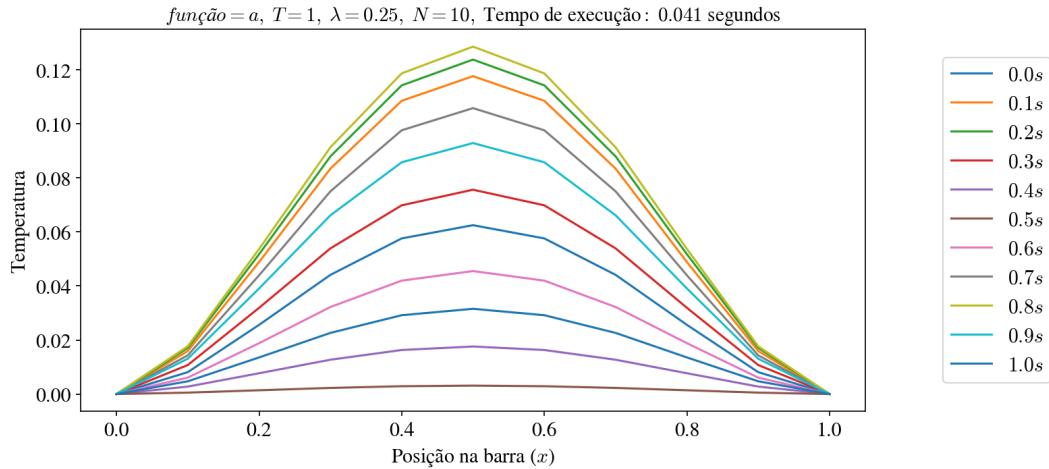
A função a) é:

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

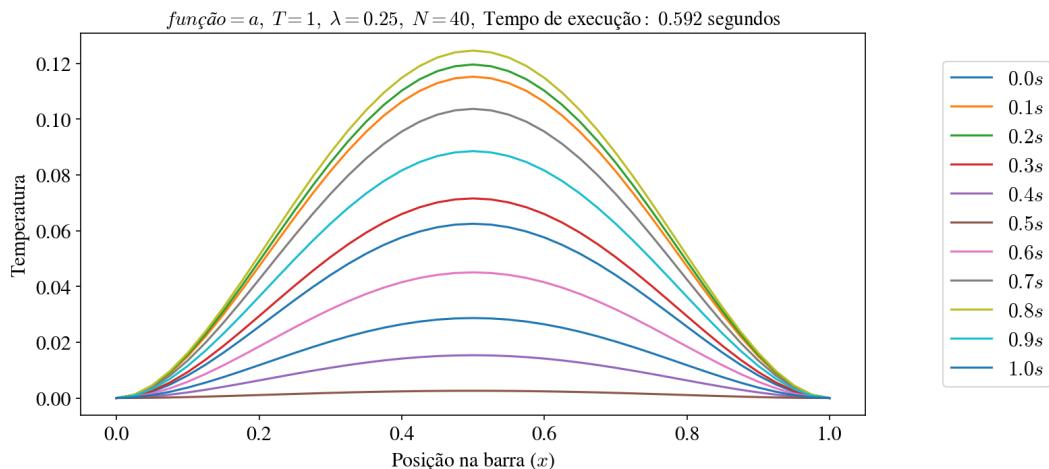
Primeiramente, testa-se o Método de Euler. Primeiramente, analisa-se o que ocorre conforme λ varia, mantendo N constante, e, em seguida, varia-se N , mantendo λ constante.

Assim, fazendo $\lambda = 0.25$ e alterando o valor de N , obtém-se os resultados para $N = 10$, $N = 40$ e $N = 320$, que estão expostos nos gráficos a seguir:

Temperatura em função da posição para certas séries temporais pelo método de Euler

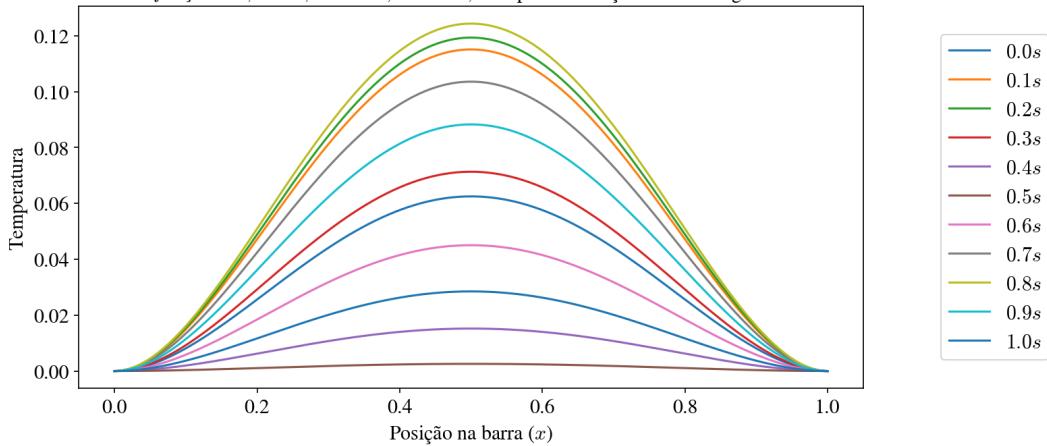


Temperatura em função da posição para certas séries temporais pelo método de Euler



Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = a, T = 1, \lambda = 0.25, N = 320,$ Tempo de execução : 58.734 segundos

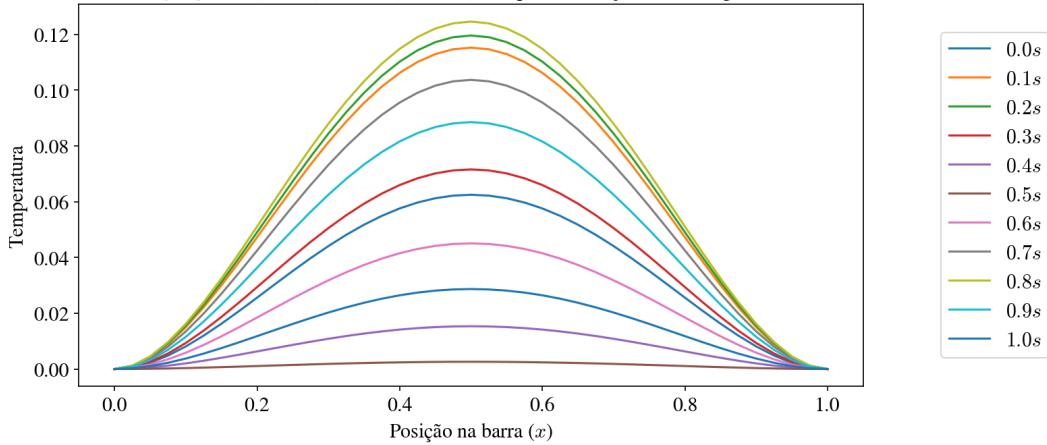


Note que o formato da curva permanece o mesmo, assim como os valores da temperatura em função da posição. A mudança principal é o aumento da suavidade da curva, que passa a ter "pontas" menos notáveis, conforme N aumenta.

Mantendo $N = 40$ e variando λ , por outro lado, obtém-se os seguintes gráficos:

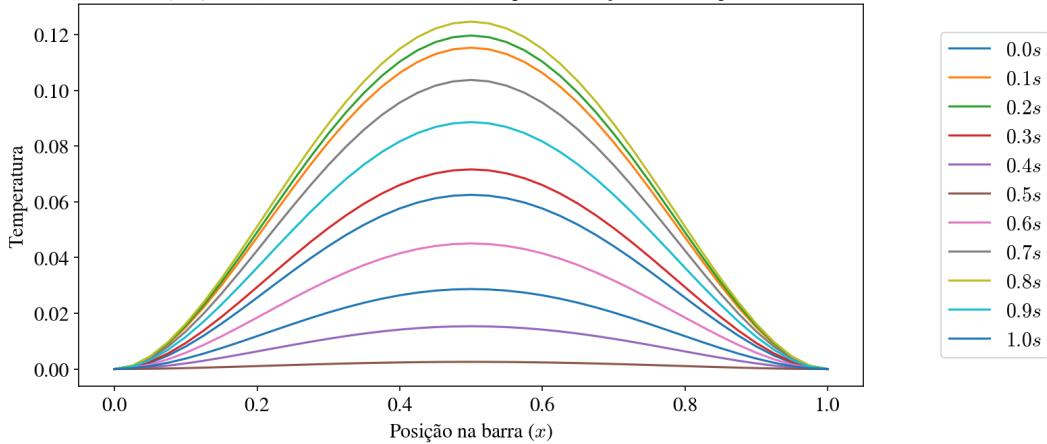
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = a, T = 1, \lambda = 0.25, N = 40,$ Tempo de execução : 0.592 segundos



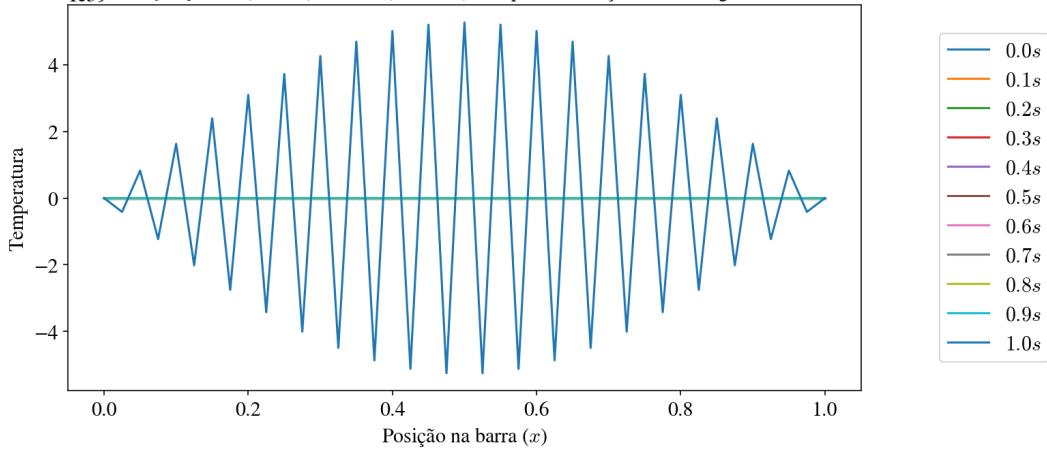
Temperatura em função da posição para certas séries temporais pelo método de Euler

$f = a$, $T = 1$, $\lambda = 0.5$, $N = 40$, Tempo de execução : 0.268 segundos



Temperatura em função da posição para certas séries temporais pelo método de Euler

$f = a$, $T = 1$, $\lambda = 0.51$, $N = 40$, Tempo de execução : 0.266 segundos

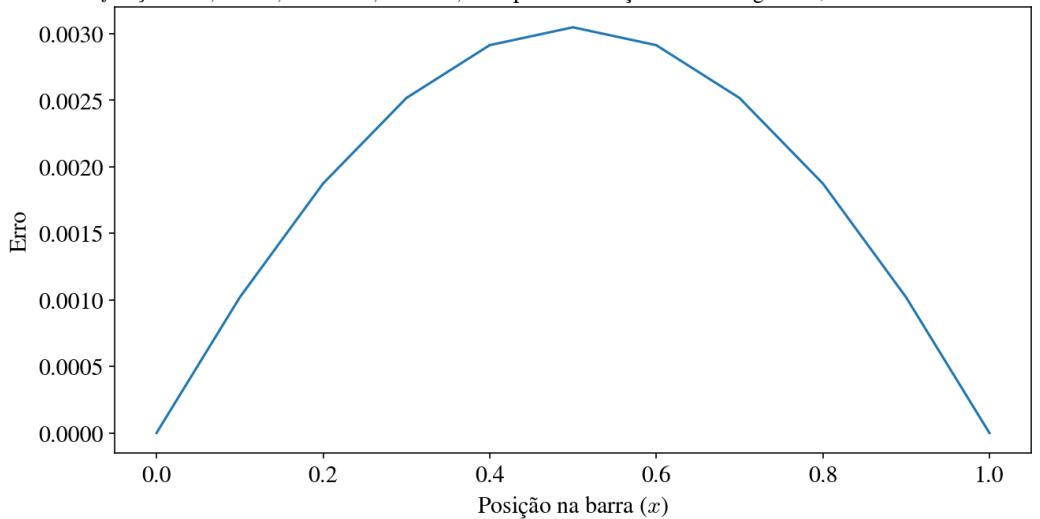


As temperaturas se comportam como o esperado nos casos $\lambda = 0.25$ e $\lambda = 0.5$. O gráfico é bem diferente quando $\lambda = 0.51$, no entanto. Isso ocorre pois o Método de Euler é condicionalmente estável, e torna-se instável caso $\lambda > 0.5$, sendo o perfil distinto do terceiro gráfico, que apresenta fortes oscilações da temperatura, consequência disto.

Observe o erro (em $T = 1$) em função de N e λ , novamente variando um e mantendo o outro constante. Primeiro, fazendo $\lambda = 0.25$ e variando N , o erro encontrado é mostrado nos gráficos a seguir (novamente para $N = 10$, $N = 40$ e $N = 320$):

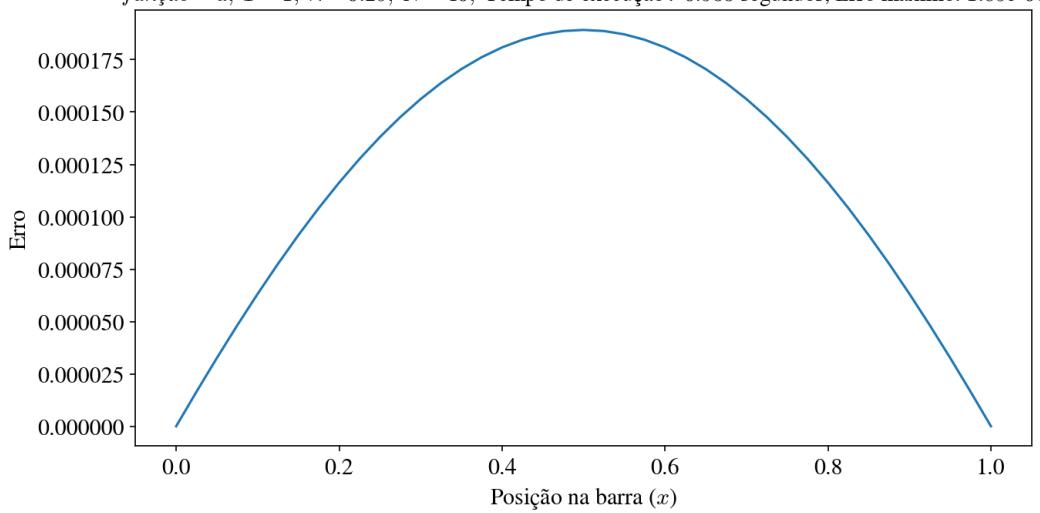
Erro em função da posição para o instante $t = T$ no método de Euler

função = a , $T = 1$, $\lambda = 0.25$, $N = 10$, Tempo de execução: 0.057 segundos, Erro máximo: 3.05e-03

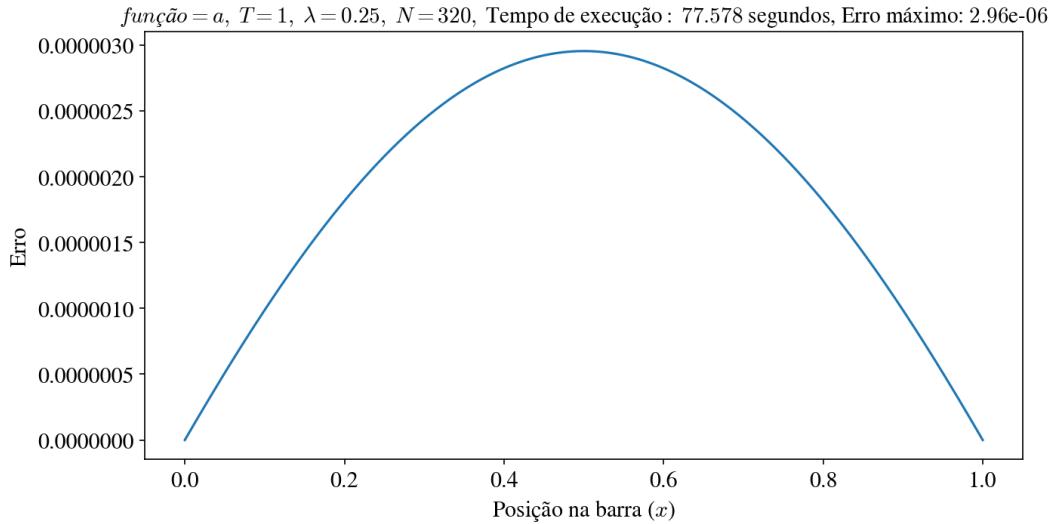


Erro em função da posição para o instante $t = T$ no método de Euler

função = a , $T = 1$, $\lambda = 0.25$, $N = 40$, Tempo de execução: 0.933 segundos, Erro máximo: 1.89e-04



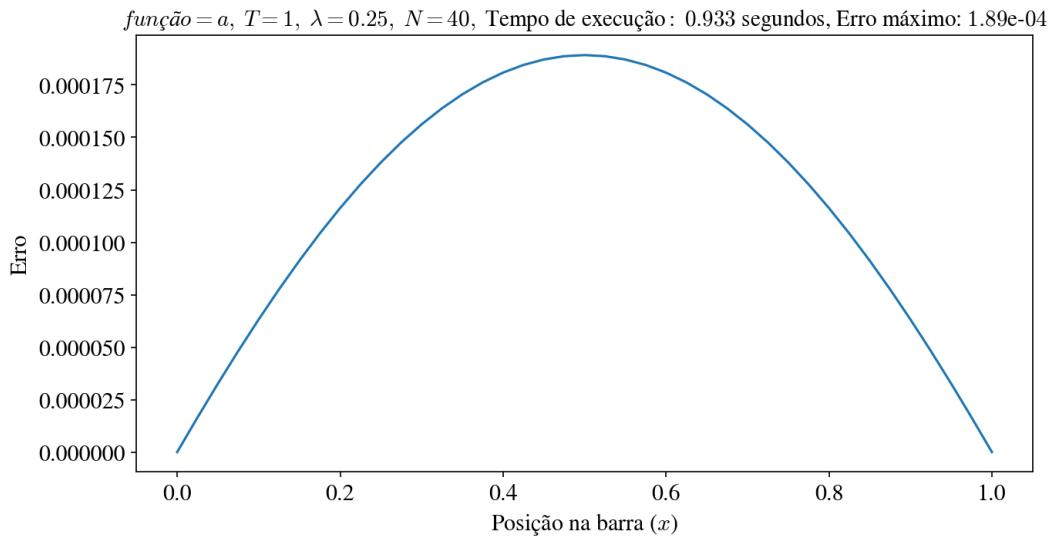
Erro em função da posição para o instante $t = T$ no método de Euler



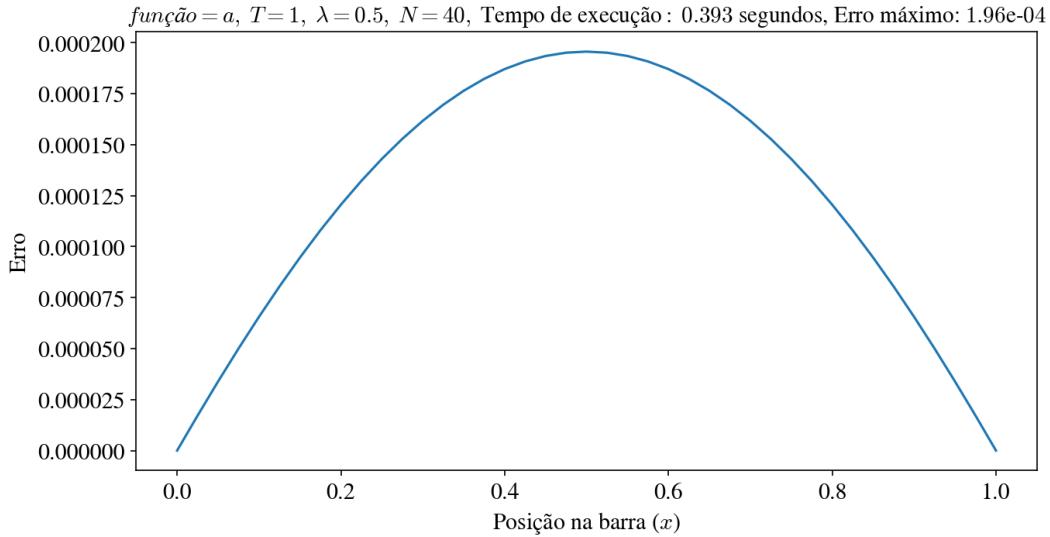
Nota-se que o erro é nulo nas extremidades da curva, o que é esperado, uma vez que a temperatura nestes pontos é conhecida e definida pelas condições de contorno, e segue aumentando conforme nos afastamos das extremidades da barra. Além disso, a ordem de grandeza do erro diminui conforme o valor de N é incrementado. Isto ocorre porque aumentar o valor de N , fixado λ , reduz o valor de Δt , o que por sua vez limita o erro de truncamento.

Mantendo $N = 40$ e variando λ :

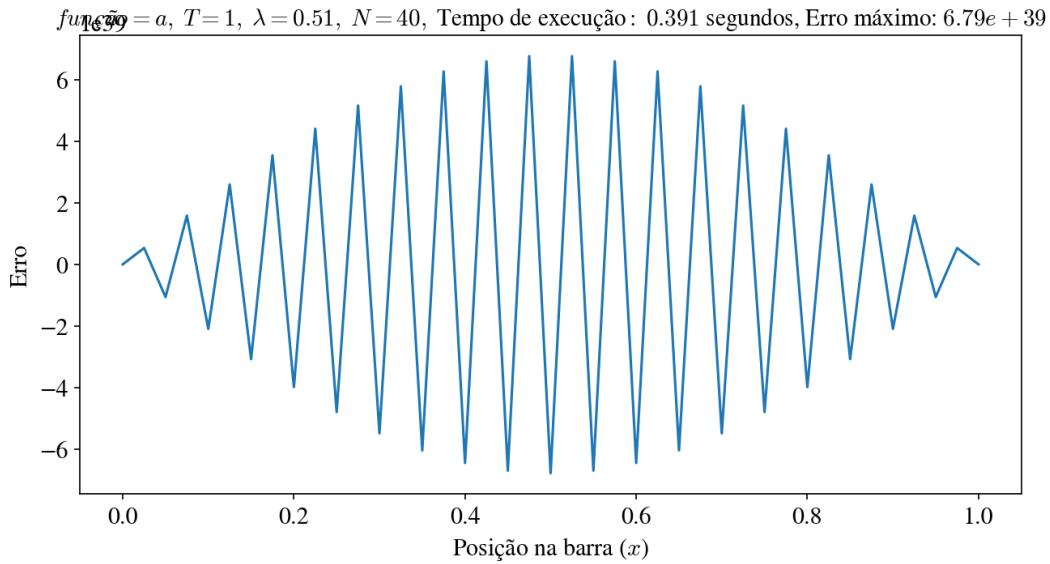
Erro em função da posição para o instante $t = T$ no método de Euler



Erro em função da posição para o instante $t = T$ no método de Euler



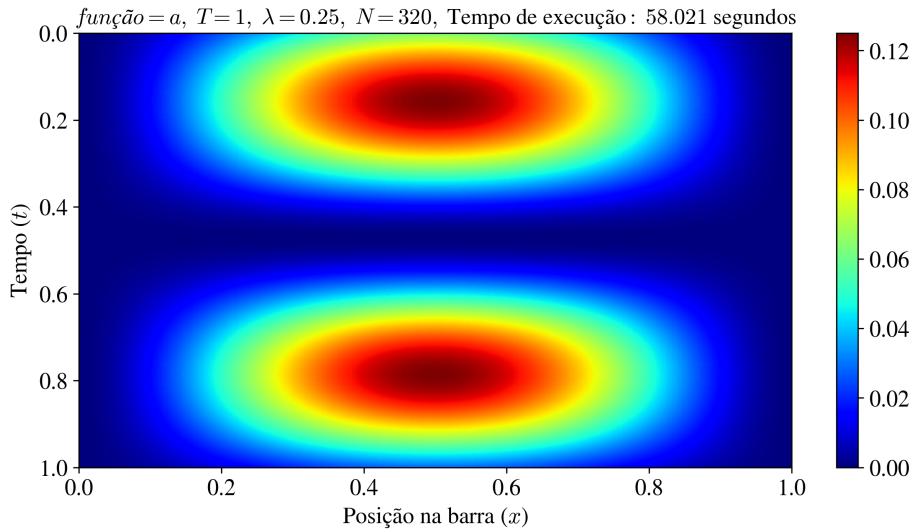
Erro em função da posição para o instante $t = T$ no método de Euler



Novamente, o comportamento oscilatório do gráfico de $\lambda = 0.51$ é devido à instabilidade do método quando $\lambda > 0.5$. A magnitude também diminui conforme aumenta-se o valor de λ , e o pico continua no meio da barra, no ponto mais distante das extremidades.

Para efeitos demonstrativos, observe o *heatmap* da barra, para $N = 320$ e $\lambda = 0,25$:

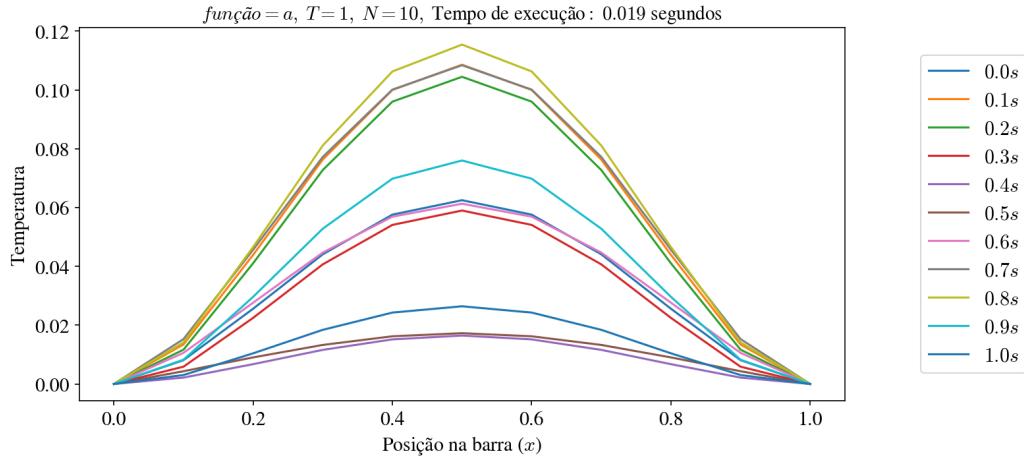
Mapa de Temperatura para a barra inteira em todos os *ticks* de tempo no método de Euler



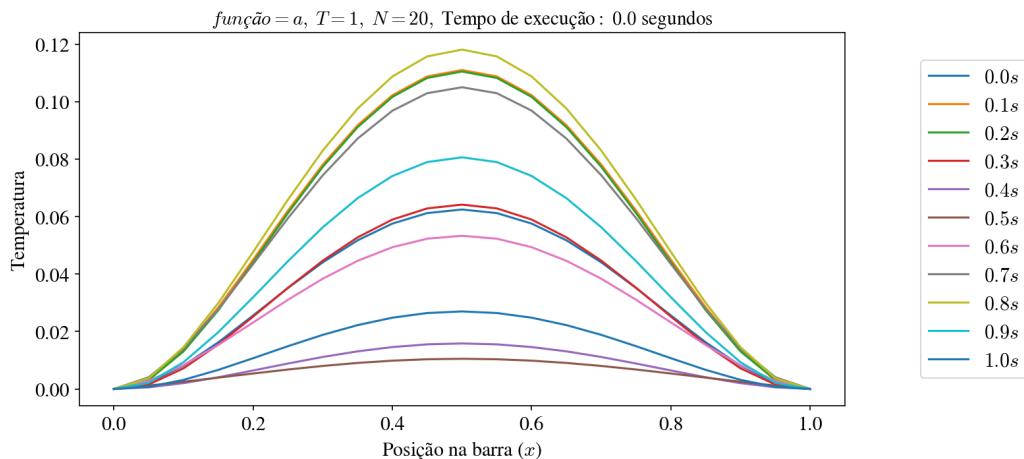
Heatmaps para todos os testes pedidos podem ser encontrados nos arquivos em anexo.

No método de Euler Implícito utiliza-se $\Delta t = \Delta x$ e, portanto, $M = N$. Para os vários valores de N :

Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

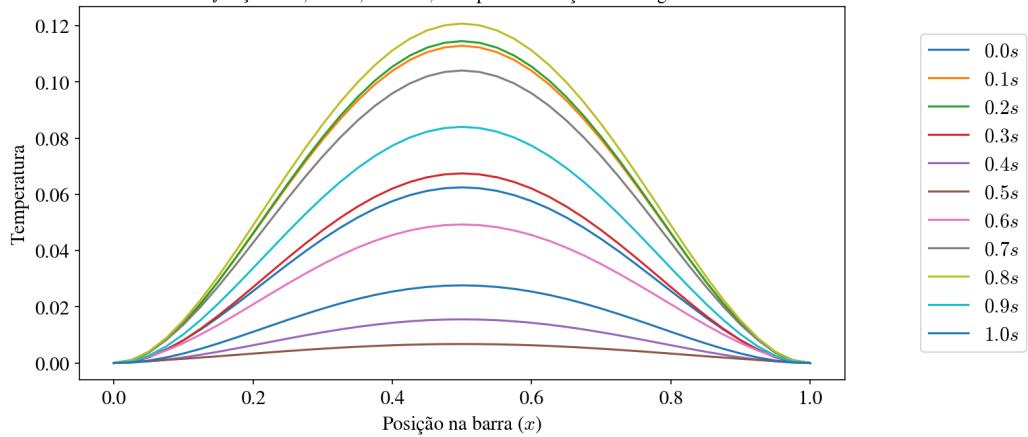


Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler



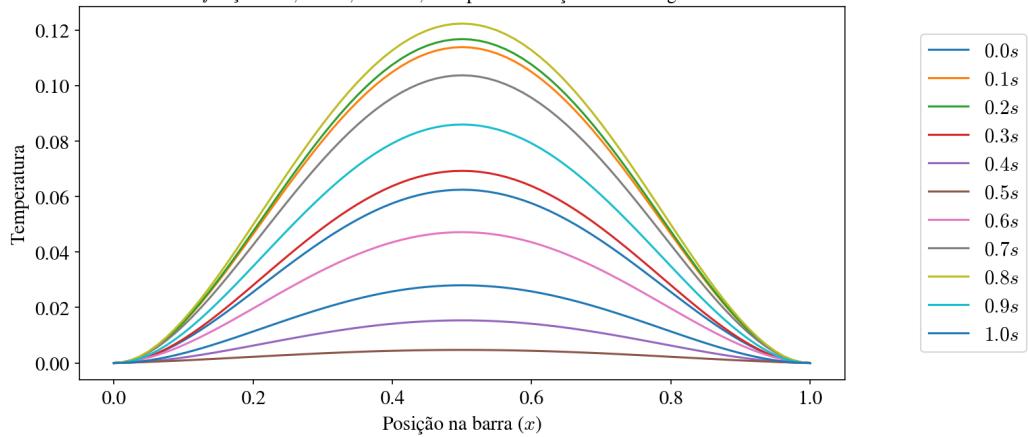
Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

função = a, T = 1, N = 40, Tempo de execução : 0.0 segundos



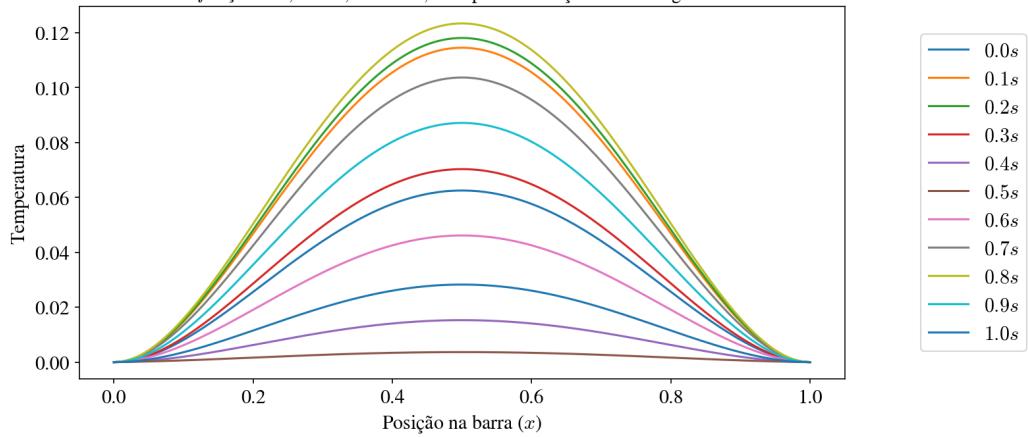
Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

função = a, T = 1, N = 80, Tempo de execução : 0.031 segundos



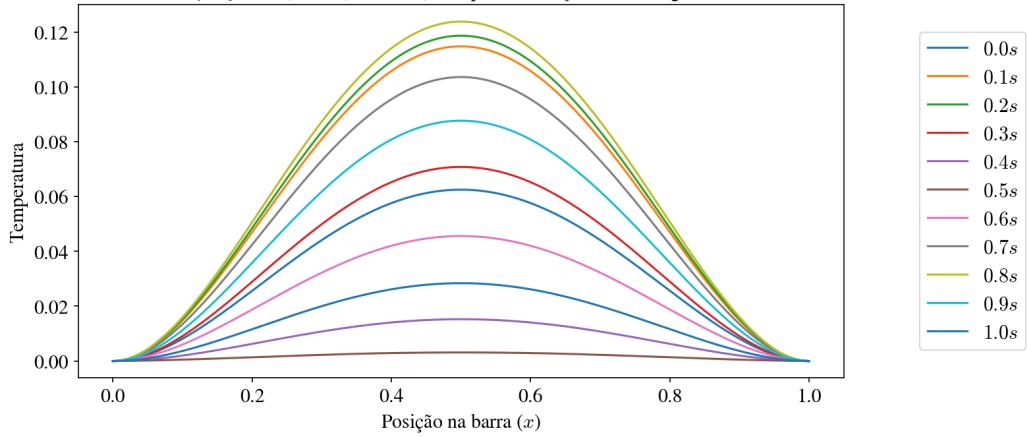
Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

função = a, T = 1, N = 160, Tempo de execução : 0.078 segundos



Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

$função = a, T = 1, N = 320$, Tempo de execução : 0.361 segundos

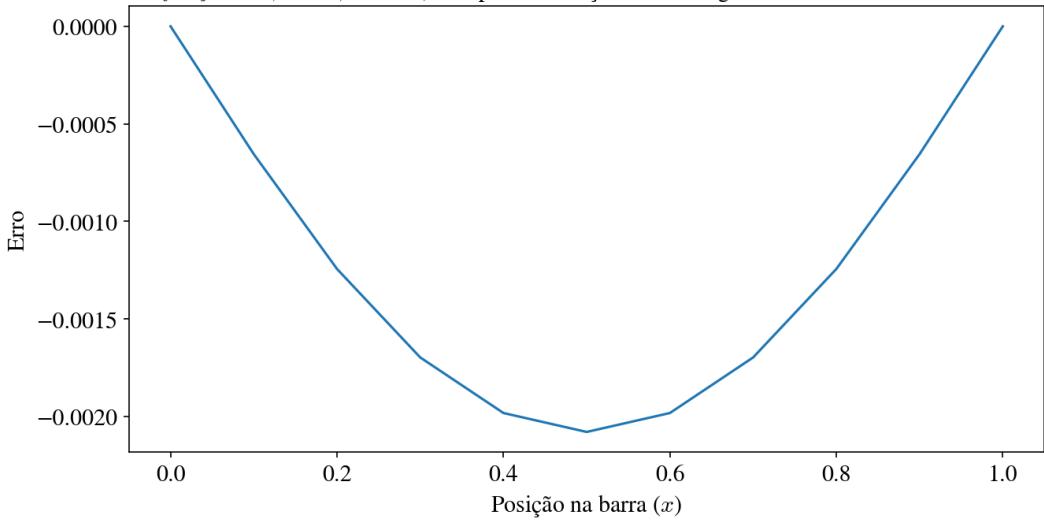


Não há mudança nos resultados, quando comparado com o método de Euler. O aumento de N contribui para suavizar a curva, e como o método é incondicionalmente estável, o valor de λ é irrelevante. A principal diferença entre os métodos é o tempo de execução, que sofre um grande decréscimo. Em $N = 320$, o caso mais aparente, o tempo de execução cai de 58.734 segundos (com $\lambda = 0.25$) no Método de Euler para 0.361 segundos no Método Implícito, uma queda de 99.39%

Analizando, agora, o erro:

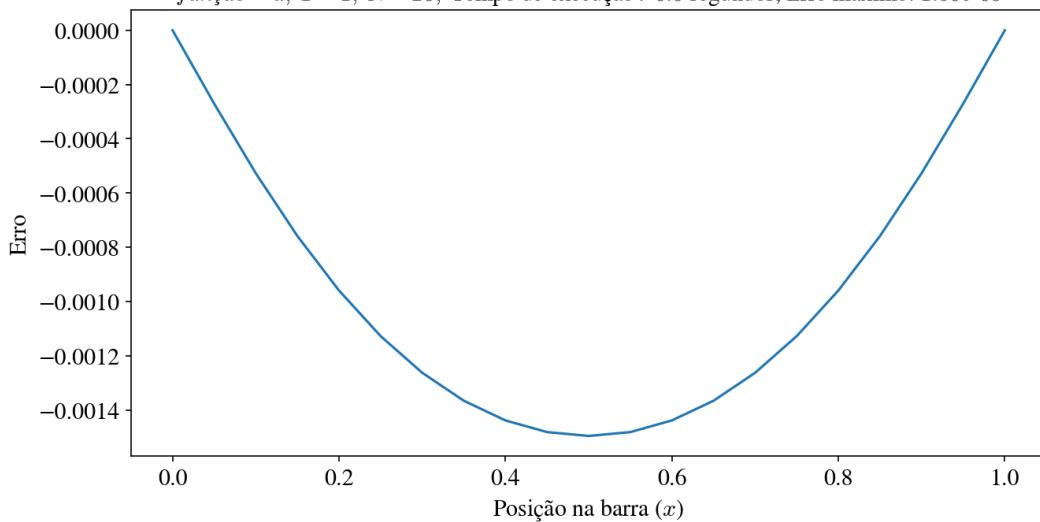
Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = a, T = 1, N = 10$, Tempo de execução : 0.019 segundos, Erro máximo: 2.08e-03



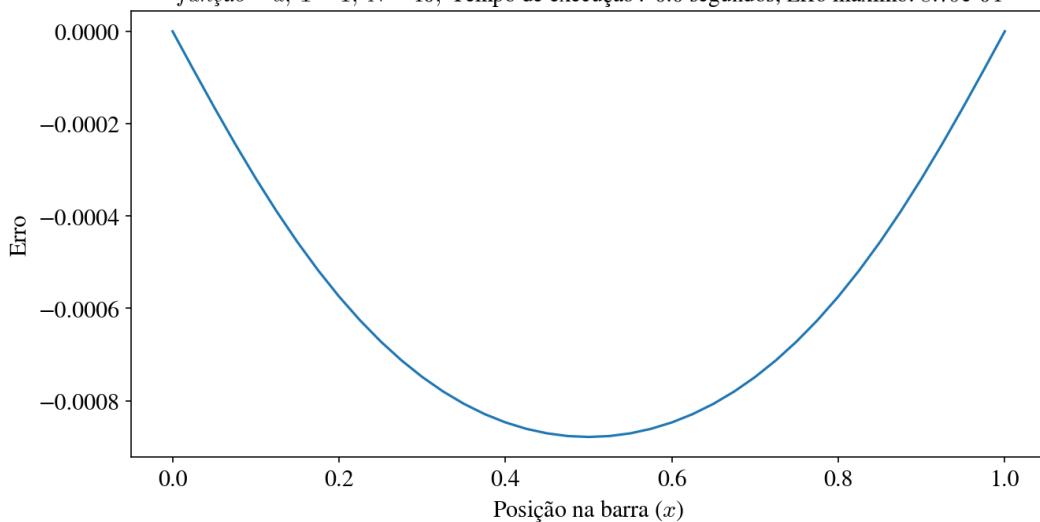
Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = a, T = 1, N = 20$, Tempo de execução : 0.0 segundos, Erro máximo: 1.50e-03



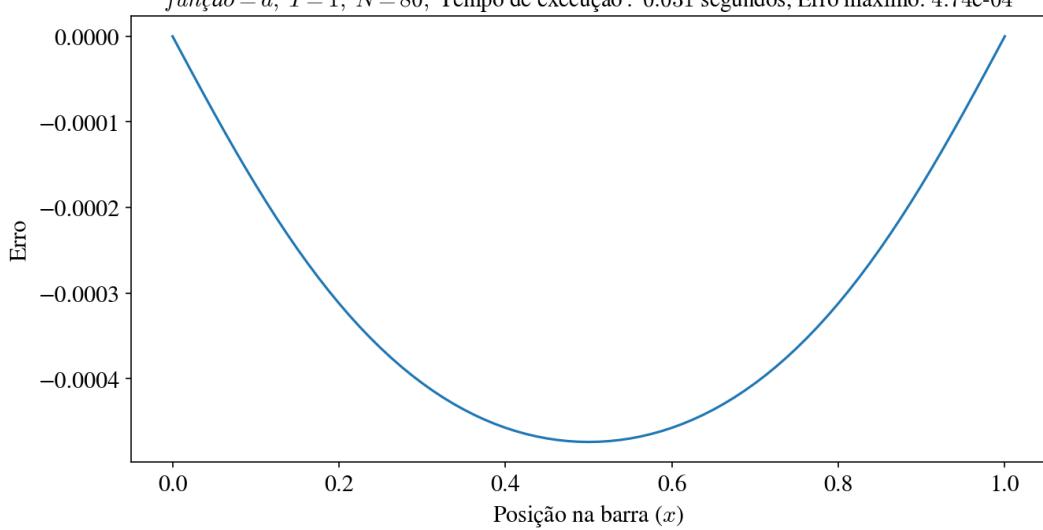
Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = a, T = 1, N = 40$, Tempo de execução : 0.0 segundos, Erro máximo: 8.79e-04



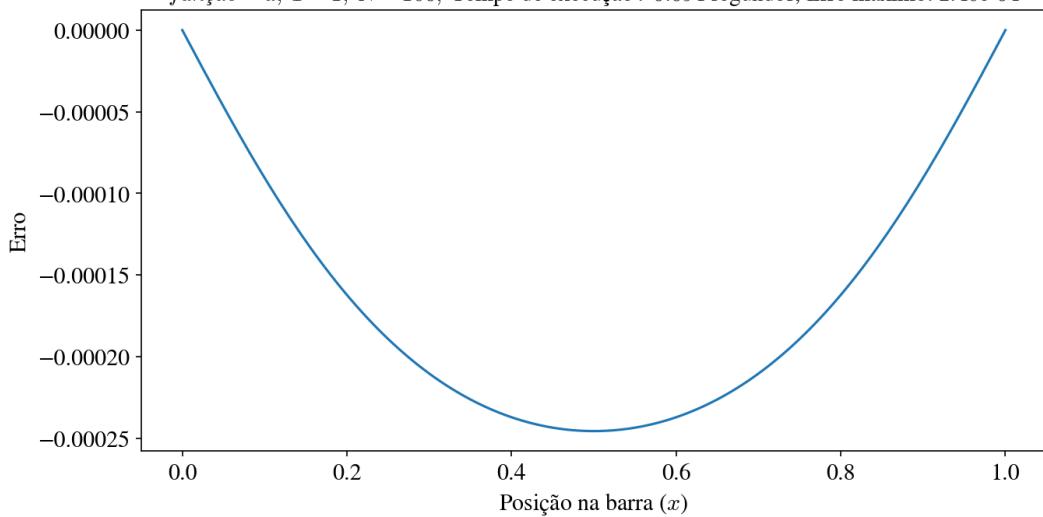
Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = a, T = 1, N = 80$, Tempo de execução : 0.031 segundos, Erro máximo: 4.74e-04

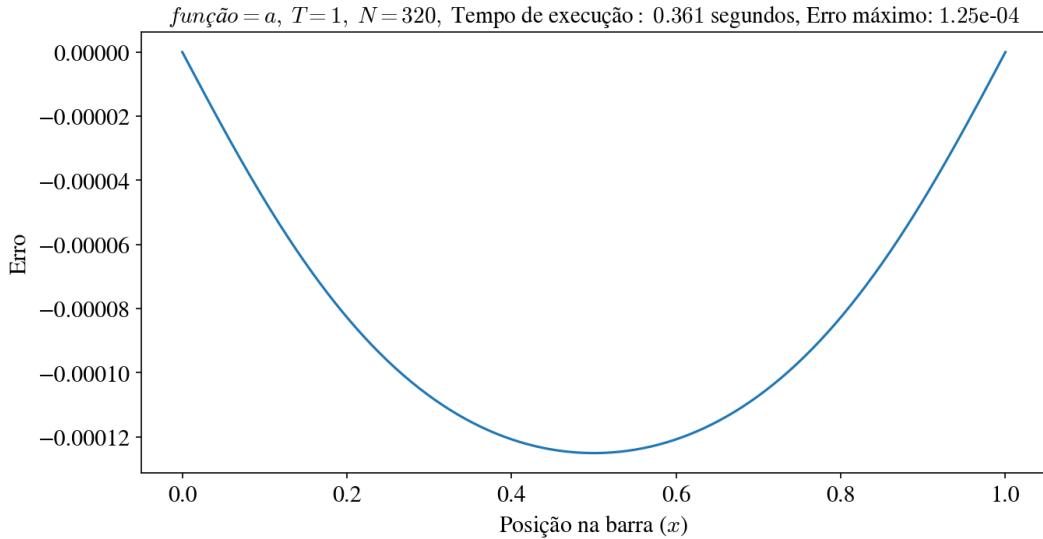


Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = a, T = 1, N = 160$, Tempo de execução : 0.094 segundos, Erro máximo: 2.46e-04



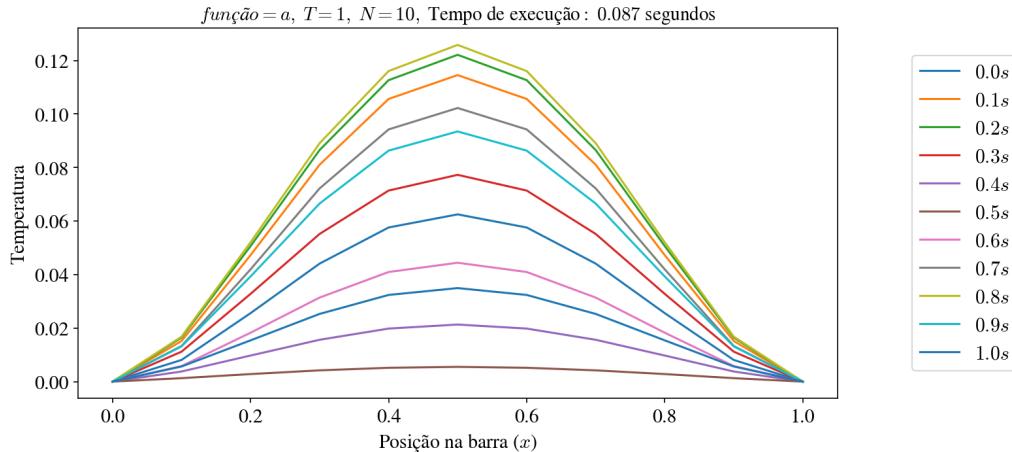
Erro em função da posição para o instante $t = T$ no método de Implicit Euler



Repara-se que o erro diminui conforme aumenta-se N . De fato, o limite superior do erro de truncamento varia linearmente com Δt , e quadraticamente com Δx . Como $\Delta x = \Delta t$ e $M = N$, aumentar N limita o valor do erro de truncamento. Comparando isto com o erro do método anterior, no entanto, observa-se que a ordem de grandeza é maior no método implícito. De fato, considerando $N = 320$, o erro máximo é da ordem de 10^{-4} no método implícito, e de 10^{-6} no método anterior. Isso ocorre porque o Método de Euler, quando estável, possui ordem de convergência 2 em Δx , enquanto o método implícito tem ordem de convergência 1 em Δt (adicionalmente $\Delta t = \Delta x$, para este método). Logo, apesar do custo computacional do primeiro método ser superior ao do segundo, ele converge mais rapidamente para a solução exata, e por isso apresenta menor erro.

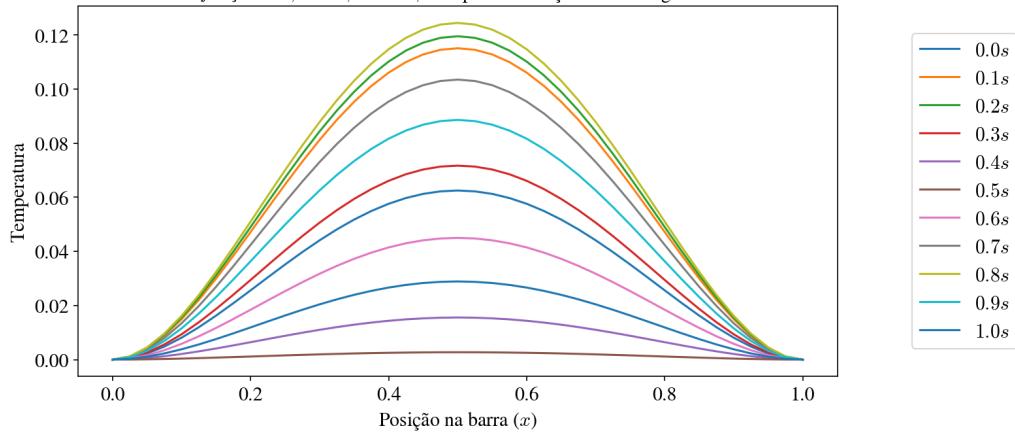
Por fim, testa-se Crank-Nicolson. Os gráficos da temperatura na barra são os seguintes:

Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson



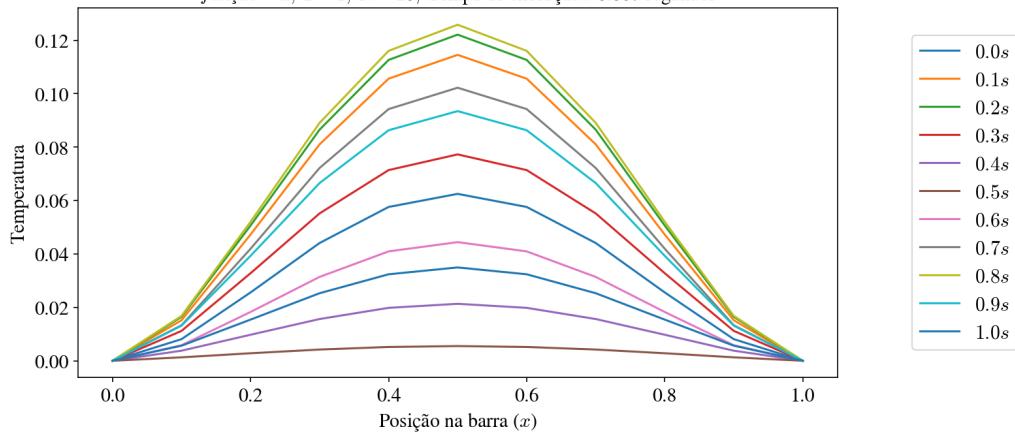
Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson

$função = a, T = 1, N = 40$, Tempo de execução : 0.047 segundos



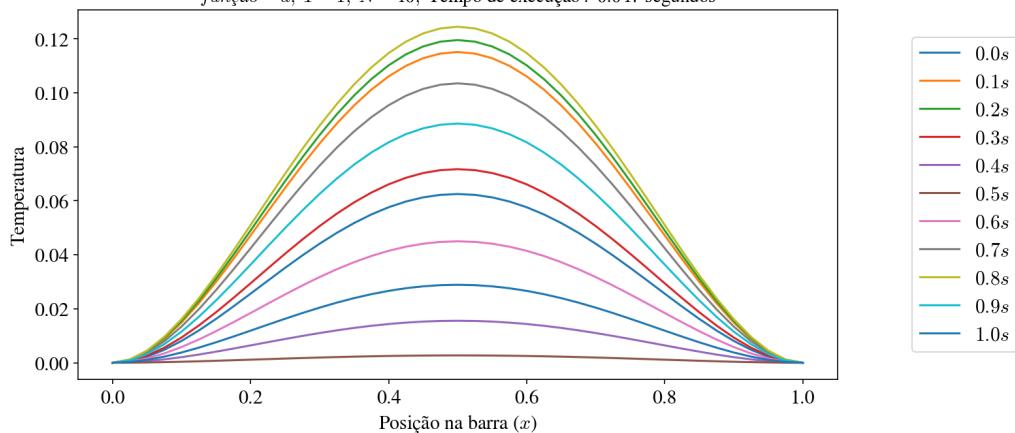
Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson

$função = a, T = 1, N = 10$, Tempo de execução : 0.087 segundos



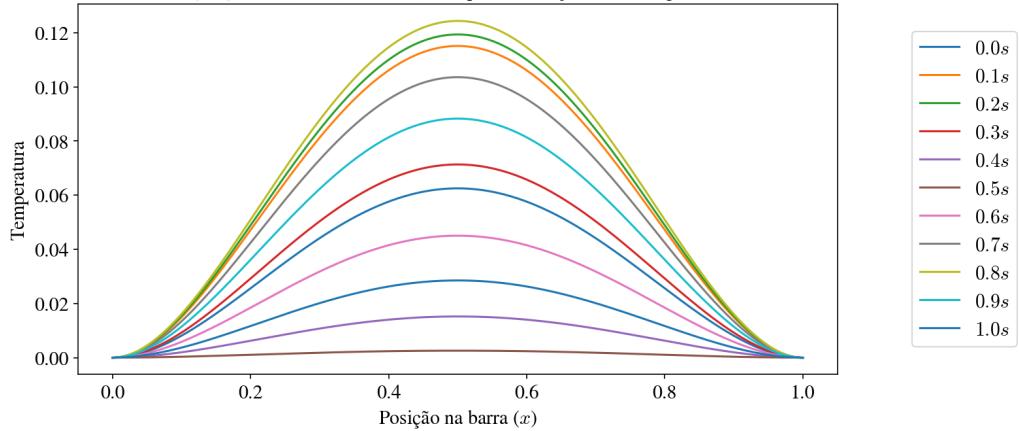
Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson

$função = a, T = 1, N = 40$, Tempo de execução : 0.047 segundos



Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson

função = a, T = 1, N = 320, Tempo de execução : 0.609 segundos

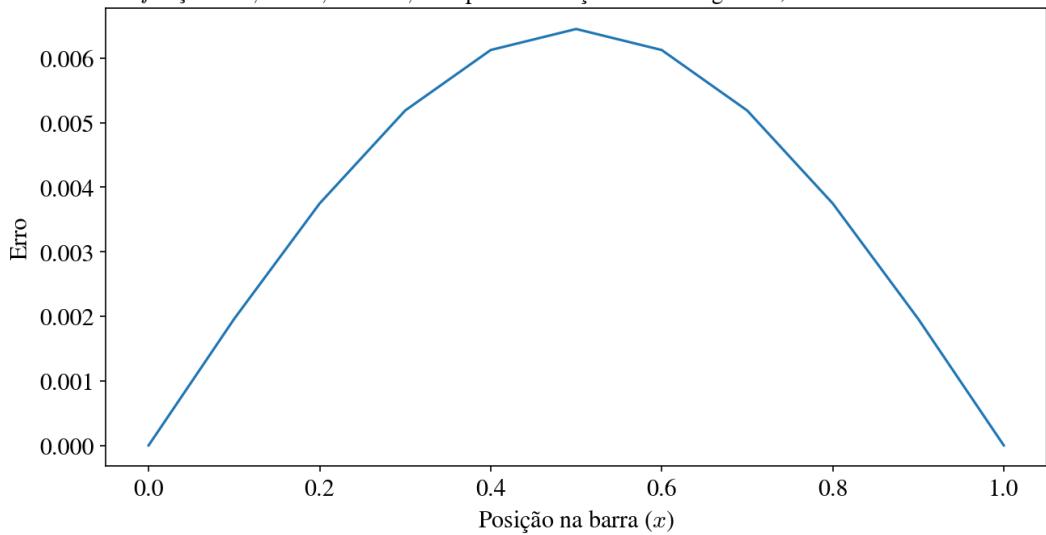


Não há mudanças significativas em comparação com o Método de Euler implícito no que diz respeito aos gráficos de temperatura. Os tempos de execução são bem próximos, diferindo por apenas 0.248 segundos no caso $N = 320$. Os valores da temperatura também são os mesmos, e o aumento de N serve para suavizar a curva.

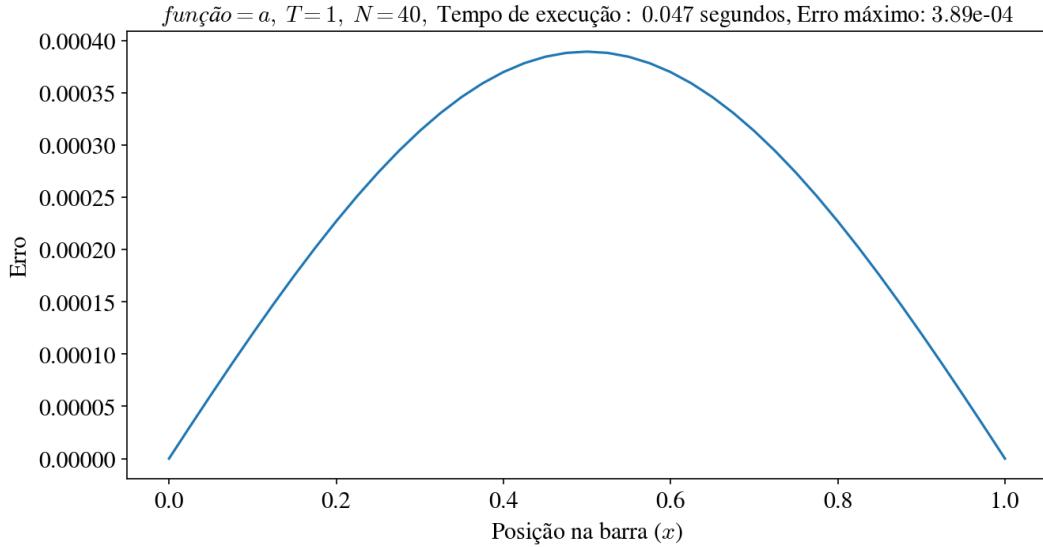
Comparamos agora os erros:

Erro em função da posição para o instante $t = T$ no método de Crank Nicolson

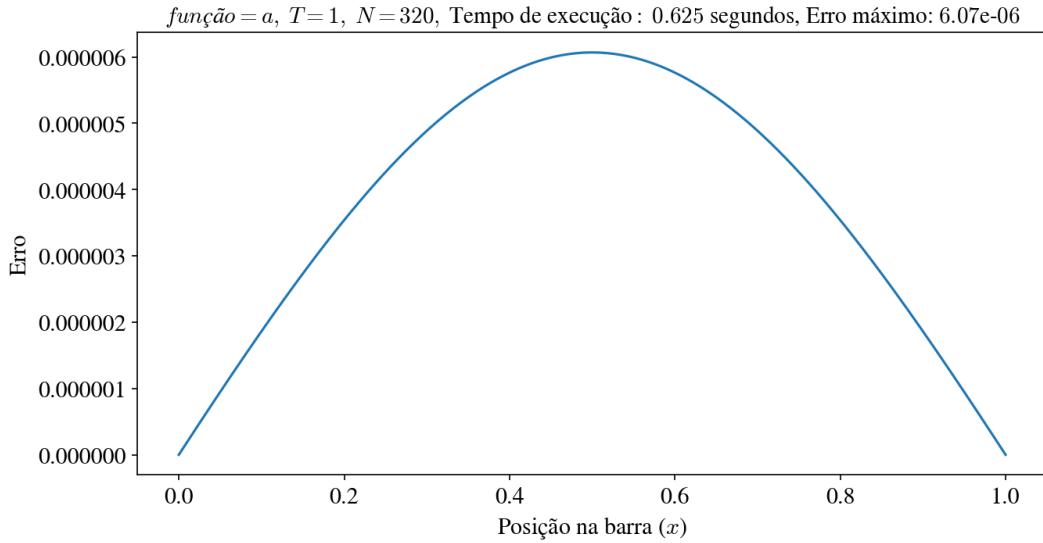
função = a, T = 1, N = 10, Tempo de execução : 0.087 segundos, Erro máximo: 6.45e-03



Erro em função da posição para o instante $t = T$ no método de Crank Nicolson



Erro em função da posição para o instante $t = T$ no método de Crank Nicolson



Note que a ordem de grandeza do erro é menor que no Método de Euler Implícito, se aproximando mais dos valores obtidos no Método de Euler. Isso ocorre porque o Método de Crank-Nicolson possui ordem de convergência 2 tanto em Δx quanto em Δt (que, novamente, são iguais).

6.2 Função b)

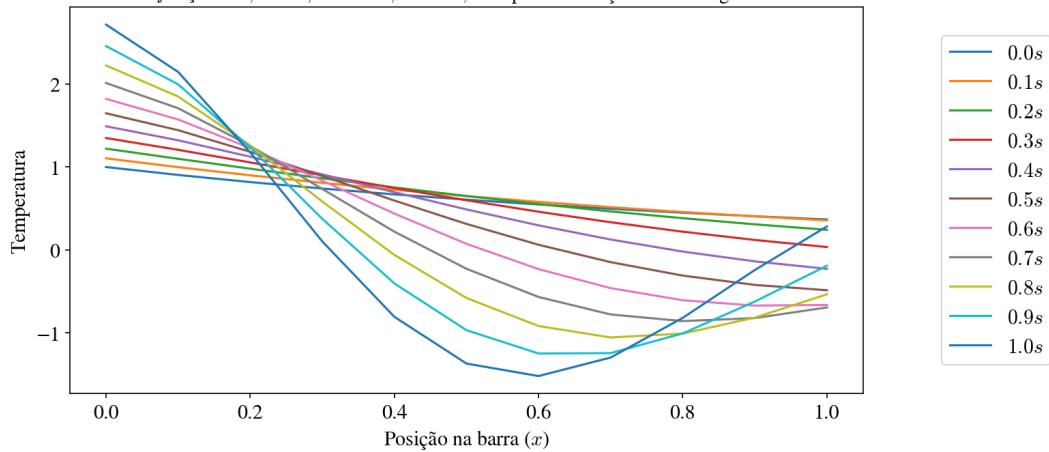
A função $f(x, t)$ do item b) foi calculada na Seção 5.2, sendo definida da seguinte forma:

$$f(x, t) = [25t^2 \cos(5tx) - 5(x + 2t) \sin(5tx)]e^{t-x}$$

Iniciando-se a análise pelo Método de Euler, para esta função, mantendo $\lambda = 0.25$ e variando N , têm-se:

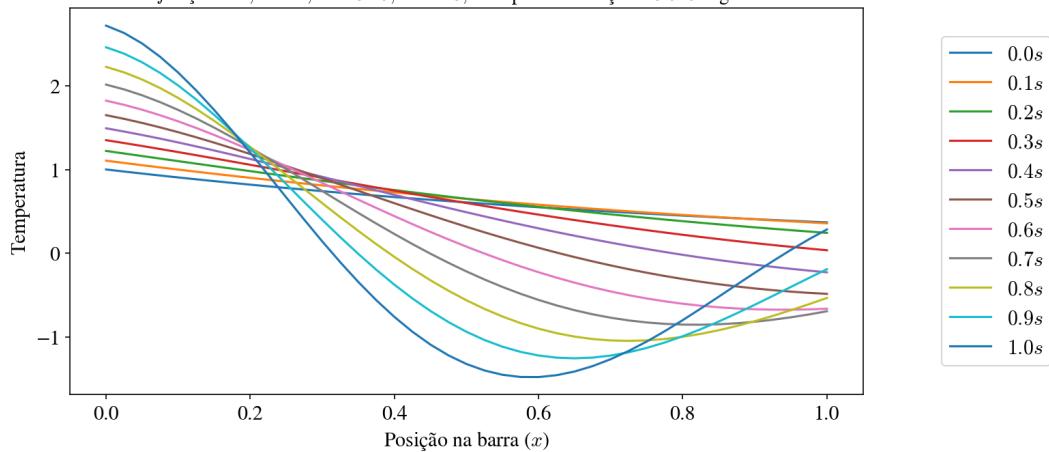
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 10$, Tempo de execução : 0.021 segundos



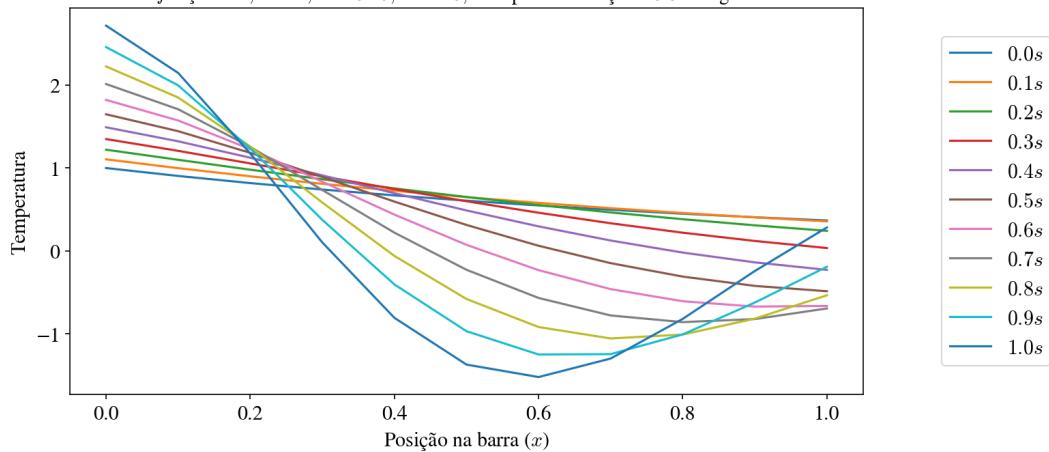
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 40$, Tempo de execução : 0.529 segundos



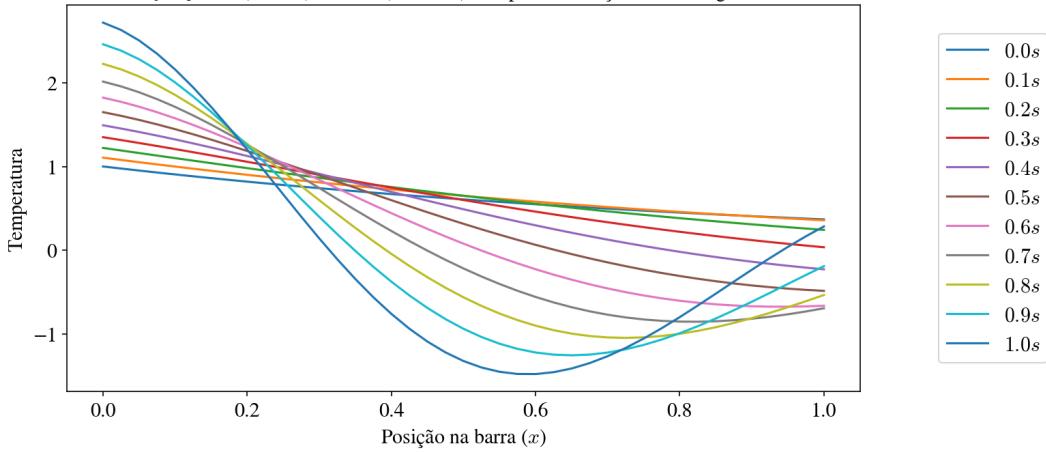
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 10$, Tempo de execução : 0.021 segundos



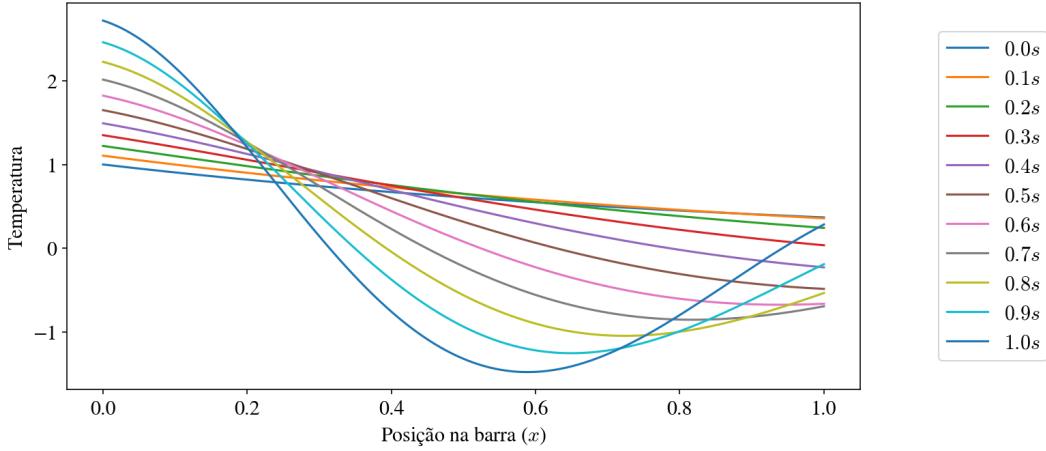
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = b, T = 1, \lambda = 0.25, N = 40$, Tempo de execução : 0.529 segundos



Temperatura em função da posição para certas séries temporais pelo método de Euler

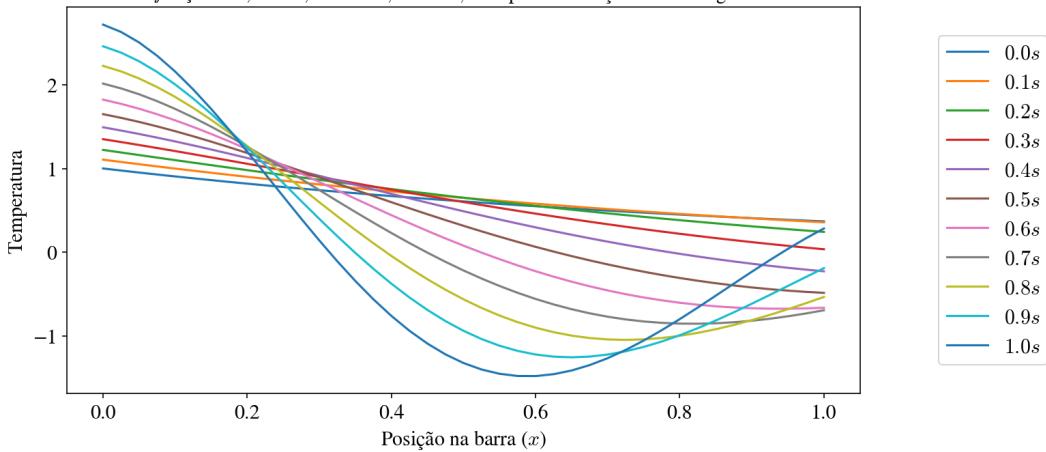
$função = b, T = 1, \lambda = 0.25, N = 320$, Tempo de execução : 58.105 segundos



Novamente, a única mudança é a suavização da curva conforme incrementa-se N . Mantendo agora $N = 40$ e variando λ :

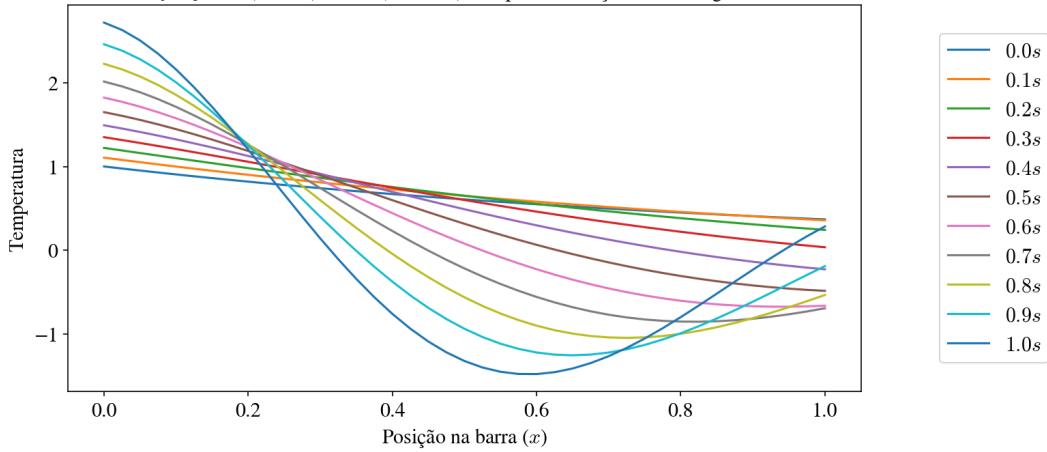
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = b, T = 1, \lambda = 0.25, N = 40$, Tempo de execução : 0.529 segundos



Temperatura em função da posição para certas séries temporais pelo método de Euler

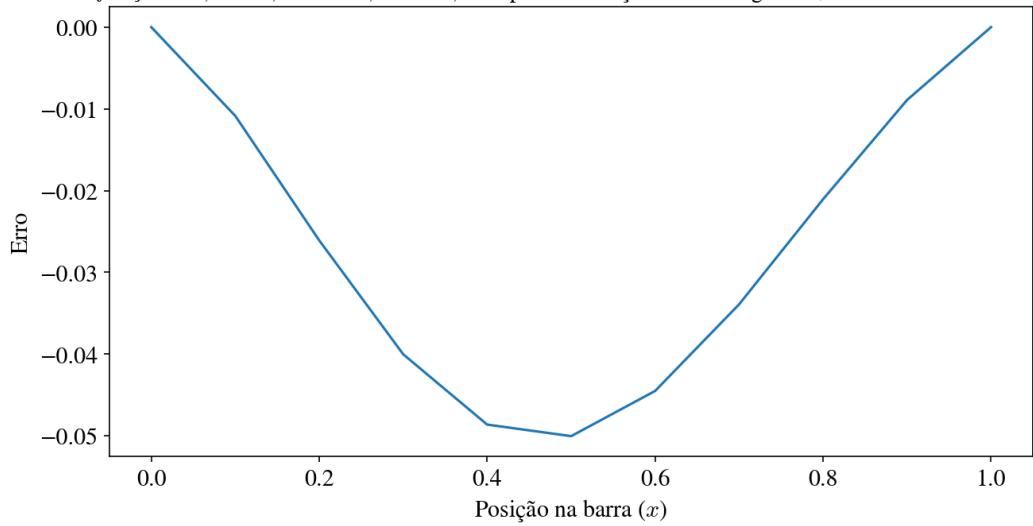
$f = b$, $T = 1$, $\lambda = 0.5$, $N = 40$, Tempo de execução : 0.283 segundos



Repare que os resultados são os mesmos, como era esperado ao variar-se o λ , pois a mudança deste parâmetro está relacionada aos erros de truncamento. Os gráficos do erro, mantendo-se λ constante e variando N , são mostrados nas figuras a seguir:

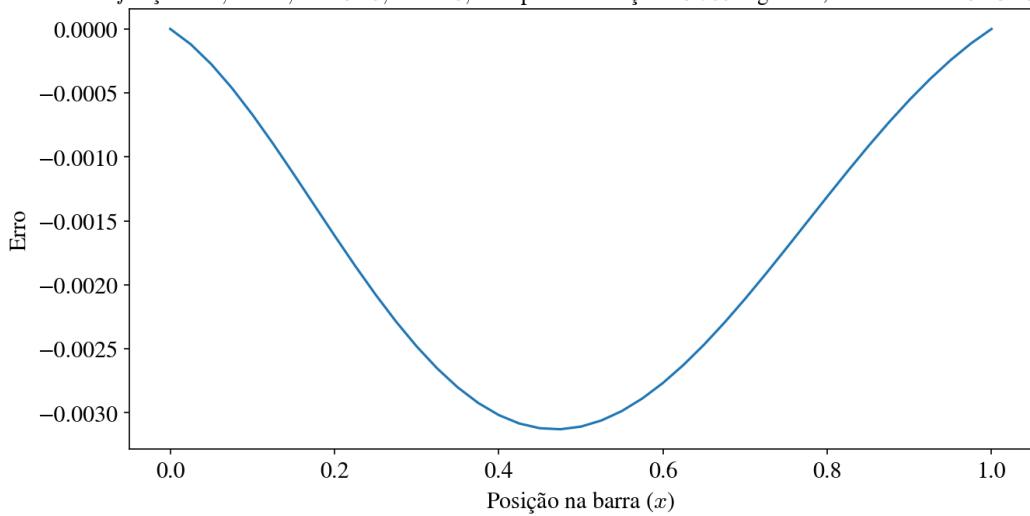
Erro em função da posição para o instante $t = T$ no método de Euler

$f = b$, $T = 1$, $\lambda = 0.25$, $N = 10$, Tempo de execução : 0.037 segundos, Erro máximo: 5.00e-02



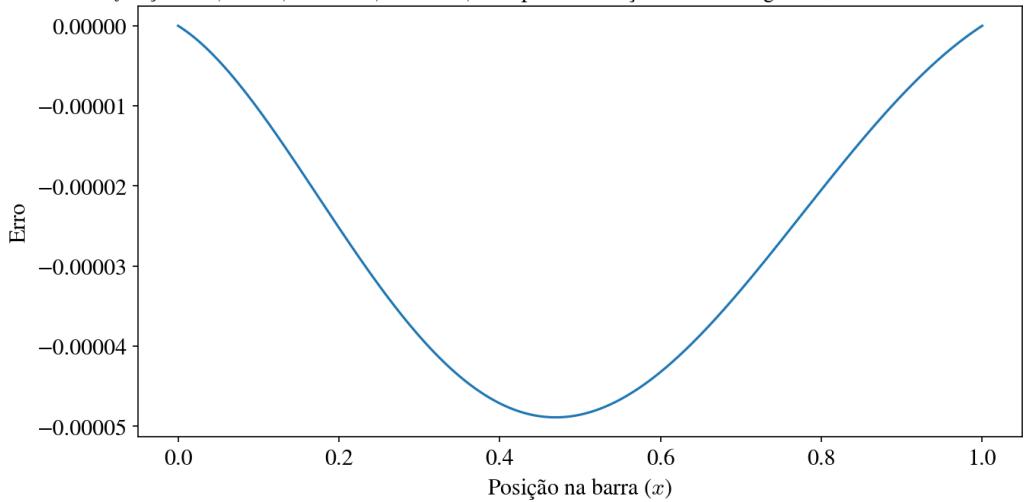
Erro em função da posição para o instante $t = T$ no método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 40$, Tempo de execução : 0.765 segundos, Erro máximo: 3.13e-03



Erro em função da posição para o instante $t = T$ no método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 320$, Tempo de execução : 79.054 segundos, Erro máximo: 4.89e-05

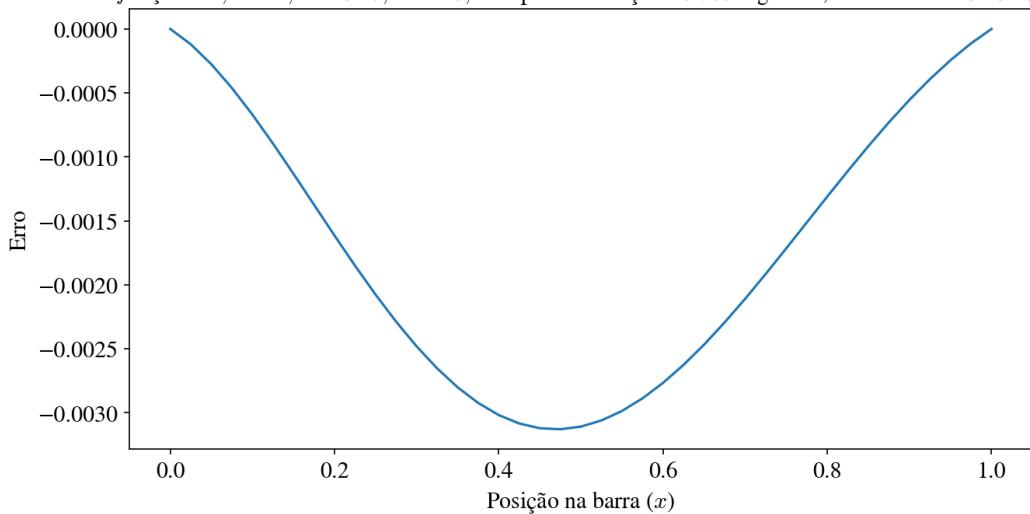


Aumentar o N proporciona uma maior suavidade aos gráficos, como esperado. Além disso, a magnitude do erro diminui (também é esperado, pois N é inversamente proporcional a Δt e Δx) e se aproxima de zero conforme se aproxima das pontas da barra, onde o erro é nulo. Além disso, o máximo da curva está no meio da barra, região mais distante das extremidades.

Mantendo $N = 40$ e variando λ :

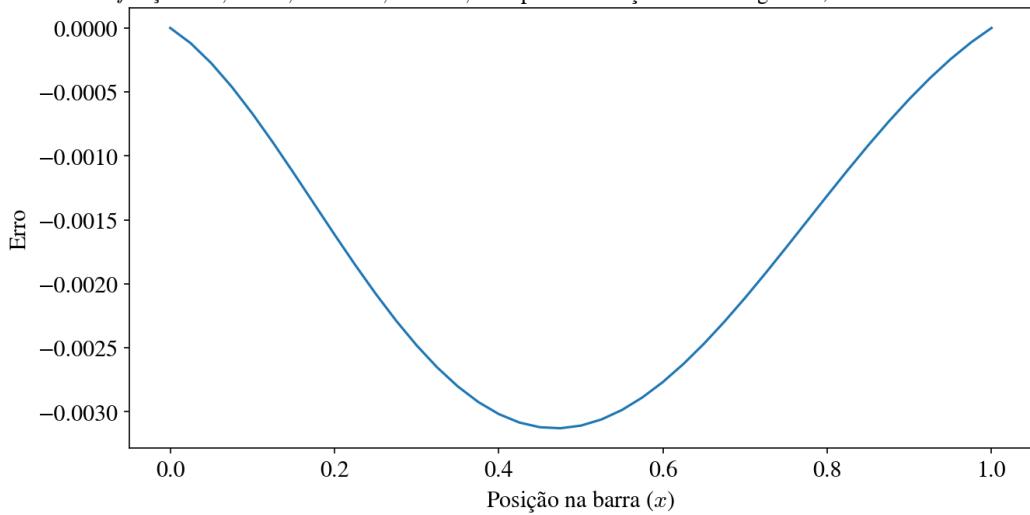
Erro em função da posição para o instante $t = T$ no método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 40$, Tempo de execução : 0.765 segundos, Erro máximo: 3.13e-03

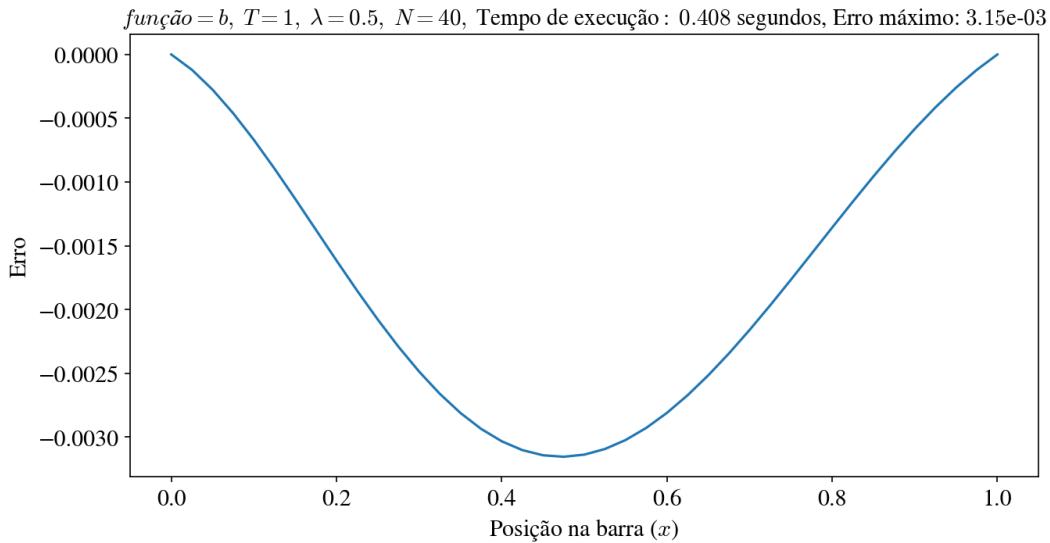


Erro em função da posição para o instante $t = T$ no método de Euler

$função = b$, $T = 1$, $\lambda = 0.25$, $N = 40$, Tempo de execução : 0.765 segundos, Erro máximo: 3.13e-03



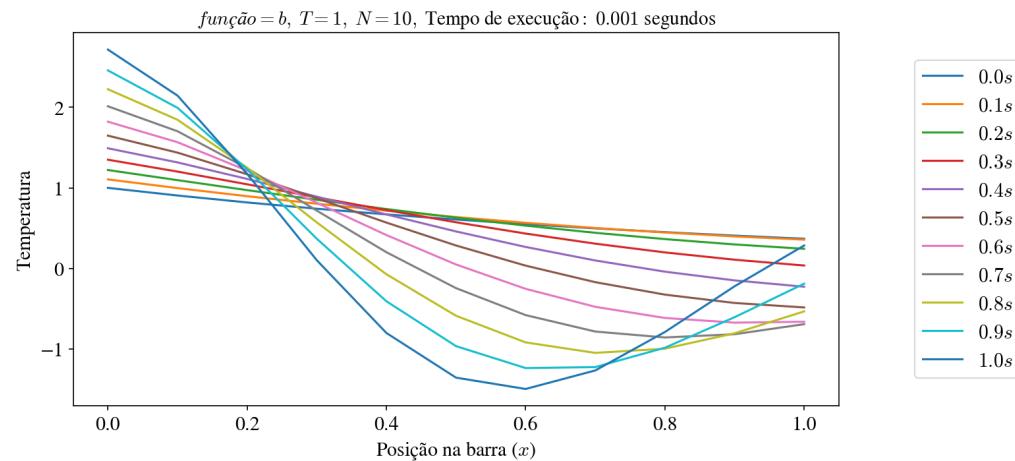
Erro em função da posição para o instante $t = T$ no método de Euler



Nota-se, neste caso, comportamento similar por parte de ambas as curvas, mas é também observável que o erro máximo é maior quando $\lambda = 0.5$. Novamente, isso é esperado, pois acarreta num aumento de Δt , que é diretamente proporcional ao erro de truncamento.

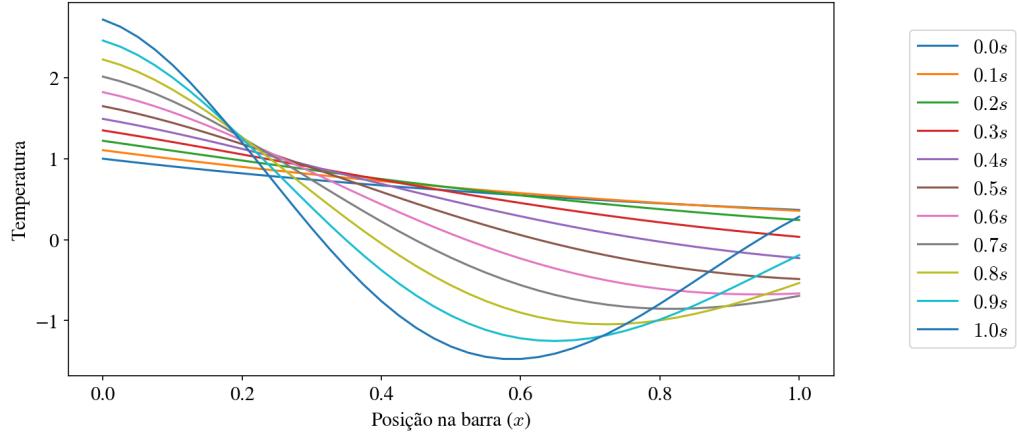
Com o método implícito os gráficos de temperatura são:

Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler



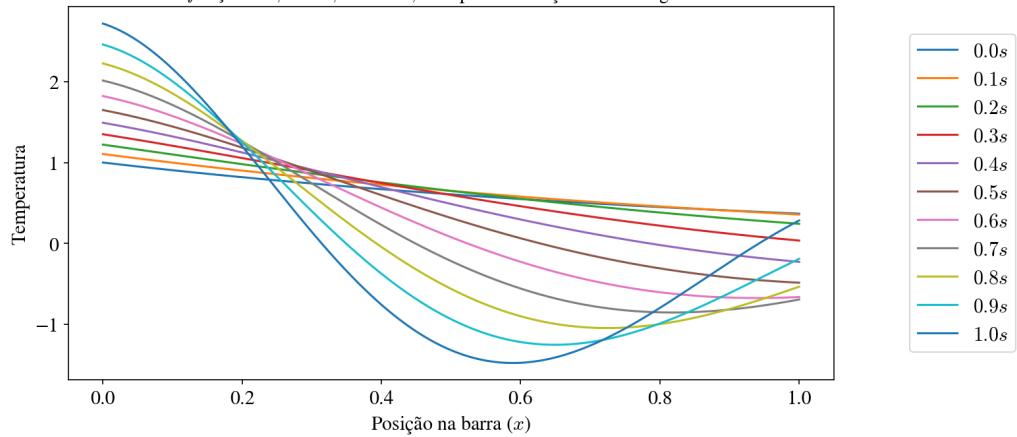
Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

$função = b$, $T = 1$, $N = 40$, Tempo de execução : 0.016 segundos



Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

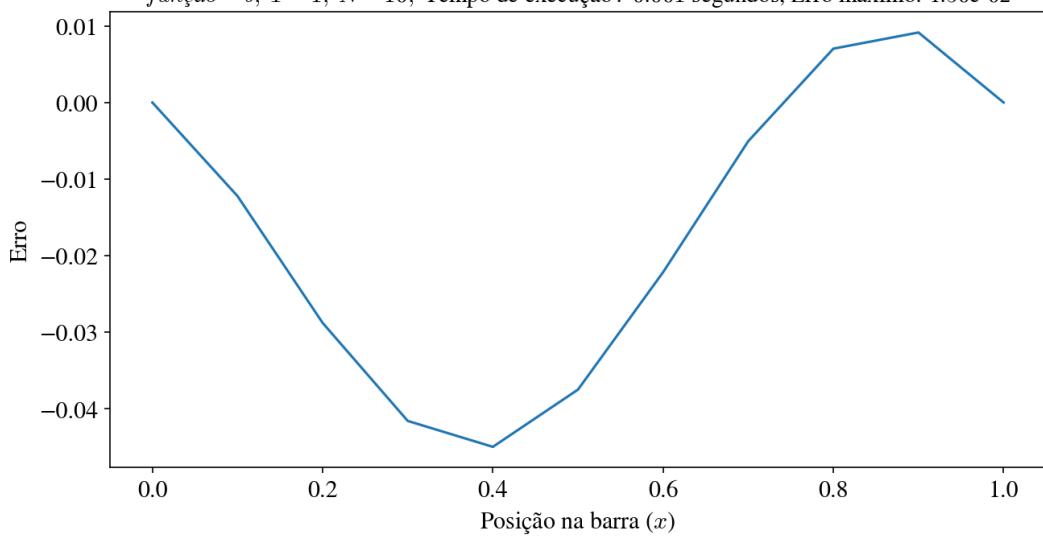
$função = b$, $T = 1$, $N = 320$, Tempo de execução : 0.348 segundos



Novamente, os valores concordam com os obtidos pelo Método de Euler, e aumentar o N suaviza a curva. Há também um grande decréscimo no tempo de execução, passando de 58.105 segundos para 0.348 segundos quando $N = 320$. Analisando o erro:

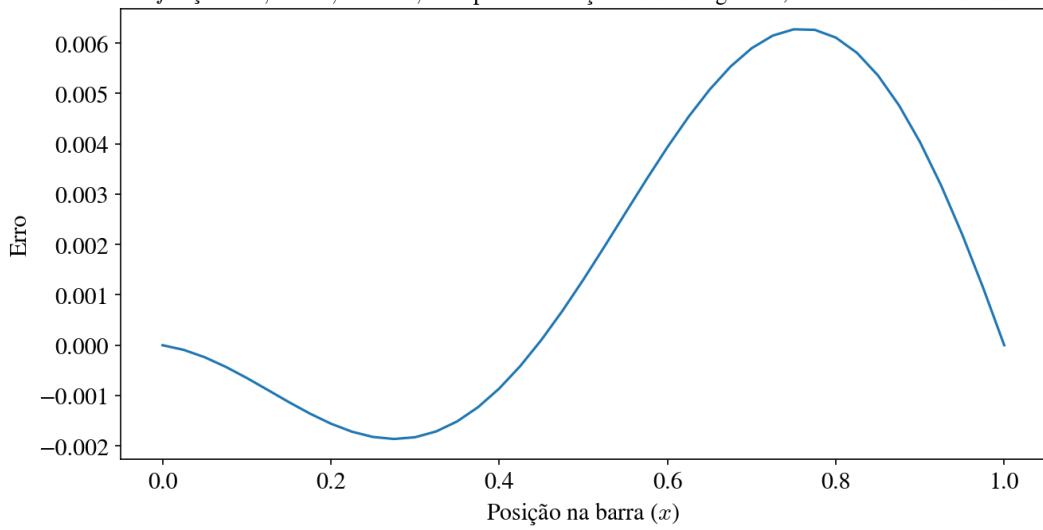
Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = b$, $T = 1$, $N = 10$, Tempo de execução: 0.001 segundos, Erro máximo: 4.50e-02

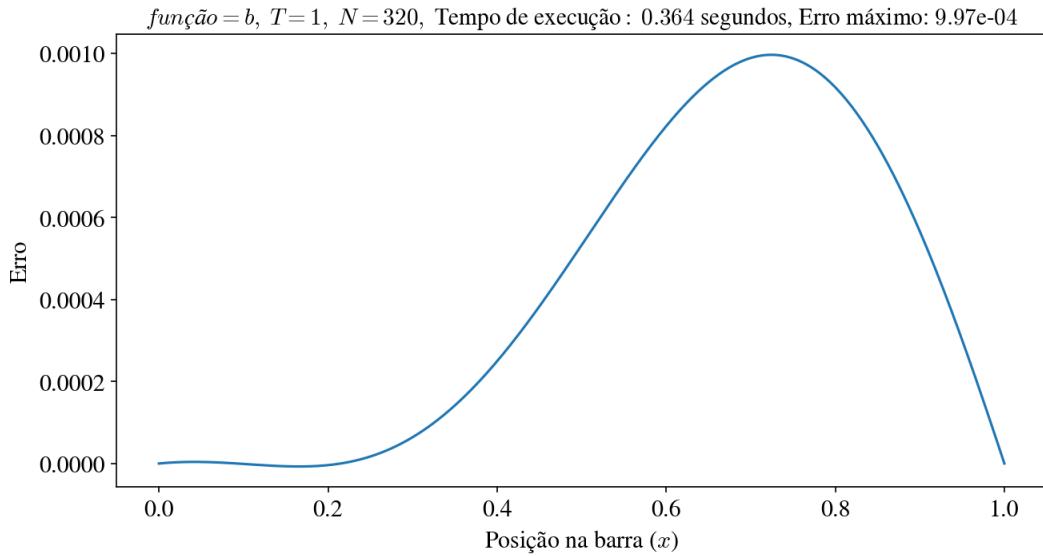


Erro em função da posição para o instante $t = T$ no método de Implicit Euler

$função = b$, $T = 1$, $N = 40$, Tempo de execução: 0.016 segundos, Erro máximo: 6.28e-03



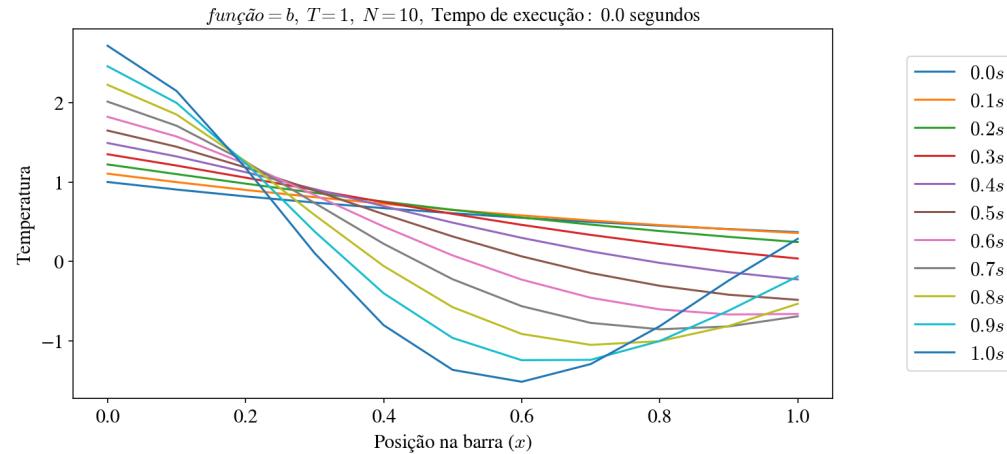
Erro em função da posição para o instante $t = T$ no método de Implicit Euler



Apesar do formato da curva não ser mantido, principalmente devido a natureza oscilatória da função, note que o erro máximo decresce com o aumento de N . Novamente, isso ocorre porque o erro máximo é limitado por uma função linear em Δt e quadrática em Δx , e aumentar N reduz ambos. Além disso, note que novamente a magnitude do erro é maior, em média, da observada no Método de Euler.

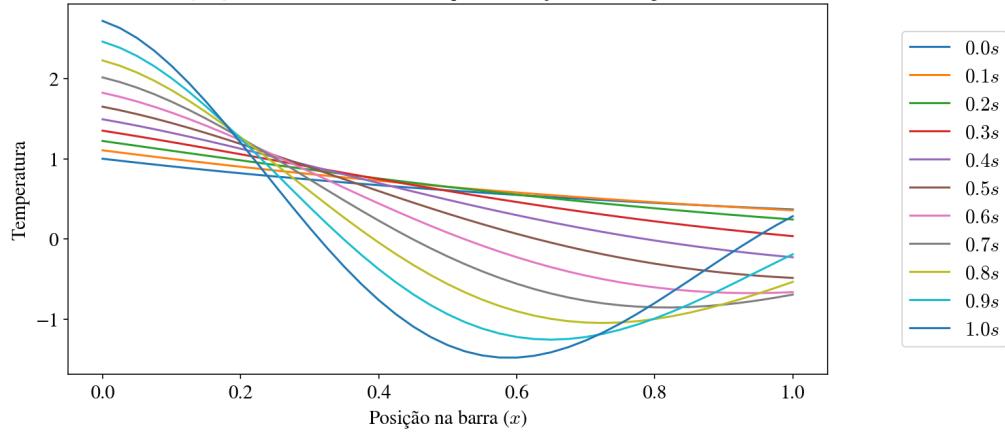
Por fim, com o método de Crank-Nicolson, obtém-se os seguintes gráficos:

Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson



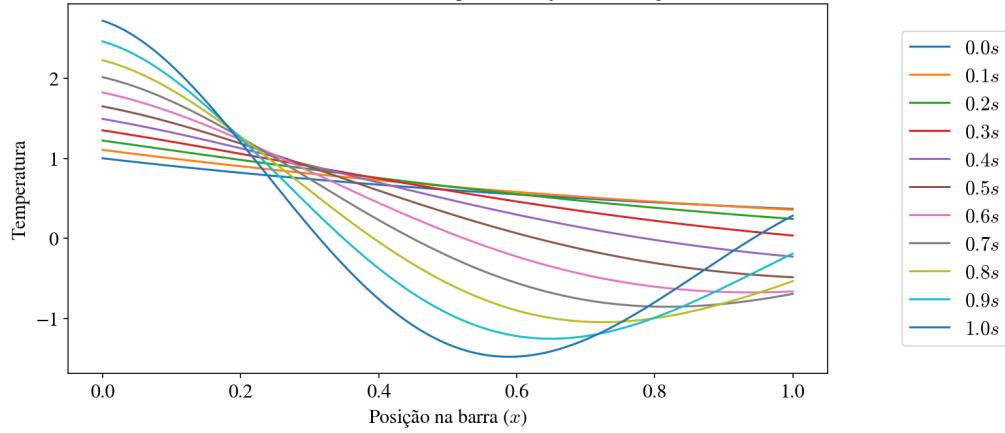
Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson

$função = b$, $T = 1$, $N = 40$, Tempo de execução : 0.016 segundos



Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson

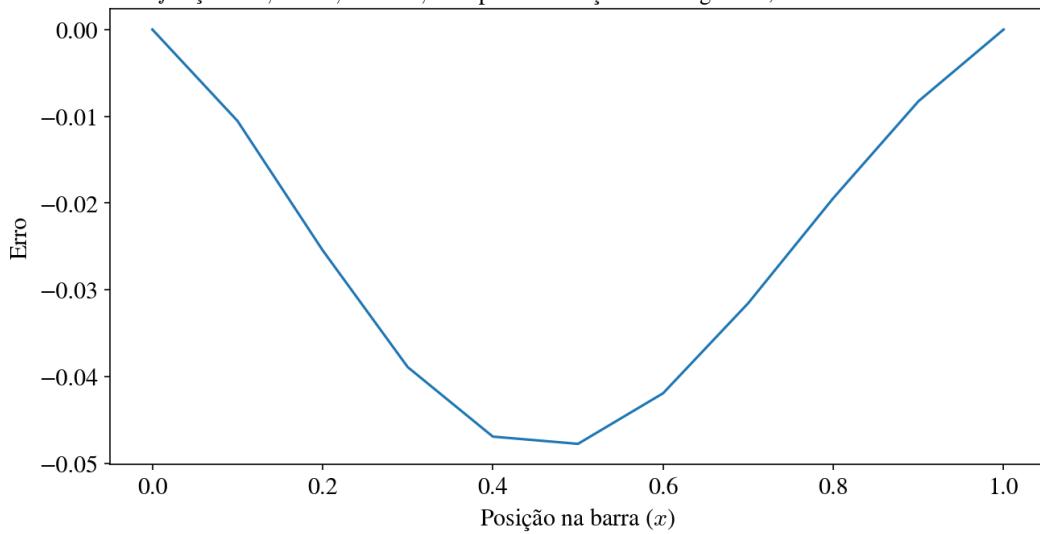
$função = b$, $T = 1$, $N = 320$, Tempo de execução : 0.346 segundos



Novamente, assim como no item a), os tempos de execução são similares aos do método implícito, assim como o formato da curva. Analisando o erro:

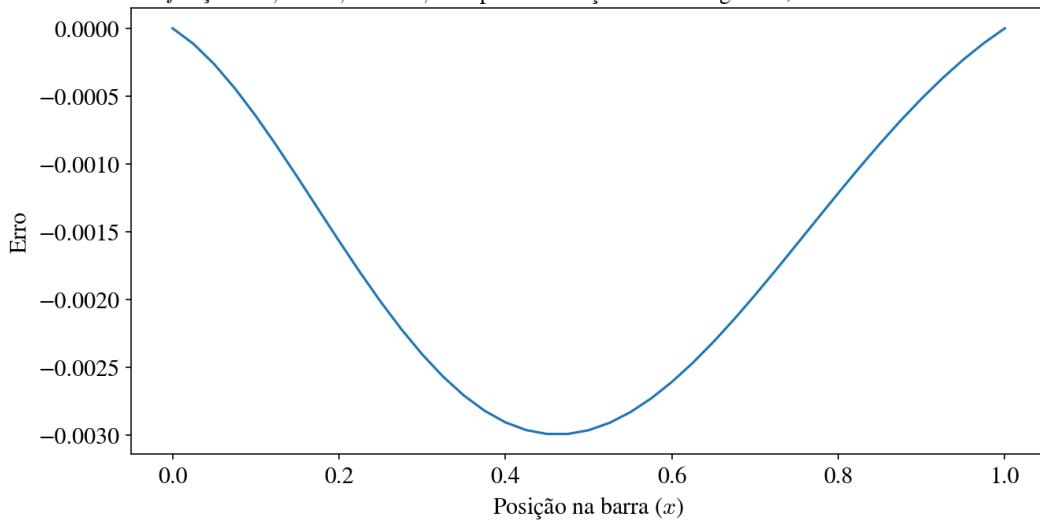
Erro em função da posição para o instante $t = T$ no método de Crank Nicolson

$função = b$, $T = 1$, $N = 10$, Tempo de execução: 0.0 segundos, Erro máximo: 4.78e-02



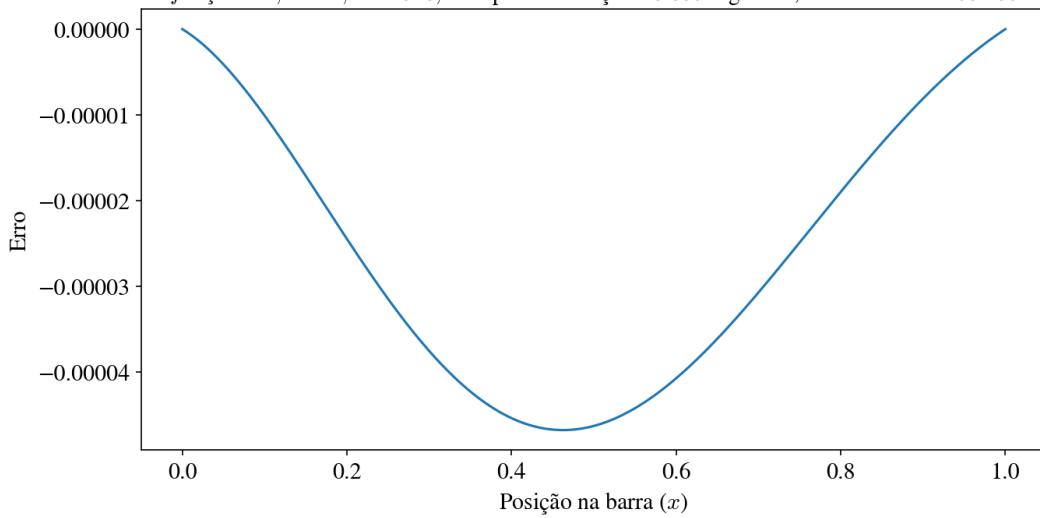
Erro em função da posição para o instante $t = T$ no método de Crank Nicolson

$função = b$, $T = 1$, $N = 40$, Tempo de execução: 0.017 segundos, Erro máximo: 2.99e-03



Erro em função da posição para o instante $t = T$ no método de Crank Nicolson

$função = b, T = 1, N = 320$, Tempo de execução : 0.359 segundos, Erro máximo: 4.68e-05



É possível observar que, novamente, a ordem de grandeza do erro se aproxima mais da obtida pelo Método de Euler, e menor que a encontrada no implícito, pelas razões citadas anteriormente.

6.3 Função c)

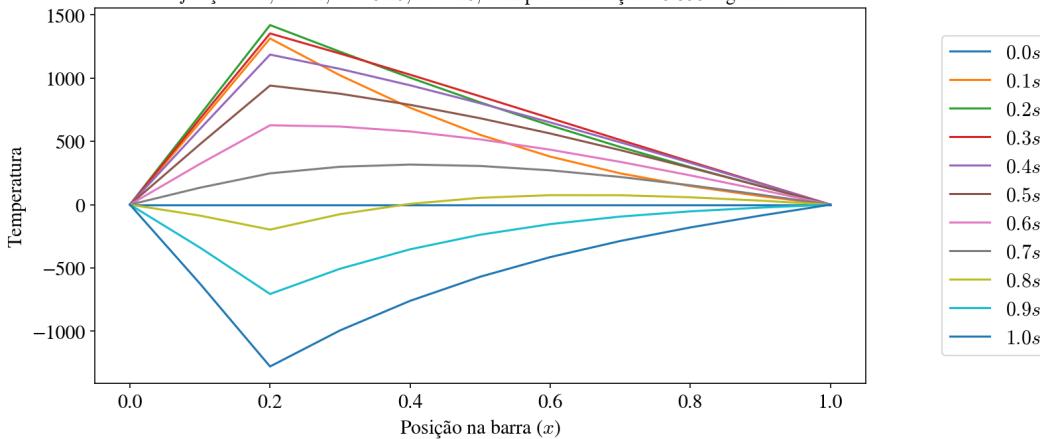
Define-se agora uma fonte pontual localizada em $p = 0.25$ cuja intensidade varia com o tempo t de acordo com a função:

$$r(t) = 10000(1 - 2t^2)$$

Sendo u_0 um vetor nulo, e $g_1(t) = g_2(t) = 0$. Novamente, variamos os valores de N e λ , mantendo-se um constante enquanto varia-se o outro parâmetro. Para $\lambda = 0.25$:

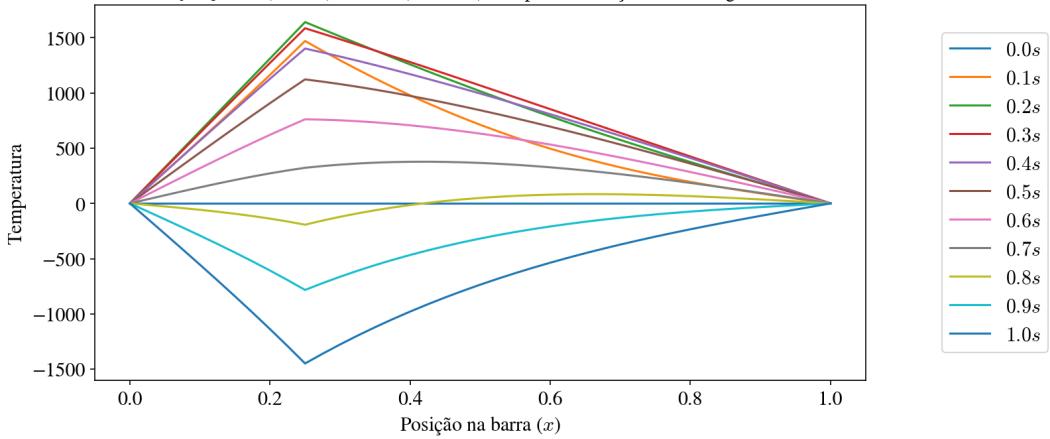
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = c, T = 1, \lambda = 0.25, N = 10$, Tempo de execução : 0.093 segundos



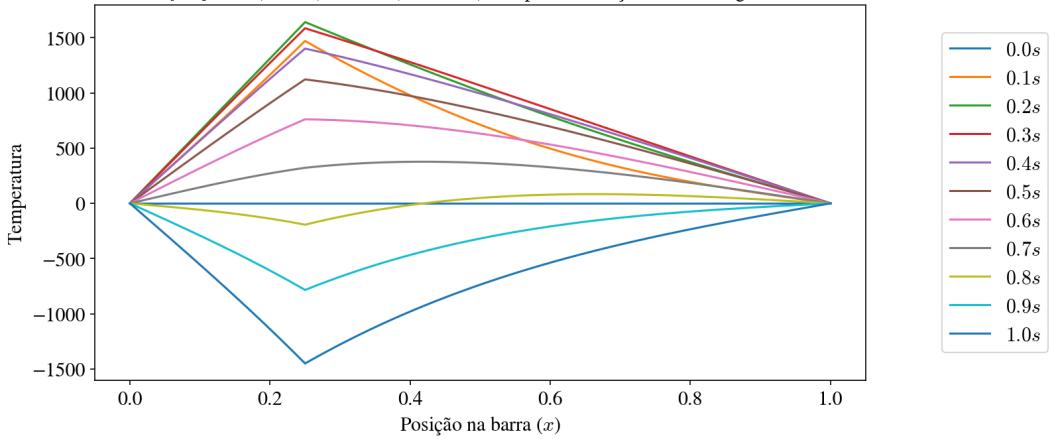
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = c, T = 1, \lambda = 0.25, N = 40$, Tempo de execução : 0.312 segundos



Temperatura em função da posição para certas séries temporais pelo método de Euler

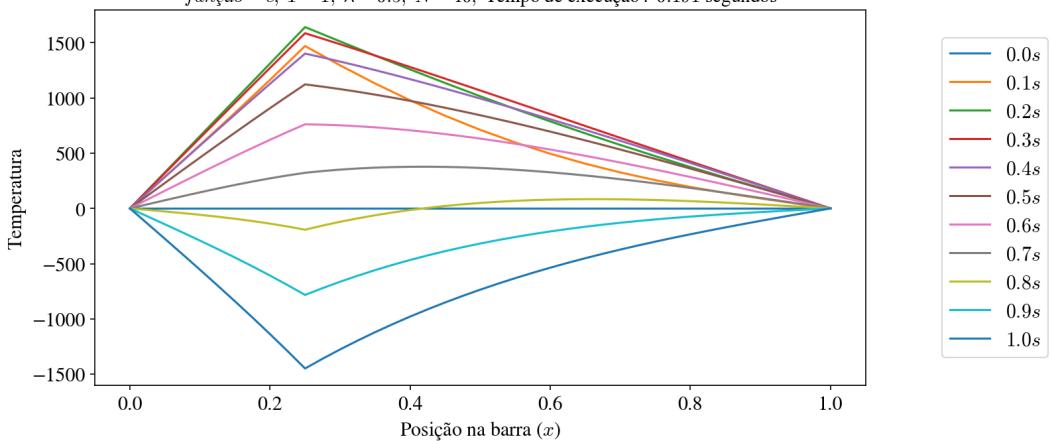
$função = c, T = 1, \lambda = 0.25, N = 320$, Tempo de execução : 50.796 segundos



Aparentemente, não há diferença entre os gráficos. Mas o que ocorre, novamente, é que a suavidade da curva aumenta, e o efeito só não é perceptível pois a magnitude das temperaturas é bem superior à dos gráficos anteriores. Mantendo $N = 40$:

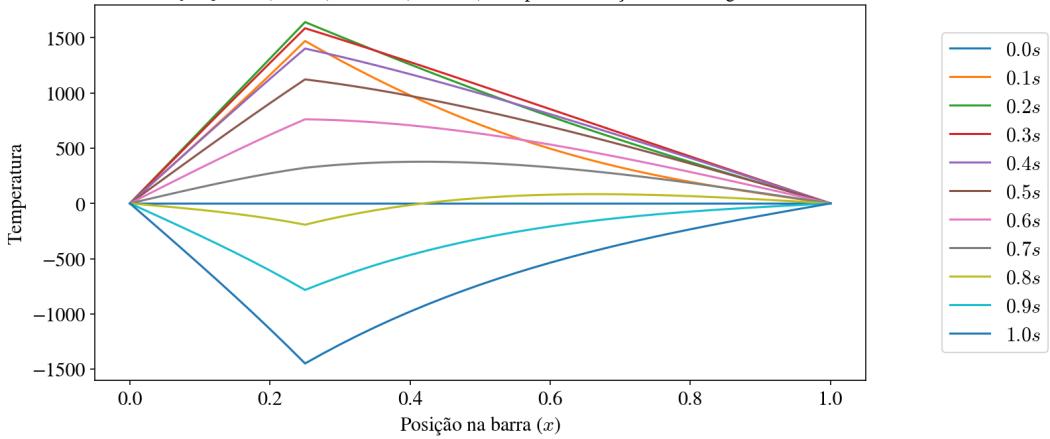
Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = c, T = 1, \lambda = 0.5, N = 40$, Tempo de execução : 0.191 segundos



Temperatura em função da posição para certas séries temporais pelo método de Euler

$função = c, T = 1, \lambda = 0.25, N = 40$, Tempo de execução : 0.312 segundos

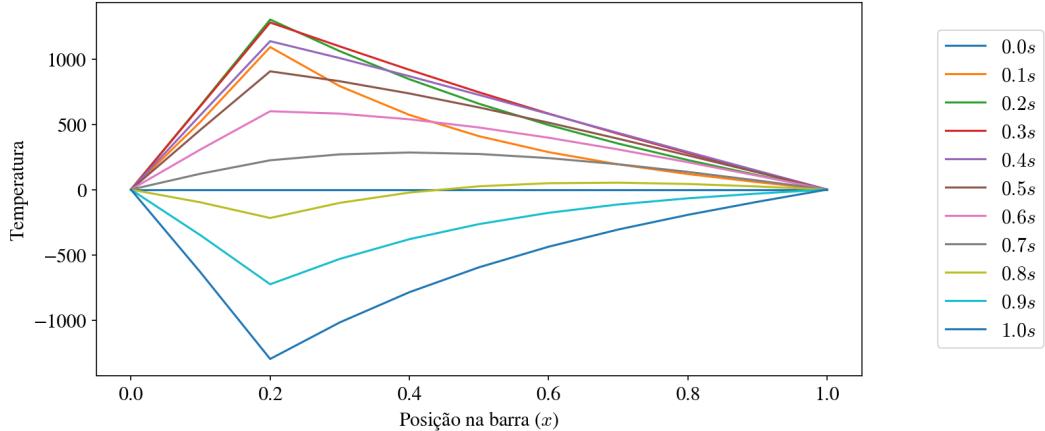


Nesse caso, não é relatada nenhuma mudança. Isto ocorre pois o λ é um parâmetro importante na obtenção do erro, que não é analisado neste item, pois não se tem em posse a solução exata para realizar-se alguma comparação.

Para o método implícito, variando N :

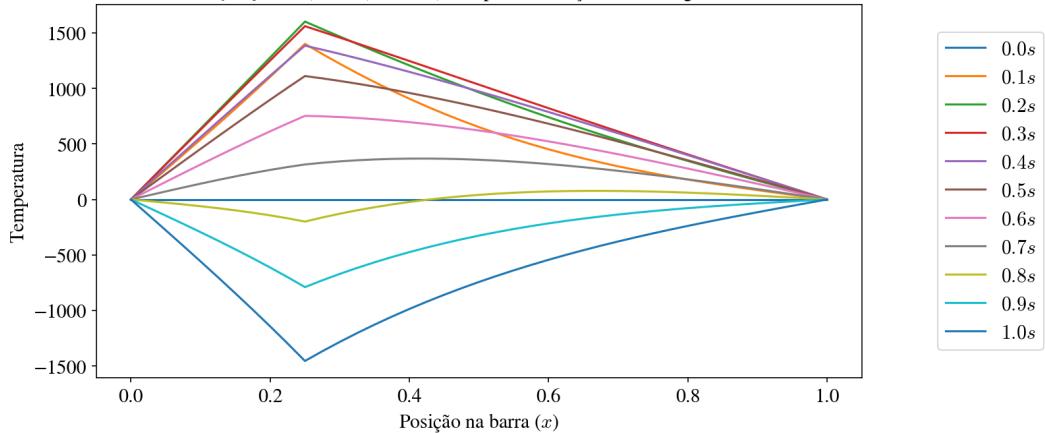
Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

$função = c, T = 1, N = 10$, Tempo de execução : 0.003 segundos

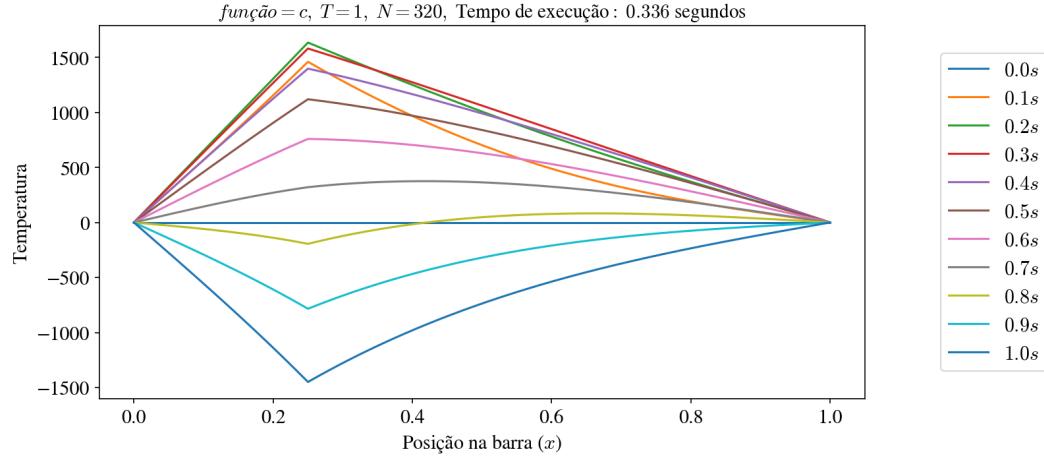


Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler

$função = c, T = 1, N = 40$, Tempo de execução : 0.016 segundos



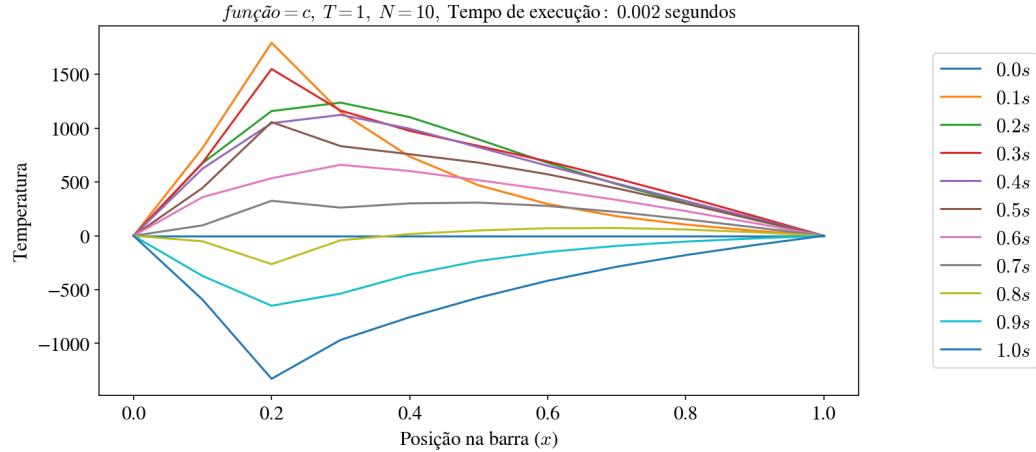
Temperatura em função da posição para certas séries temporais pelo método de Implicit Euler



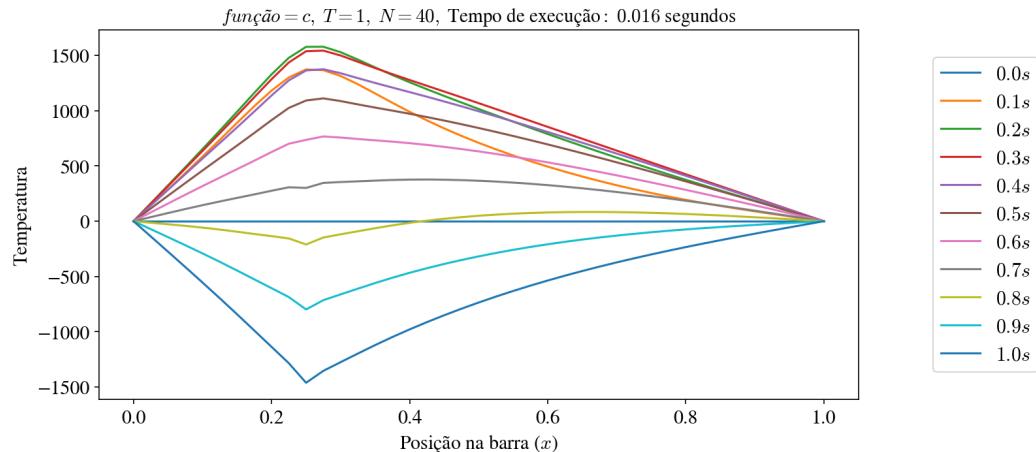
Como esperado, os resultados são mantidos em relação ao Método de Euler, e a mudança é o tempo de execução, que passa de 50.796 segundos para 0.336 segundos (para $N = 320$).

Finalmente, testando Crank-Nicolson:

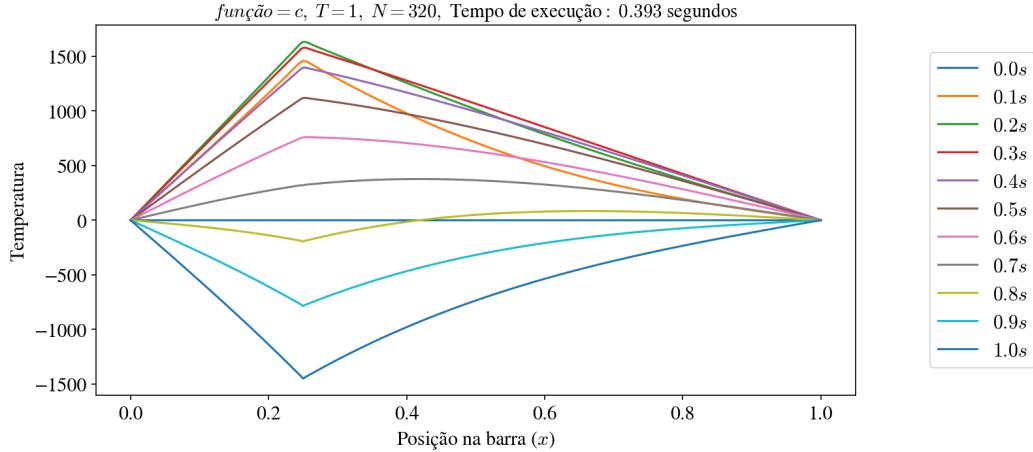
Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson



Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson



Temperatura em função da posição para certas séries temporais pelo método de Crank Nicolson



Notam-se tempos de execução similares aos do método implícito, para $N = 320$, e que o comportamento das curvas de ambos os métodos é similar para valores maiores de N (isto porque ambos os métodos possuem complexidade $\mathcal{O}(N)$). No entanto, para valores baixos de N (como $N=10$), o gráfico se distingue dos gráficos gerados pelos outros métodos. Isso se deve à forma na qual o Método de Crank-Nicolson calcula as temperaturas, considerando instantes de tempo intermediários ($t^k + 0.5\Delta t$). Para valores baixos de N , os instantes t^k e t^{k+1} estão consideravelmente separados, mas para valores maiores ficam mais próximos.

6.4 Fator de redução e Ordem de Convergência

O fator de redução ϕ é definido através da seguinte relação:

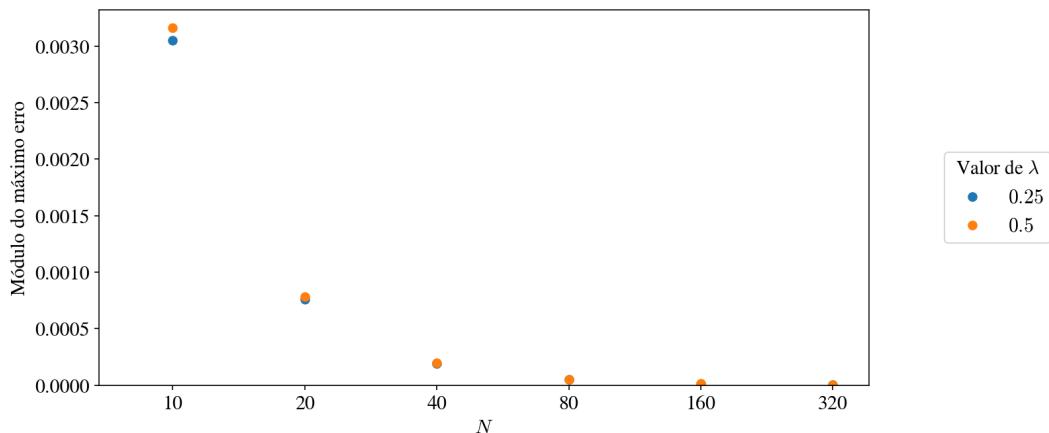
$$\phi = \frac{e_{2N}}{e_N}$$

Pode-se escolher diversos pares para e_{2N} e e_N , calculando-se então todas as possibilidades. O esperado é que o fator de redução se aproxime do valor previsto para o método conforme aumentamos N .

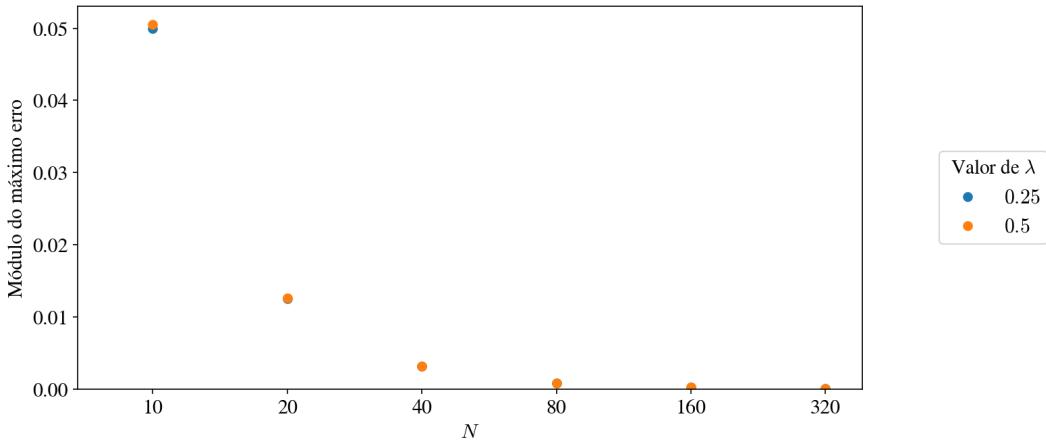
6.4.1 Método de Euler

Para o método de Euler, os gráficos do erro máximo em função de N estão expostos a seguir, para as funções a) e b).

Ordem de convergência do método Euler para a função a



Ordem de convergência do método Euler para a função b



Para a função a) e $\lambda = 0.25$, os valores de ϕ para os valores de N desde 10 até 160 são dados pela matriz a seguir:

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2485 \\ 0.2493 \\ 0.2503 \\ 0.2495 \\ 0.2508 \end{bmatrix}$$

Onde:

$$\phi_N = \frac{e_{2N}}{e_N}$$

Já para $\lambda = 0.50$:

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2481 \\ 0.2500 \\ 0.2495 \\ 0.2495 \\ 0.2500 \end{bmatrix}$$

Para a função b), com $\lambda = 0.25$:

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2500 \\ 0.2504 \\ 0.2498 \\ 0.2494 \\ 0.2508 \end{bmatrix}$$

E com $\lambda = 0.50$:

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2495 \\ 0.2500 \\ 0.2502 \\ 0.2500 \\ 0.2503 \end{bmatrix}$$

Note que todos os valores gravitam em torno de 0.25. Em média, para a função a), tem-se $\phi = 0.2497$ para $\lambda = 0.25$ e $\phi = 0.2494$ para $\lambda = 0.50$. Para a função b), tem-se $\phi = 0.2501$ para $\lambda = 0.25$ e $\phi = 0.2500$ para $\lambda = 0.50$.

A partir dos fatores de redução, é possível calcular a ordem de convergência do método a partir de:

$$\phi = 2^{-\mu_{Euler}}$$

$$\mu_{Euler} = -\log_2 \phi$$

Como todos os valores de λ se aproximam de 0.25, é possível afirmar que $\log_2 0.25 = -2$. Logo:

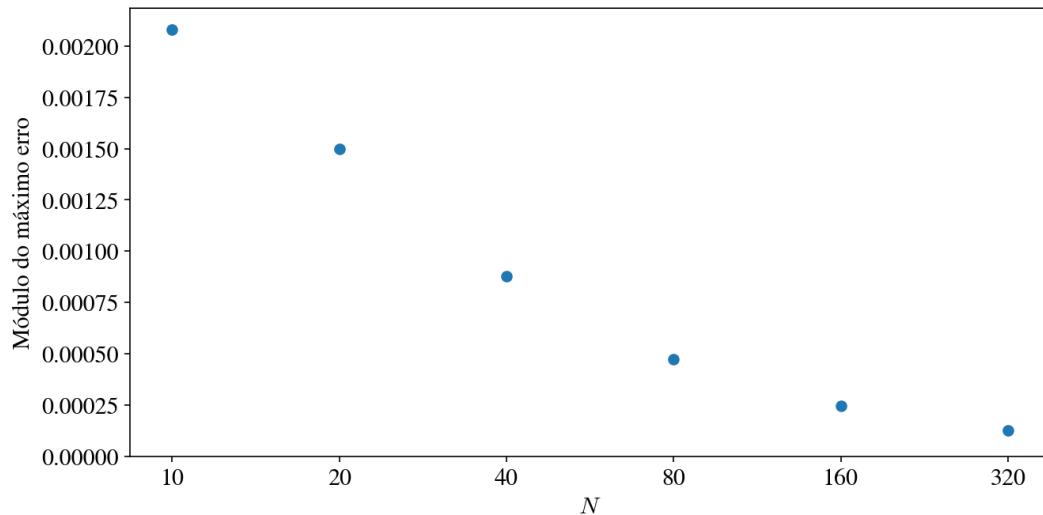
$$\mu_{Euler} = 2$$

Ou seja, a convergência é de segunda ordem, como previsto para o Método de Euler.

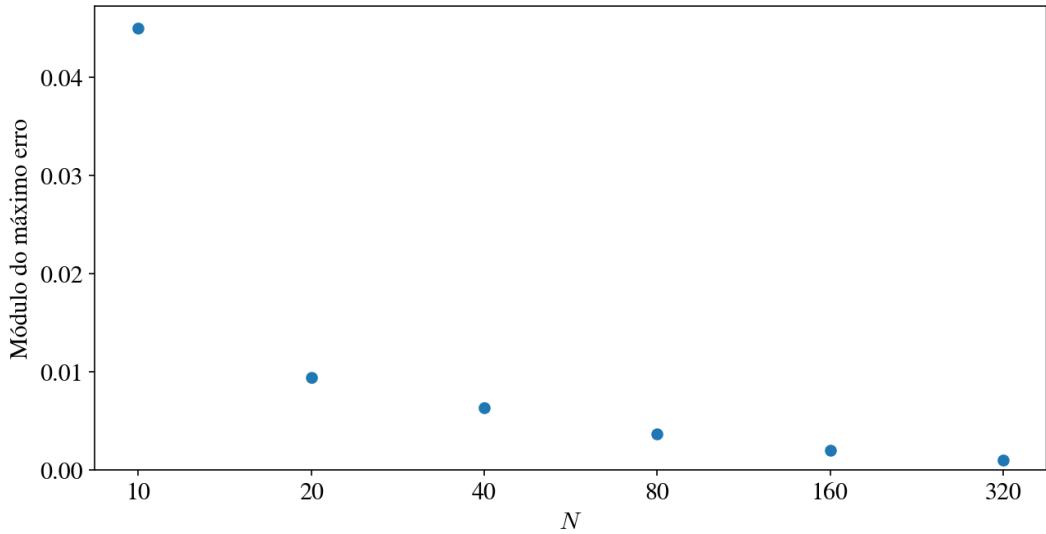
6.4.2 Método de Euler Implícito

Para o Método de Euler Implícito, os gráficos do erro máximo em função de N são mostrados a seguir, para as funções a) e b).

Ordem de convergência do método Implicit Euler para a função a



Ordem de convergência do método Implicit Euler para a função b



Para a função a), os valores do fator de redução para diferentes valores de N são mostrados a seguir:

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.7212 \\ 0.5860 \\ 0.5392 \\ 0.5190 \\ 0.5081 \end{bmatrix}$$

Para a função b):

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2082 \\ 0.6702 \\ 0.5748 \\ 0.5346 \\ 0.5166 \end{bmatrix}$$

Note que os valores se distanciam do esperado para o método, em boa parte por conta dos valores de ϕ quando $N = 10$ e $N = 20$. Pode-se atribuir isto à forte melhora da eficácia do método nos primeiros refinamentos, que reduz consideravelmente o erro. Nos refinamentos posteriores observa-se uma maior proximidade do valor esperado, $\phi = 0.50$.

Para calcular um fator de redução que se aproxime dos valores que obteria-se utilizando valores maiores de N (como $N = 640$, $N = 1280$, etc), desconsideram-se os valores dos *outliers* (ϕ_{10} e ϕ_{20}) e calcula-se a média dos demais.

Para a função a), o valor de redução médio é 0.5221, enquanto para a função b) é 0.5420. Embora ambos estejam próximos do valor esperado, 0.5, faz-se necessário calcular a ordem de convergência com a média destes valores, 0.5321:

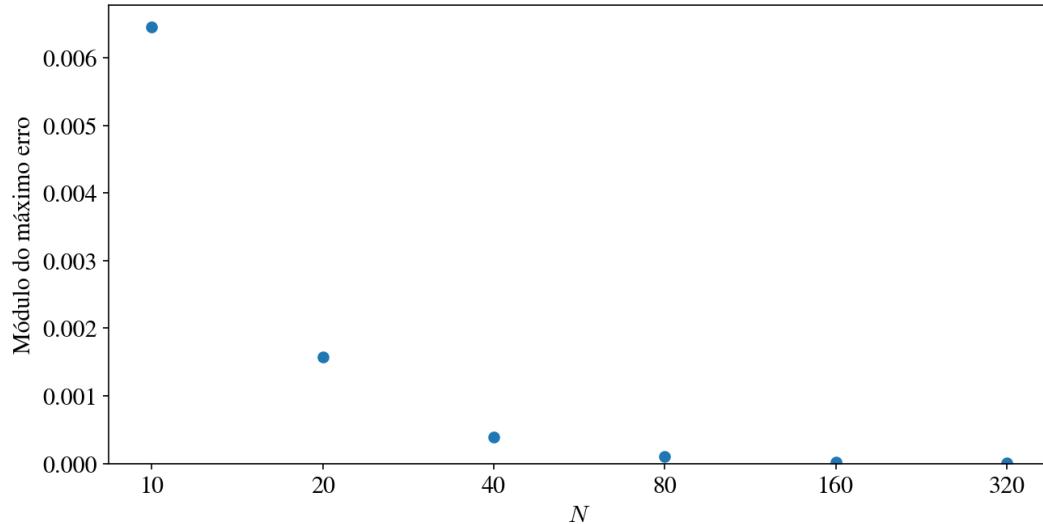
$$\mu_{IE} = -\log_2 0.5321 \Rightarrow \mu_{IE} = 0.9104$$

Apesar de estar relativamente distante do valor esperado para um método de ordem 1, isto pode ser atribuído aos baixos valores de N . Para refinamentos maiores, a tendência esperada é que a ordem converja para 1, pois é possível observar que o fator de redução apresentava tendência de se aproximar de 0.5.

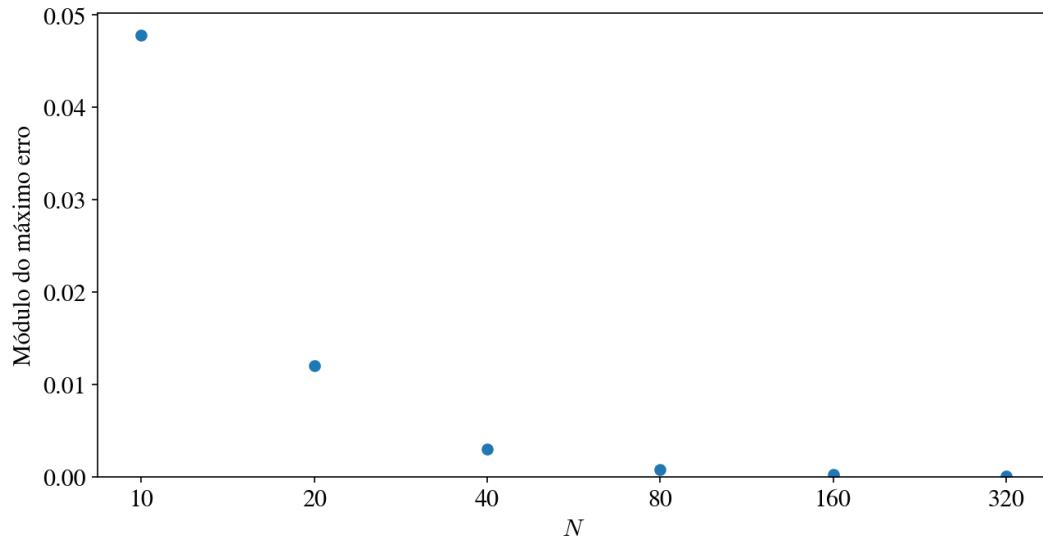
6.4.3 Método de Crank-Nicolson

Para o Método de Crank-Nicolson, os gráficos do erro máximo em função de N são mostrados a seguir, para as funções a) e b).

Ordem de convergência do método Crank Nicolson para a função a



Ordem de convergência do método Crank Nicolson para a função b



Para a função a), os valores do fator de redução para diferentes valores de N são mostrados a seguir:

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2434 \\ 0.2478 \\ 0.2499 \\ 0.2500 \\ 0.2498 \end{bmatrix}$$

Para a função b):

$$\begin{bmatrix} \phi_{10} \\ \phi_{20} \\ \phi_{40} \\ \phi_{80} \\ \phi_{160} \end{bmatrix} = \begin{bmatrix} 0.2510 \\ 0.2492 \\ 0.2505 \\ 0.2497 \\ 0.2503 \end{bmatrix}$$

Tal como no Método de Euler, os valores de ϕ gravitam em torno de 0.25. A média de ϕ para a função a) é de 0.2482, e para o teste b) é de 0.2501. Novamente, ambos os valores se aproximam de 0.25, e portanto é justo concluir que:

$$\mu_{CN} = 2$$

Os valores dos fatores de redução estão resumidos na tabela a seguir:

	Euler		Euler Implícito	Crank-Nicolson
	$\lambda = 0.25$	$\lambda = 0.5$		
função a	0.2497	0.2494	0.5221	0.2482
função b	0.2501	0.2500	0.5420	0.2501
ordem	2	2	1	2

Tabela 1: Fator de redução do erro e ordem de convergência para cada método e função de fonte

7 Conclusões e considerações finais

É possível observar que os gráficos obtidos convergem fortemente para a solução analítica, qualitativamente e quantitativamente, conforme refina-se a malha. Foi provado, para cada um dos métodos, sua convergência, experimentalmente, a partir dos fatores de redução calculados. Em suma, constatou-se a eficácia de métodos numéricos na resolução de equações diferenciais parciais, que surgem em diversas áreas da engenharia, porém nem sempre possuem uma solução analítica disponível. O conhecimento destes métodos numéricos prova-se indispensável nestas situações, e o exercício deles nesta tarefa será de grande valor.

Referências

- [1] Álvaro L. [De Bortoli], Greice S.L. Andreis e Felipe N. Pereira. “Chapter 6 - Numerical Methods for Reactive Flows”. Em: *Modeling and Simulation of Reactive Flows*. Ed. por Álvaro L. [De Bortoli], Greice S.L. Andreis e Felipe N. Pereira. Elsevier, 2015, pp. 123–169. ISBN: 978-0-12-802974-9. DOI: <https://doi.org/10.1016/B978-0-12-802974-9.00006-4>. URL: <http://www.sciencedirect.com/science/article/pii/B9780128029749000064>.
- [2] E. Love e W.J. Rider. “On the convergence of finite difference methods for PDE under temporal refinement”. Em: *Computers Mathematics with Applications* 66.1 (2013), pp. 33–40. ISSN: 0898-1221. DOI: <https://doi.org/10.1016/j.camwa.2013.04.019>. URL: <http://www.sciencedirect.com/science/article/pii/S0898122113002265>.
- [3] Jeremy Orloff. “The Heat Equation”. Notas de aula. Mar. de 2018. URL: <http://web.mit.edu/jorloff/www/18.03-esg/notes/heatequation.pdf>.
- [4] Janet Peterson. “Crank Nicolson Scheme for the Heat Equation”. Notas de aula. Nov. de 2017. URL: <https://people.sc.fsu.edu/~jpeterson/5-CrankNicolson.pdf>.