# ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

---

MAP 3121 – MÉTODOS NUMÉRICOS E APLICAÇÕES

## EXERCÍCIO PROGRAMA I

UM PROBLEMA INVERSO PARA OBTENÇÃO DE DISTRIBUIÇÃO DE TEMPERATURA


Adriano Carvalho e Sousa (Turma 1) - 10773291

Maria Tereza Ferreira Silva (Turma 6) - 10769726


SÃO PAULO

2020

# SUMÁRIO

# 1. INTRODUÇÃO

Buscando aplicar os conceitos abordados pela disciplina de Cálculo Numérico, são propostos dois exercícios programa. Neste trabalho, dividido em duas partes, serão aplicados métodos numéricos para a resolução de um problema ligado à equação do calor.

Esta primeira parte, contemplada neste EP 1, refere-se ao problema direto, e será visto como determinar a evolução da distribuição de temperatura em uma barra sujeita a fontes de calor, a partir de uma dada distribuição inicial e condições de contorno.

Problemas diretos são, em geral, bem-postos. De acordo com Jacques Hadamard (1865-1963), um problema é bem-posto se ele satisfaz três condições:

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

Por outro lado, problemas inversos são frequentemente mal-postos, no sentido que eles não satisfazem pelo menos uma das hipóteses acima, e a falta de informações suficientes para encontrar a causa do efeito é um dos motivos. Sendo assim, devido à maior complexidade, será abordado no EP2.

Para este exercício computacional, foi utilizada a linguagem Python, na versão 3.7.6, com as bibliotecas auxiliares Matplotlib para os gráficos e NumPy para operações matemáticas. O programa foi escrito usando o ambiente de desenvolvimento Spyder.

## 2. CONTEXTUALIZAÇÃO

A evolução da distribuição de temperatura em uma barra é dada pela seguinte equação diferencial parcial:

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

Considerando que *t* é a variável temporal, *x* a variável espacial e empregando a notação compacta para derivadas parciais.

O comprimento da barra foi normalizado para 1 e a equação será integrada num intervalo de tempo de 0 a T. A variável u(t,x) descreve a temperatura no instante *t* na posição *x*, sendo:

$$u(0,x) = u_0(x)\ em\ [0,1] \qquad (2)$$
$$u(t,0) = g_1(t)\ em\ [0,T] \qquad (3)$$
$$u(t,1) = g_2(t)\ em\ [0,T] \qquad (4)$$

A função *f* descreve as fontes de calor ao longo do tempo.

O principal objetivo desse trabalho é resolver tal equação a partir de métodos numéricos que serão descritos a seguir e em seguida avaliar os erros ao adotar tais métodos.

# 3. METODOLOGIA

Para resolver esse problema, usamos o método da aproximação numérica das derivadas parciais por diferenças finitas. Estas são baseadas em expansões de Taylor. São elas:

$$u_t(t,\ x) \ = \ \frac{u(t+\Delta t, x) - u(t, x)}{\Delta t} - \Delta t \ \frac{u_{tt}(\bar{t}, x)}{2} \qquad (5)$$

$$u_{xx}(t,\ x) \ = \ \frac{u(t, x-\Delta x) - 2u(t, x) + u(t, x+\Delta x)}{\Delta x^2} - \Delta x^2 \ \frac{u_{xxxx}(t, \bar{x})}{4!} \qquad (6)$$

Em que $\bar{t}$ é um valor entre t e t + $\Delta$t e $\bar{x}$ é um ponto entre x e x + $\Delta$x. Nas expressões para $u_t(t,\ x)$ e $u_{xx}(t,\ x)$, dadas acima, os erros são proporcionais a $\Delta$t e $\Delta x^2$, respectivamente. Se estes incrementos tenderem a zero, é obtida a convergência das aproximações por diferenças finitas para as derivadas parciais correspondentes.

Para a discretização da equação do calor têm-se uma malha espacial dada pelos pontos $x_i = i\Delta x\ ;\ i \ = \ 0;\ \cdots ;\ N,\ com\ \Delta x \ = \ 1/N.$ Para a discretização temporal definimos $\Delta t \ = \ T/M$, e calculamos aproximações nos instantes $t_k \ = \ k\Delta t;\ k \ = \ 1;\ \cdots ;\ M.$ Denotamos a aproximação para a solução nos pontos de malha u( $t_k$ ; $x_i$ ) por $u_i^k$.

Desta forma teremos a condição inicial dada por:

$$u_i^{\ 0} = u_0(x_i);\ i \ = \ 0;\ \cdots ;\ N$$

Ao longo da evolução temporal as condições de fronteira são dadas por:

$$u_0^k \ = g_1(t_k)\ e\ u_k^N \ = g_2(t_k);\ k \ = \ 1;\ \cdots ;\ M$$

### 3.1. Método de Euler explícito

Utilizando a metodologia da discretização descrita acima, para os pontos interiores a evolução é aproximada por:

$$u_i^{k+1} = u_i^k + \Delta t(\frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2} \ + f(x_{i,}\ t_k))\ ,\ \ i \ = 1, ..., \ N-1,\ e\ k = 0, ..., \ M-1. \qquad (7)$$

No código foi implementado um laço que calcula esse método.

### 3.2. Método de Euler implícito

Em um método implícito, a solução em um ponto de malha no novo instante depende também de outros valores no mesmo instante. Esta interdependência dos valores no novo instante leva à necessidade de resolver um sistema de equações a cada passo no tempo.

$$u_i^{k+1} = u_i^k + \lambda(u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) + \Delta t f(x_i, t_{t+1}), \; i = 1, \ldots, N-1, \; e \; k = 0, \ldots, M-1. \quad (8)$$

Neste método, para a evolução temporal, é necessário resolver a cada passo um sistema linear com uma matriz A tridiagonal simétrica, como segue:

$$\begin{bmatrix} 1+2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\lambda & 1+2\lambda & -\lambda \\ 0 & \cdots & 0 & -\lambda & 1+2\lambda \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \begin{bmatrix} u_1^k + \Delta t f_1^{k+1} + \lambda g_1(t^{k+1}) \\ u_2^k + \Delta t f_2^{k+1} \\ \vdots \\ u_{N-2}^k + \Delta t f_{N-2}^{k+1} \\ u_{N-1}^k + \Delta t f_{N-1}^{k+1} + \lambda g_2(t^{k+1}) \end{bmatrix} \quad (9)$$

### 3.3. Método de Crank-Nicolson

O método é da forma:

$$u_i^{k+1} = u_i^k + \frac{\lambda}{2}((u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) + (u_{i-1}^k - 2u_i^k + u_{i+1}^k)) + \frac{\Delta t}{2}(f(x_i, t_k) + f(x_i, t_{k+1})) \quad (10)$$

Analogamente ao método implícito, deve-se resolver um sistema linear. Este envolverá também uma matriz tridiagonal simétrica, como no método anterior, trocando λ por λ/2. Este é um esquema de segunda ordem, centrado no tempo intermediário $t_k + 0.5\Delta$.

$$\begin{bmatrix} 1+\lambda & -\frac{\lambda}{2} & 0 & \cdots & 0 \\ -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} \\ 0 & \cdots & 0 & -\frac{\lambda}{2} & 1+\lambda \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \begin{bmatrix} u_1^k + \frac{\lambda}{2}(u_0^k - 2u_1^k + u_2^k) + \frac{\Delta t}{2}(f_1^k + f_1^{k+1}) + \frac{\lambda}{2}g_1(t^{k+1}) \\ u_2^k + \frac{\lambda}{2}(u_1^k - 2u_2^k + u_3^k) + \frac{\Delta t}{2}(f_2^k + f_2^{k+1}) \\ \vdots \\ u_{N-2}^k + \frac{\lambda}{2}(u_{N-3}^k - 2u_{N-2}^k + u_{N-1}^k) + \frac{\Delta t}{2}(f_{N-2}^k + f_{N-2}^{k+1}) \\ u_{N-1}^k + \frac{\lambda}{2}(u_{N-2}^k - 2u_{N-1}^k + u_N^k) + \frac{\Delta t}{2}(f_{N-1}^k + f_{N-1}^{k+1}) + \frac{\lambda}{2}g_2(t^{k+1}) \end{bmatrix} \quad (11)$$

Em ambos os métodos implícitos explicados, é necessária a resolução de um sistema tridiagonal simétrico. Para resolver o sistema, foi utilizada a decomposição $A = LDL^t$ da matriz do sistema tridiagonal em questão, onde a matriz L é bidiagonal triangular unitária inferior e D diagonal.

### 3.4. Cálculo do erro

#### 3.4.1. Erro na aproximação

O erro de aproximação é definido como a diferença entre a solução exata e a solução obtida através dos métodos aqui descritos. Isso significa que, quanto mais a solução numérica se aproxima da exata, menor será esse erro, como descrito na seguinte equação:

$$e_i^k = u(t_k, x_i) - u_i^k \qquad (12)$$

Ainda, a norma do erro em determinado instante é definida como o valor máximo do módulo dos erros calculados nesse instante:

$$||e^k|| = max_i |e_i^k| \qquad (13)$$

Nesse EP, o erro só será avaliado em T=1. É importante ressaltar que o erro de aproximação deve ser delimitado pelo erro de truncamento, explicado na sequência, para que a solução possa convergir.

### 3.4.2. Erro de truncamento

O erro de truncamento resulta de aproximações para representar procedimentos matemáticos exatos, e está associado ao método de aproximação empregado. Ele surge cada vez que se substitui um processo matemático infinito por um processo finito ou discreto.

- Método de Euler explícito

No método explícito, o erro local de truncamento, que mede quão bem a solução exata u(t,x) e satisfaz à equação discretizada (na forma dividida por $\Delta$t), é dado por:

$$\tau_i^k(\Delta t, \Delta x) = \frac{u(t_{k+1}, x_i) - u(t_k, x_i)}{\Delta t} - \frac{u(t_k, x_{i-1}) - 2u(t_k, x_i) + u(t_k, x_{i+1})}{\Delta x^2} - f(x_i, t_k) \qquad (14)$$

$$= \frac{u_t(t_k, x_i) + \Delta t\, u_{tt}(\overline{t_k}, x_i)}{2} - u_{xx}(t_k, x_i) - \Delta x^2 \frac{u_{xxxx}(t_k, \overline{x_i})}{4!} - f(x_i, t_k) \qquad (15)$$

$$= \Delta t \frac{u_{tt}(\overline{t_k}, x_i)}{2} - \Delta x^2 \frac{u_{xxxx}(t_k, \overline{x_i})}{4!} \qquad (16)$$

Considerando a hipótese de u(t, x) ter 4 derivadas contínuas em x e duas em t no domínio de integração D = [0, T] × [0, 1] pode-se delimitar:

$$\tau(\Delta t, \Delta x) = max_{k,i} |\tau_i^k(\Delta t, \Delta x)| \leq C1\Delta t + C2\Delta x^2 \qquad (17)$$

$$\text{com } C1 = max_D \left| \frac{u_{tt}(t, x)}{2} \right| \text{ e } C2 = max_D \left| \frac{u_{xxxx}(t, \overline{x})}{4!} \right|$$

$$\text{e, por consequência, } \lim_{\Delta t, \Delta x \to 0} \tau(\Delta t, \Delta x) = 0 \qquad (18)$$

- Método de Euler implícito

No método de Euler implícito, de forma análoga, o erro de truncamento é definido como:

$$\tau_i^k(\Delta t, \Delta x) = \frac{u(t_{k+1}, x_i) - u(t_k, x_i)}{\Delta t} - \frac{u(t_{k+1}, x_{i-1}) - 2u(t_{K+1}, x_i) + u(t_{k+1}, x_{i+1})}{\Delta x^2} - f(x_i, t_{k+1}) \qquad (19)$$

- Método de Crank-Nicolson

No método de Crank-Nicolson, de forma análoga, o erro de truncamento é definido como:

$$\tau_i^k(\Delta t,\ \Delta x) = \frac{u(t_{k+1}, x_i) - u(t_k, x_i)}{\Delta t} - \frac{(u(t_{k+1}, x_{i-1}) - 2u(t_{K+1}, x_i) + u(t_{k+1}, x_{i+1})) + (u(t_k, x_{i-1}) - 2u(t_k, x_i) + u(t_k, x_{i+1}))}{2\Delta x^2} - \frac{f(x_i, t_k) + f(x_i, t_{k+1})}{2} \quad (20)$$

### 3.5. Convergência da solução

- Método de Euler explícito

No caso do método da equação discretizada, para que a solução aproximada nos pontos de malha convirja para a solução exata da equação do calor, é estabelecida uma condição que garante que isto ocorre. Inicialmente define-se o erro entre a solução aproximada e a exata como na equação (12):

$$e_i^k = u(t_k, x_i) - u_i^k$$

Combinando a equação para a determinação da solução aproximada (7) e a equação definindo o erro local de truncamento (14) para obter a seguinte equação para o erro:

$$e_i^{k+1} = e_i^k + \Delta t\left(\frac{e_{i-1}^k - 2e_i^k + e_{i+1}^k}{\Delta x^2} + \tau_i^k\right),\ i = 1, \cdots, N-1,\ e\ k = 0, \cdots, M-1\ . \quad (21)$$

A norma do erro no instante $t_k$ é definida pela equação (13). Na sequência, estima-se como o erro evolui em função do tempo. Vamos denominar $\lambda = \Delta t/\Delta x^2$. Então, da equação do erro (21), obtêm-se que:

$$\left|e_i^{k+1}\right| \le |1 - 2\lambda||e_i^k| + |\lambda|(|e_{i-1}^k| + |e_{i+1}^k|) + \Delta t|\tau_i^k| \quad (22)$$

$$\le (|1 - 2\lambda| + 2|\lambda|)\|e^k\| + \Delta t\tau(\Delta t,\ \Delta x) \quad (23)$$

Se λ ≤ 1/2 então 1 − 2λ ≥ 0, já que e λ é sempre positivo. Assim, para esta escolha de λ:

$$|e_i^{k+1}| \le ((1 - 2\lambda) + 2\lambda)\|e^k\| + \Delta t\tau(\Delta t,\ \Delta x) \quad (24)$$

$$\le \|e^k\| + \Delta t\tau(\Delta t,\ \Delta x), \text{ e segue que } \|e^{k+1}\| \le \|e^k\| + \Delta t\tau(\Delta t,\ \Delta x)\ . \quad (25)$$

Usando esta última estimativa recursivamente, é obtido que:

$$\|e^{k+1}\| \leq \|e^{k-1}\| + 2\Delta t\tau(\Delta t, \Delta x) \quad (26)$$

$$\leq \|e^{0}\| + (k+1)\Delta t\tau(\Delta t, \Delta x) = t_{k+1}\tau(\Delta t, \Delta x) \quad (27)$$

$$\leq T(C1\Delta t + C2\Delta x^2). \quad (28)$$

Nesta última estimativa, considera-se ainda que o erro inicial $\|e^0\|$ é nulo, uma vez que $u_0(x)$ é dado, e a delimitação do erro de truncamento (17). Esta estimativa mostra, que se $\Delta$t e $\Delta$x tenderem a zero o erro também vai a zero e, portanto a aproximação calculada converge para a solução exata da equação. Utiliza-se, ainda, a hipótese que:

$$\lambda = \frac{\Delta t}{\Delta x^2} \leq 1/2. \quad (29)$$

Isso porque o método da discretização é dito condicionalmente convergente, e, para obter convergência, $\Delta$t deve ser da ordem de $\Delta x^2$. Portanto, se a condição (29) estiver satisfeita para $\Delta t = \alpha\Delta x^2$, com $\alpha \leq 1/2$, então o erro fica menor que uma constante vezes $\Delta x^2$. O método é (condicionalmente) convergente de ordem 2 em $\Delta$x.

- Método de Euler implícito

No caso do método de Euler implícito, a definição do erro de truncamento e a análise da convergência são análogas ao do caso explícito. Porém, nesse caso não foi preciso fazer qualquer restrição na escolha de $\Delta$t e $\Delta$x. O método de Euler implícito é incondicionalmente estável e convergente de ordem 2 em $\Delta$x e ordem 1 em $\Delta$t. Assim, mesmo não sofrendo restrições de estabilidade, a precisão do esquema estaria limitada pela escolha de $\Delta$t.

- Método de Crank-Nicolson

Já no caso do método de Crank-Nicolson, que também é incondicionalmente estável, a convergência é de ordem 2 em $\Delta$x e $\Delta$t. As notações anteriores são as mesmas, mas como este é um esquema de segunda ordem, centrado no tempo intermediário $t_k + 0.5\Delta$, a análise da convergência é, no entanto, mais complexa que as anteriores, envolvendo os autovalores da Matriz do sistema.

# 4. PRIMEIRA TAREFA

A primeira tarefa pede a implementação do método explícito. Dividiremos a resolução em quatro casos, assim como foi feito no código. As condições iniciais e de contorno, bem como as funções da fonte de calor variam em cada caso, por isso, serão detalhadas em cada um deles.

Levando em conta que o enunciado solicita que os valores de M e N sejam escolhidos em tempo de execução e fornece valores $\lambda$ e de N para os testes, devemos efetuar o cálculo do valor de M a partir da fórmula:

$$M = \frac{N^2 T}{\lambda} \qquad (30)$$

O resultado, em alguns casos, pode resultar em um número não inteiro, caso isso ocorra, deve-se arredondar para o menor número inteiro que seja maior que o resultado não inteiro obtido, como pode ser verificado na tabela 1 com λ = 0,51.

Sendo assim, podemos obter os resultados que serão utilizados como referência para as entradas nos testes a serem executados em cada caso dessa tarefa.

Considerando T=1, para alguns valores de N e $\lambda$, podem ser encontrados na tabela abaixo:

| | Valor de M | | |
|---|---|---|---|
| N | λ = 0,25 | λ = 0,50 | λ = 0,51 |
| 10 | 400 | 200 | 197 |
| 20 | 1600 | 800 | 785 |
| 40 | 6400 | 3200 | 3138 |
| 80 | 25600 | 12800 | 12550 |
| 160 | 102400 | 51200 | 50197 |
| 320 | 409600 | 204800 | 200785 |
| 640 | 1638400 | 819200 | 803138 |
| 1280 | 6553600 | 3276800 | 3212550 |

*Tabela 1 - Valores do parâmetro M*

## 4.1. CASO 1

O primeiro caso abordado no item “a” do enunciado é o caso em que a equação da fonte é igual a $f(t, x) = 10x^2 (x - 1) - 60xt + 20t$ , com T = 1, a partir de $u_0(x) = 0$ e condições de fronteiras nulas.

As integrações com N = 10, 20, 40, 80, 160 e 320 para λ = 0,5 e λ = 0,25 apresentam gráficos bastante similares entre si e próximos da solução exata, por esse motivo, optou-se por trazer alguns exemplos a fim de ilustrar a solução, os demais casos podem ser facilmente obtidos com a execução do programa.

Sendo assim, obtém-se os seguintes gráficos de evolução temporal, levando em conta os instantes de 0 a T = 1 a cada 0,1:

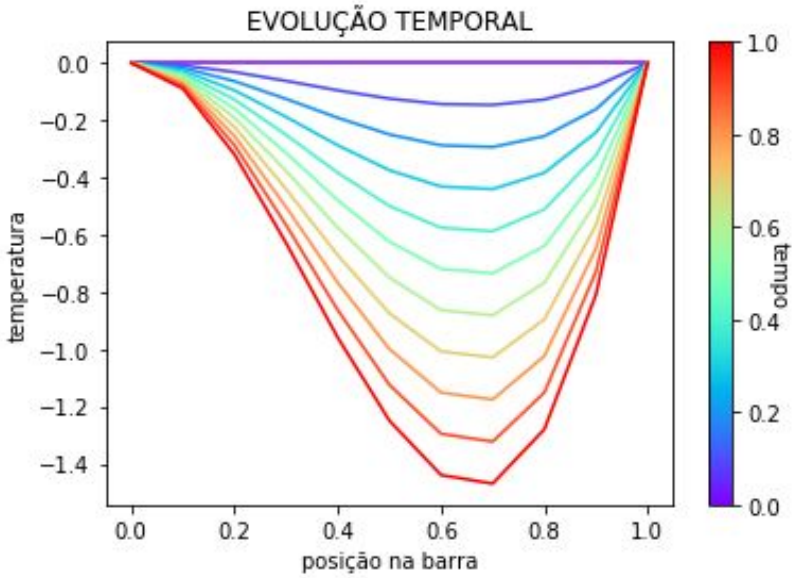


*Fig. 1 - Evolução temporal com N = 10 e M = 200 (λ = 0,50)*

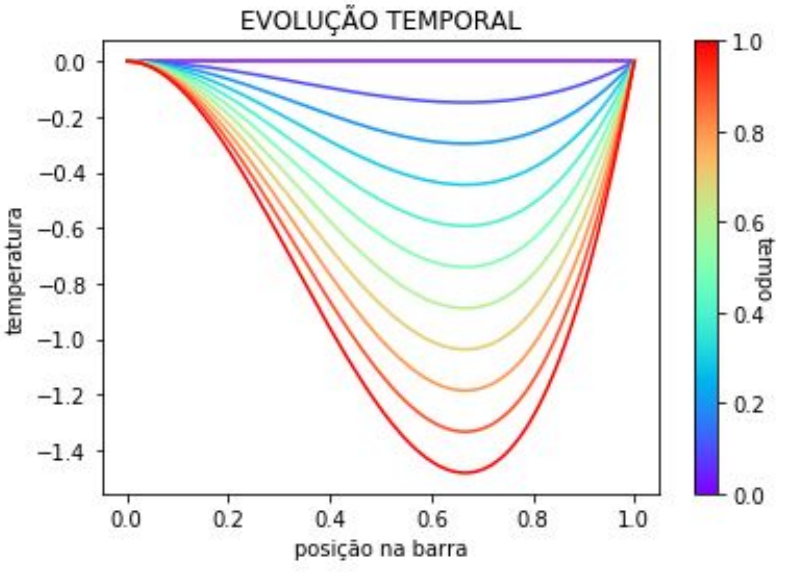


*Fig. 2 - Evolução temporal com N = 320 e M = 204800 (λ = 0,50)*

Verifica-se que a solução exata neste caso é $u(t,\ x)\ =\ 10tx^2\ (x-1)$, conforme indicado abaixo:

$$u_t(t,x)\ =\ u_{xx}(t,\ x)\ +\ f(t,\ x)\ em\ [0,\ T\ ]\times[0,\ 1]$$
$$u_t(t,x)\ =\ 10x^2\ (x-1)$$
$$u_{xx}(t,\ x)\ =10t(6x-2)$$
$$u_t(t,\ x)\ =\ 10x^2\ (x-1)\ =10t(6x-2)\ +\ 10x^2\ (x-1)-60xt\ +\ 20t$$

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

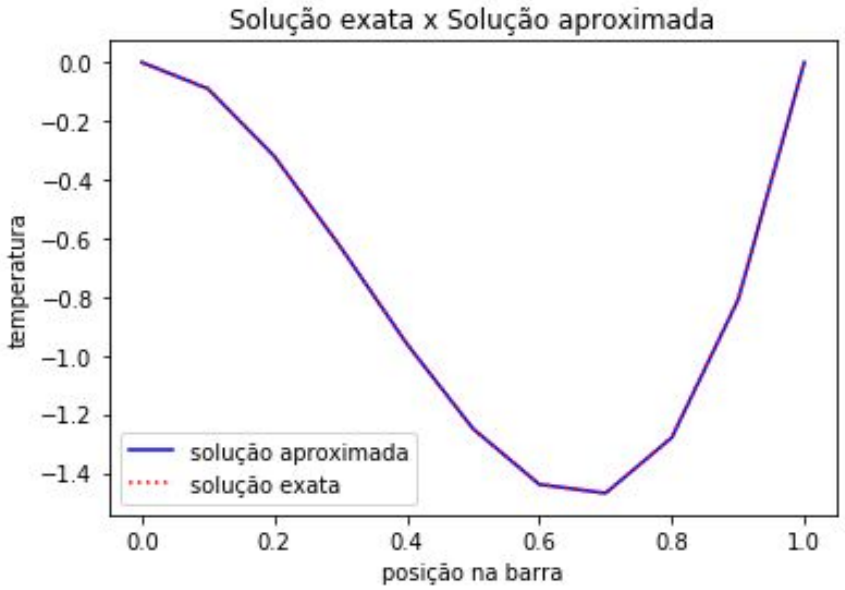


*Fig. 3 - Comparando com solução exata para N = 10 e M = 200 (λ = 0,50)*

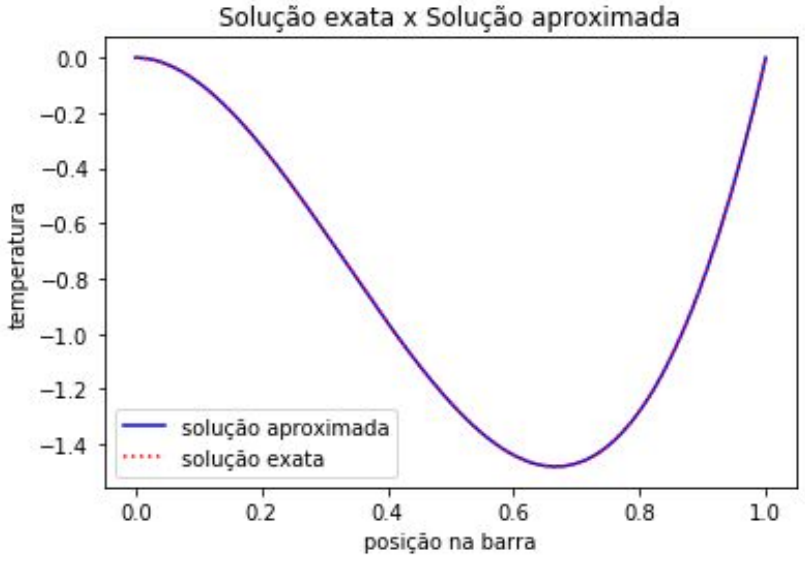


*Fig. 4 - Comparando com solução exata para N = 320 e M = 204800 (λ = 0,50)*

Porém, quando λ ultrapassa 0,5, o gráfico apresenta instabilidade, pois, como explicado, a convergência da solução no método explícito é condicional para $\lambda \leq 0{,}5$. Sendo assim, resulta nos seguintes gráficos:

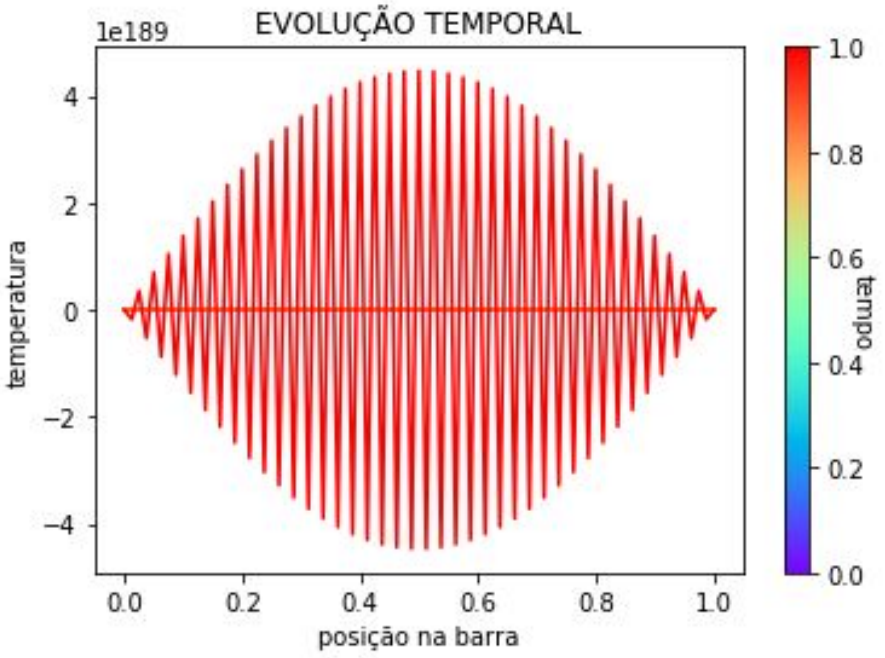


*Fig. 5 - Evolução temporal com N = 80 e M = 12550(λ = 0,51)*

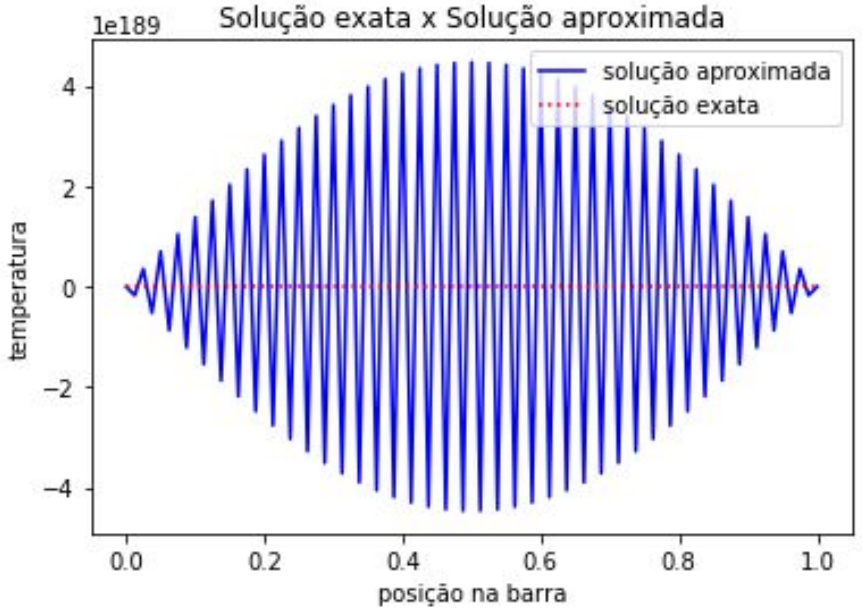


*Fig. 6 - Comparando com solução exata para N = 80 e M = 12550(λ = 0,51)*

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| | **Módulo do erro na aproximação** | | | **Módulo do erro de truncamento** | | |
|---|---|---|---|---|---|---|
| **N** | **λ = 0,25** | **λ = 0,50** | **λ = 0,51** | **λ = 0,25** | **λ = 0,50** | **λ = 0,51** |
| 10 | 8,8818E-16 | 4,4409E-16 | 6,6613E-16 | 4,6185E-14 | 4,6185E-14 | 6,3949E-14 |
| 20 | 1,5543E-15 | 8,8818E-16 | 4,8015E-10 | 4,7606E-13 | 4,1922E-13 | 4,7606E-13 |
| 40 | 1,7764E-15 | 1,3323E-15 | 1,3927E+30 | 1,6041E-12 | 1,6129E-12 | 1,7124E-12 |
| 80 | 1,1768E-14 | 1,7764E-15 | 4,4674E+189 | 6,4020E-12 | 7,3754E-12 | 7,5389E-12 |
| 160 | 2,4203E-14 | 2,6645E-15 | NaN | 3,0877E-11 | 2,6709E-11 | 2,7597E-11 |
| 320 | 2,3759E-14 | 1,3323E-14 | NaN | 1,7867E-10 | 1,2050E-10 | 1,1780E-10 |

*Tabela 2 - Valores máximos dos erros em T = 1 no caso 1*

Analisando o comportamento do módulo do erro, inicialmente para os casos em que a solução converge, conclui-se que ele é nulo, a menos de erros de arredondamento, e por isso ele cresce à medida que a malha é refinada.

Verifica-se, também, nos casos em que a solução converge que, conforme o esperado, o módulo do erro de truncamento limita superiormente o erro de aproximação, sendo ambos muito próximos em ordem de grandeza.

Pode-se demonstrar que o erro de truncamento é nulo, o que implica em um erro de aproximação também nulo, a menos de erro de arredondamento, conforme já mencionado. Tem-se:

$$\tau\,(\Delta t,\ \Delta x)\ =\ max_{k,i}\,|\tau_i^k\,(\Delta t,\ \Delta x)| \le C1\Delta t\ +\ C2\Delta x^2$$

$$\text{com}\ \ C1\ =\ max_D|\ \tfrac{u_{tt}(t,\,x)}{2}\ |\ e\ C2\ =\ max_D|\ \tfrac{u_{xxxx}(t,\,\bar{x})}{4!}|$$

É fácil perceber que tanto a derivada de segunda ordem em t, quanto a de quarta ordem em x, resulta em zero. Sendo assim, tem-se C1 = C2 = 0, portanto o erro de truncamento deve ser sempre nulo.

Nos casos em que λ ultrapassa 0,5, o erro na aproximação não é limitado pelo erro de truncamento e a solução não converge, com os valores do erro da aproximação indo para infinito, conforme verificamos na tabela.

Como o erro nesse caso é notadamente nulo, não há fator de redução para ele, já que deve ser sempre zero, mas na prática aumenta devido aos erros de arredondamento do método ao ser refinado.

Considerando que número de passos refere-se a discretização do tempo, podemos assumir que é igual ao parâmetro M, que tem seus valores exibidos na tabela 1. Sendo assim, para um dado λ, se N dobra, M quadruplica, já que ele varia com o quadrado de N.

## 4.2. CASO 2

O segundo caso abordado no item “a” é o caso em que a equação da fonte é igual a $f(t, x) = 10cos(10t)x^2 (1-x)^2 - (1 + sin(10t))(12x^2 - 12x + 2)$, que corresponde à solução exata $u(t, x) = (1 + sin(10t))x^2 (1-x)^2$, com valor inicial $u_0(x) = x^2(1-x)^2$ e condições de fronteiras nulas.

Assim como no caso anterior, os gráficos obtidos para todos os parâmetros solicitados (N = 10, 20, 40, 80, 160 e 320) se aproximam da solução exata e são semelhantes entre si, sendo que à medida em que a malha é refinada o resultado converge para a solução exata, se λ for menor ou igual a 0,5.

Para ilustrar a evolução temporal de 0 a T = 1, com t variando a cada 0,1, seguem dois gráficos com baixo e alto refinamento:

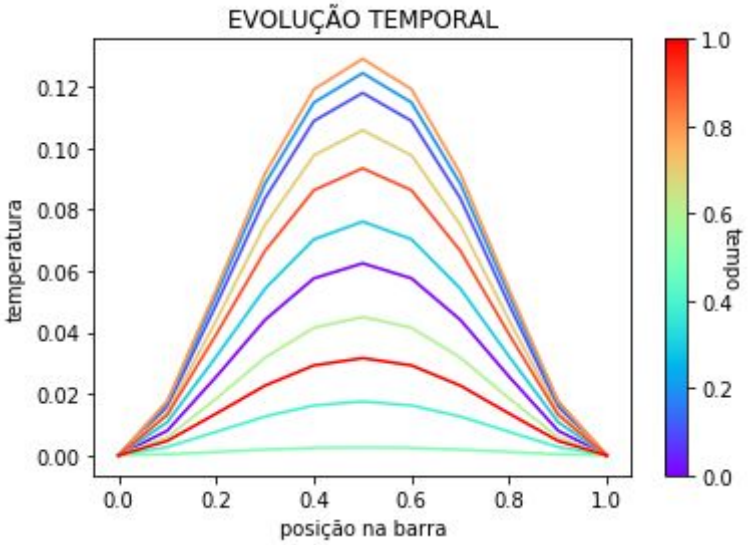


*Fig. 7 - Evolução temporal com N = 10 e M = 200 (λ = 0,50)*

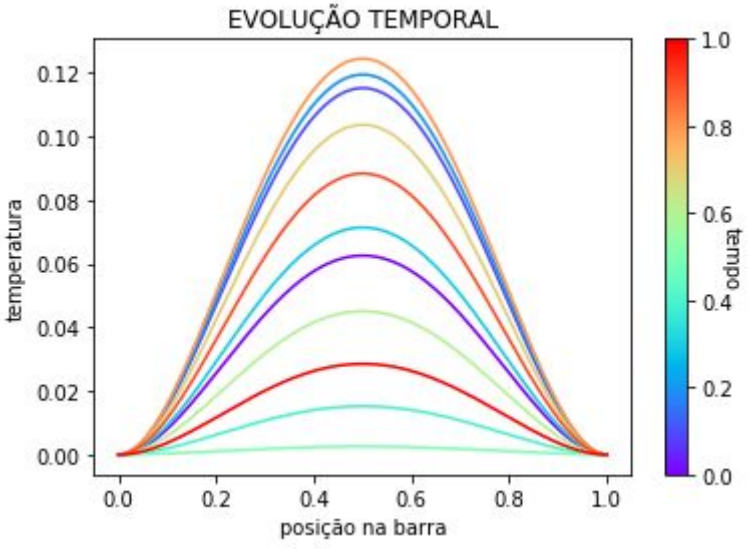


*Fig. 8 - Evolução temporal com N = 320 e M = 204800 (λ = 0,50)*

Verifica-se que a solução exata neste caso é $u(t,\ x)\ =\ (1\ +\ sin(10t))x^2\ (1-x)^2$ , conforme indicado abaixo:

$$u_t(t,x)\ =\ u_{xx}(t,\ x)\ +\ f(t,\ x)\ em\ [0,\ T\ ]\times[0,\ 1]$$
$$u_t(t,\ x)\ =10x^2cos(10t)(1-x)^2$$
$$u_{xx}(t,\ x)=(1+sin(10t)(12x^2-12x+2)$$
$$10x^2cos(10t)(1-x)^2=(1+sin(10t)(12x^2-12x+2)+10cos(10t)x^2\ (1-x)^2-(1\ +\ sin(10t))(12x^2-12x\ +\ 2)$$

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

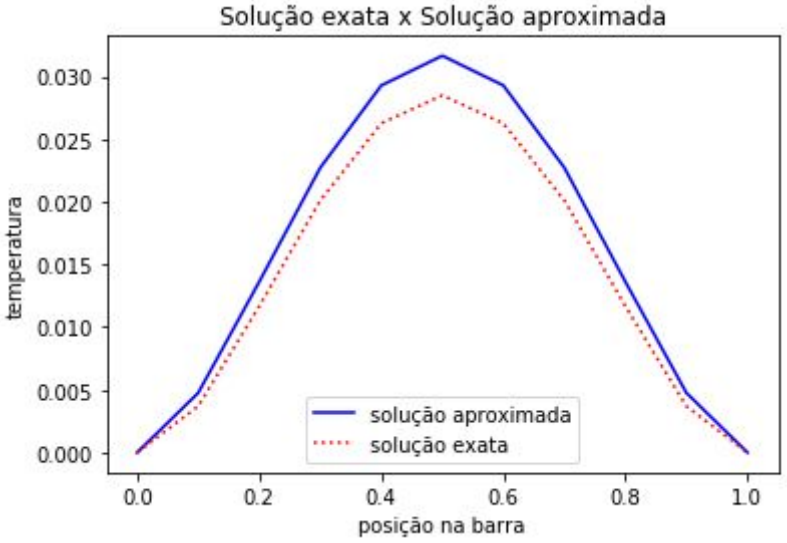


*Fig. 9 - Comparando com solução exata para N = 10 e M = 200 (λ = 0,50)*

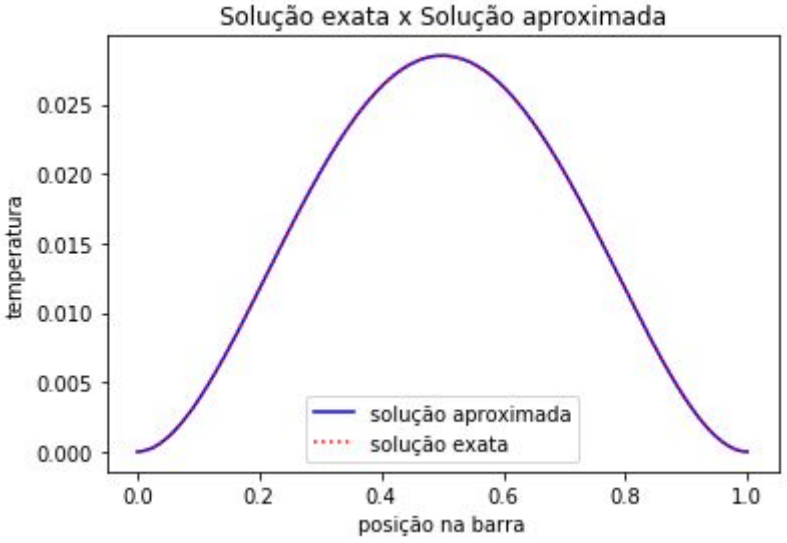


*Fig. 10 - Comparando com solução exata para N = 320 e M = 204800 (λ = 0,50)*

Como esperado para esse método, quando λ ultrapassa 0,5, o gráfico apresenta instabilidade, já que a convergência da solução no método explícito é condicional para λ $\leq$ 0,5. Sendo assim, resulta nos seguintes gráficos:

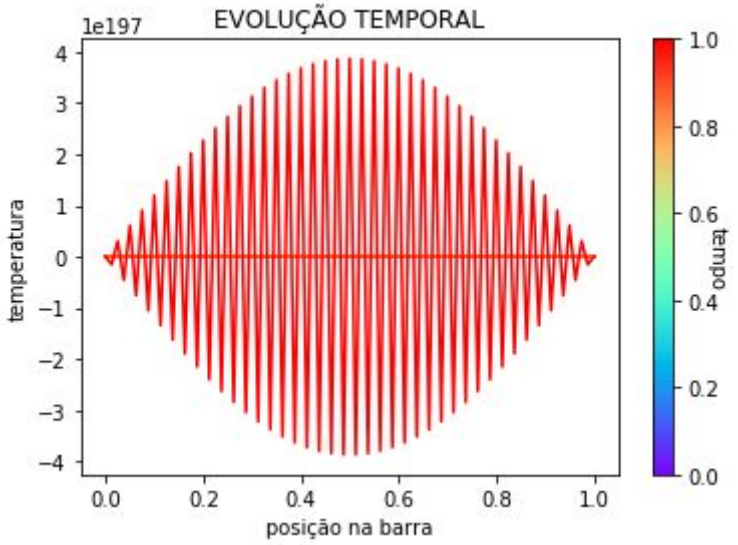


*Fig. 11 - Evolução temporal com N = 80 e M = 12550(λ = 0,51)*

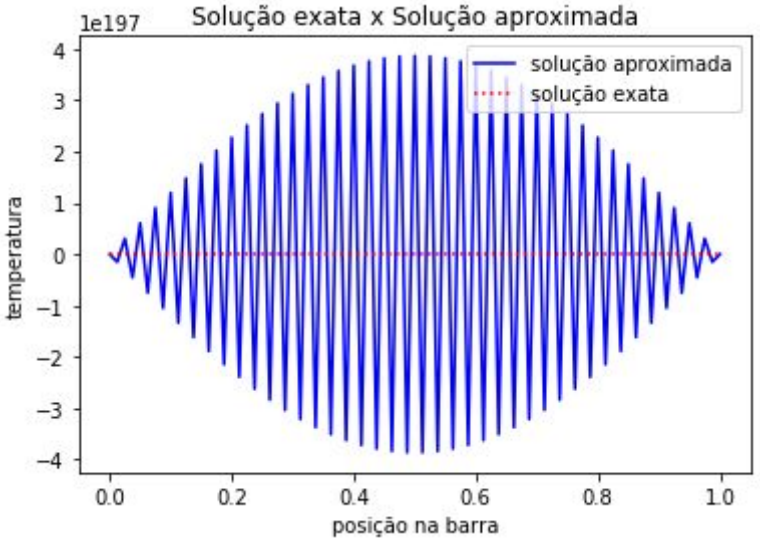


*Fig. 12 - Comparando com solução exata para N = 80 e M = 12550(λ = 0,51)*

Foram exibidas figuras que mostram os casos mais significativos, já que, no geral os resultados são bastante semelhantes entre si. No entanto, caso seja necessário, todos os demais parâmetros podem ser testados no código e os gráficos gerados por ele.

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| | Erro na aproximação | | | Erro de truncamento | | |
|---|---|---|---|---|---|---|
| **N** | **λ = 0,25** | **λ = 0,50** | **λ = 0,51** | **λ = 0,25** | **λ = 0,50** | **λ = 0,51** |
| 10 | 3,0468E-03 | 3,1637E-03 | 3,1673E-03 | 8,5617E-03 | 7,9898E-03 | 7,9722E-03 |
| 20 | 7,5782E-04 | 7,8421E-04 | 1,0756E+01 | 2,2414E-03 | 2,2027E-03 | 2,2012E-03 |
| 40 | 1,8922E-04 | 1,9564E-04 | 1,8716E+39 | 5,6745E-04 | 5,6492E-04 | 5,6482E-04 |
| 80 | 4,7289E-05 | 4,8883E-05 | 3,8697E+197 | 1,4233E-04 | 1,4217E-04 | 1,4216E-04 |
| 160 | 1,1821E-05 | 1,2219E-05 | NaN | 3,5613E-05 | 3,5603E-05 | 3,5602E-05 |
| 320 | 2,9553E-06 | 3,0547E-06 | NaN | 8,9052E-06 | 8,9045E-06 | 8,9045E-06 |

*Tabela 3 - Valores máximos dos erros em T = 1 no caso 2*

Inicialmente, nota-se que o erro de aproximação vai a infinito a medida que a malha é refinada para λ = 0,51, já que o método não converge para esse caso.

Assim como explicado no caso anterior, para um dado λ, se N dobra, M quadruplica, já que ele varia com o quadrado de N. E dessa forma, $\Delta t$ pode ser escrito como $\Delta t = \lambda * \Delta x^2 = \lambda * (1/N)^2$, sendo λ uma constante.

Para realizar o cálculo do fator de redução para esse caso, levando em consideração que $\Delta t = \lambda * \Delta x^2 = \lambda * (1/N)^2$ e que a convergência é de ordem 2 em $\Delta x$ e de ordem 1 em $\Delta t$:

$$max_{k,i} |\tau_i^k (\Delta t, \Delta x)| \leq C1\Delta t + C2\Delta x^2, \; com \; C1 \; e \; C2 \; constantes$$

$$max_{k,i} |\tau_i^k (\Delta t, \Delta x)| \leq C'(\frac{1}{N^2}) + C''(\frac{1}{N^2})$$

Ao avaliar o erro em um caso em que N muda, temos que a variação de um caso para outro vai estar associada a um fator $1/N^2$.

Logo, quando o N dobra, é esperado que o erro se reduza para um quarto. Sendo assim, o fato de redução é $\frac{1}{4}$. Essa redução é comprovada através tabela de erros acima. Cada vez que N dobra, para um dado valor de λ, percebe-se que o erro é dividido por 4, nos casos em que o método converge.

### 4.3. CASO 3

No item “b”, a determinação de $u_0(x),\ g_1(t),\ g_2(t)\ e\ f(t,\ x)$, sabendo que a solução exata é dada por $u(t,\ x)\ =\ e^{t-x}cos(5tx)$, é feita a partir das equações (2), (3), (4) e (1), respectivamente. Logo:

$$u_0(x) = u(0,x) = \ e^{0-x}cos(5 * 0 * x)\ = e^{-x}$$

$$g_1(t) = u(t,0) = e^{t-0}cos(5 * t * 0) = e^{t}$$

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

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

$$u_{xx}\ (t,\ x) = \ e^{t-x}cos(5tx) + 10e^{t-x}tsin(5yx)\ - 25e^{t-x}t^2cos(5tx)$$

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

Após terem sido feitas as mesmas análises dos casos anteriores, considerando os parâmetros solicitados (N = 10, 20, 40, 80, 160 e 320), foi possível constatar que os gráficos também aproximam da solução exata e são semelhantes entre si. Conforme a malha é refinada, o resultado converge ainda mais para a solução exata, se λ for menor ou igual a 0,5.

Da mesma forma, optou-se por trazer alguns exemplos a fim de ilustrar a solução obtida. Para ilustrar a evolução temporal de 0 a T = 1, com t variando a cada 0,1, seguem dois gráficos com baixo e alto refinamento:

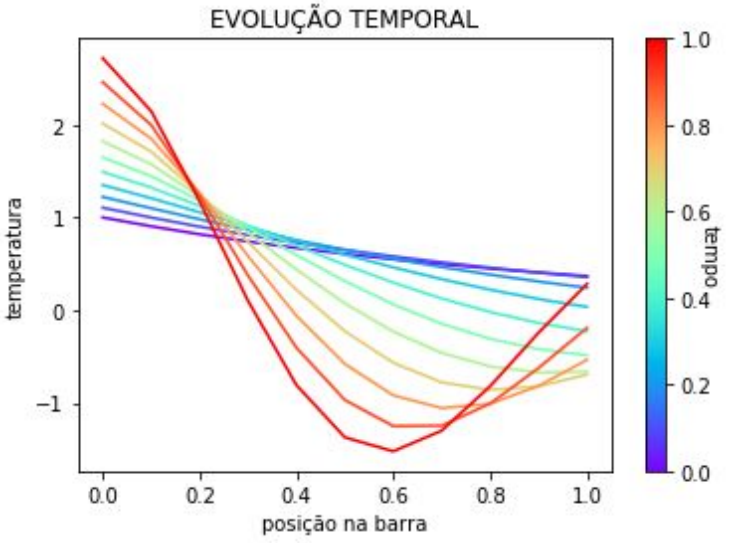


*Fig. 13 - Evolução temporal com N = 10 e M = 200 (λ = 0,50)*

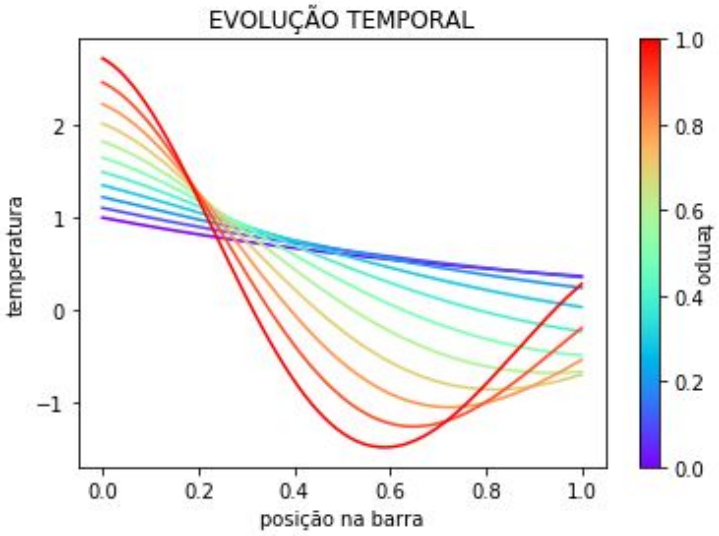


*Fig. 14 - Evolução temporal com N = 320 e M = 204800 (λ = 0,50)*

A comparação da solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, pode ser representada pelos seguintes gráficos:

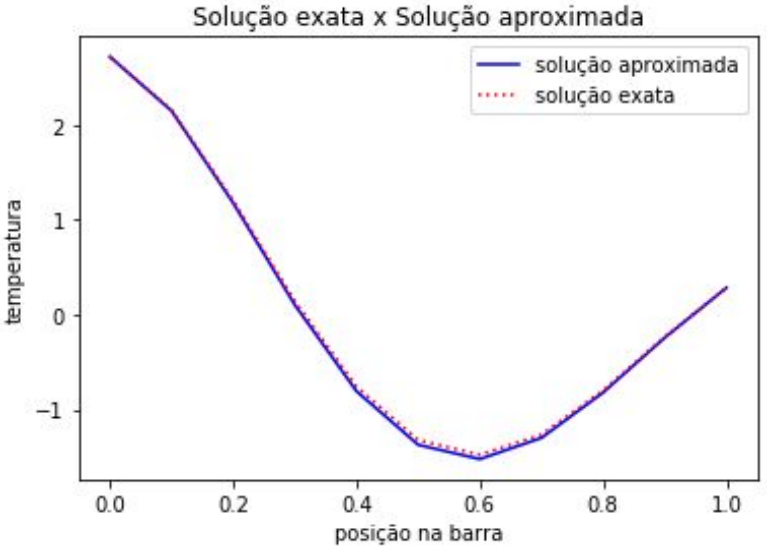


*Fig. 15 - Comparando com solução exata para N = 10 e M = 200 (λ = 0,50)*

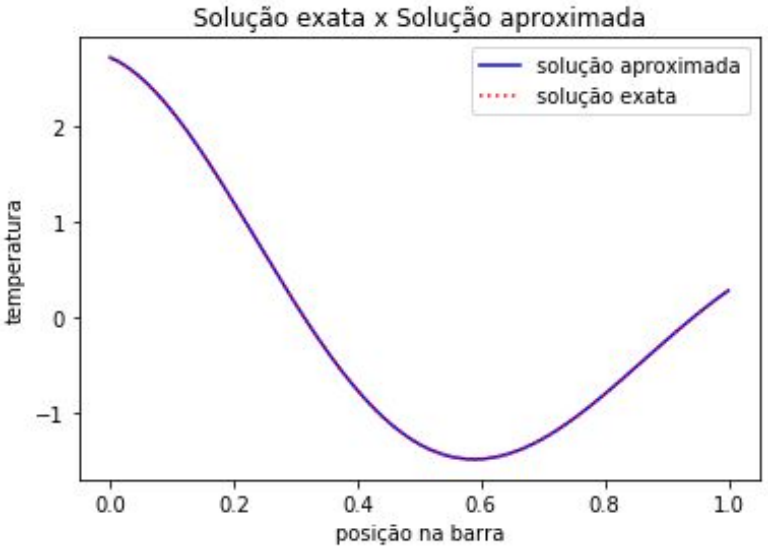


*Fig. 16 - Comparando com solução exata para N = 320 e M = 204800 (λ = 0,50)*

O resultado dos gráficos, quando λ ultrapassa 0,5, também apresenta instabilidade nesse caso, o que comprova novamente a condicionalidade do método:

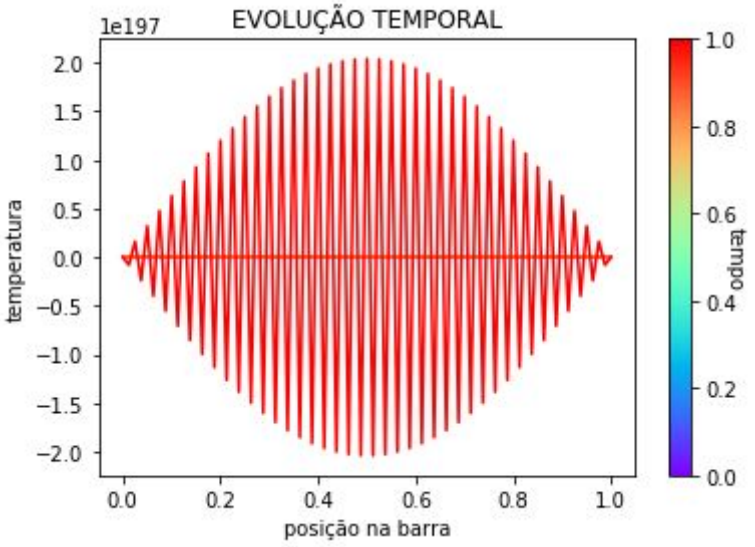


*Fig. 17 - Evolução temporal com N = 80 e M = 12550(λ = 0,51)*

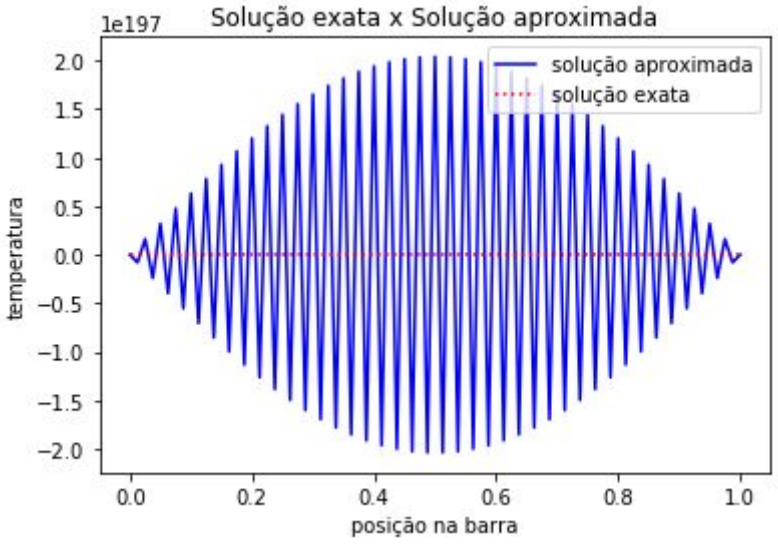


*Fig. 18 - Comparando com solução exata para N = 80 e M = 12550(λ = 0,51)*

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| | **Erro na aproximação** | | | **Erro de truncamento** | | |
|---|---|---|---|---|---|---|
| **N** | **0,25** | **0,50** | **0,51** | **0,25** | **0,50** | **0,51** |
| 10 | 5,0048E-02 | 5,0452E-02 | 5,0465E-02 | 9,4924E-01 | 9,4384E-01 | 9,4367E-01 |
| 20 | 1,2499E-02 | 1,2582E-02 | 5,6978E+00 | 2,4181E-01 | 2,4130E-01 | 2,4128E-01 |
| 40 | 3,1285E-03 | 3,1537E-03 | 9,8786E+38 | 6,0686E-02 | 6,0442E-02 | 6,0432E-02 |
| 80 | 7,8187E-04 | 7,8822E-04 | 2,0418E+197 | 1,5505E-02 | 1,5454E-02 | 1,5451E-02 |
| 160 | 1,9550E-04 | 1,9704E-04 | NaN | 4,0385E-03 | 4,0254E-03 | 4,0249E-03 |
| 320 | 4,8873E-05 | 4,9261E-05 | NaN | 1,0297E-03 | 1,0264E-03 | 1,0263E-03 |

*Tabela 4 - Valores máximos dos erros em T = 1 no caso 3*

Inicialmente, nota-se que o erro de aproximação vai a infinito a medida que a malha é refinada para λ = 0,51, já que o método não converge para esse caso.

Assim como explicado nos casos anteriores, o número de passos M quadruplica a medida em que N dobra. Já que estamos empregando o mesmo método de solução, teremos também um fator de redução igual a $\frac{1}{4}$, conforme já foi demonstrado.

Essa redução é comprovada através tabela de erros acima. Cada vez que N dobra, para um dado valor de λ, percebe-se que o erro é dividido por 4, nos casos em que o método converge.

## 4.4. CASO 4

O último caso abordado no item “c” do enunciado é o caso em que existe uma fonte pontual localizada em determinado ponto p do domínio e essa fonte pontual varia sua intensidade ao longo do tempo.

A equação que determina essa variação de intensidade ao longo do tempo é dada como:

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

Além disso, foi explicado que para implementar uma fonte pontual, dita de intensidade 1, a fonte f localizada pode ser vista como limite de fontes $g_h$ que atuam em uma pequena região (cujo tamanho vai a zero com h) em torno do ponto em questão, sendo não nulas apenas nesta pequena região, de forma que a integral de cada $g_h$ seja constante igual a 1. Para obter tal efeito, foi adotado então:

$$g_h(x) = \frac{1}{h}\ , se\ p - h/2 \leq x \leq p + h/2,\ e\ gh(x) = 0\ caso\ contrário.$$

Sendo assim, a função f que descreve a fonte do caso 4 depende tanto de t quanto de x e é produto das duas funções, $r(t)$ e $g_h(x)$.

$$f(x, t) = r(t) * g_h(x)$$

Como solicitado, o parâmetro p foi definido como 0,25, T = 1, as condições $g_1(t)\ e\ g_2(t) = 0$ e a condição inicial também nula. Neste caso não é conhecida a solução exata, e ela foi determinada numericamente.

Analogamente aos casos anteriores, foram considerados os parâmetros solicitados, e os gráficos gerados também ficaram próximos da solução exata e são semelhantes entre si, convergindo mais para a solução exata com aumento do refinamento da malha, levando em consideração a condição de convergência para o método, $\lambda \leq 0{,}5$.

Da mesma forma, optou-se por trazer dois exemplos com baixo e alto refinamento para ilustrar a evolução temporal de 0 a T = 1, com t variando a cada 0,1, mas os demais casos podem ser são obtidos com a execução do programa.

Como a neste caso não não se sabe a solução exata, o erro não foi possível de ser calculado.

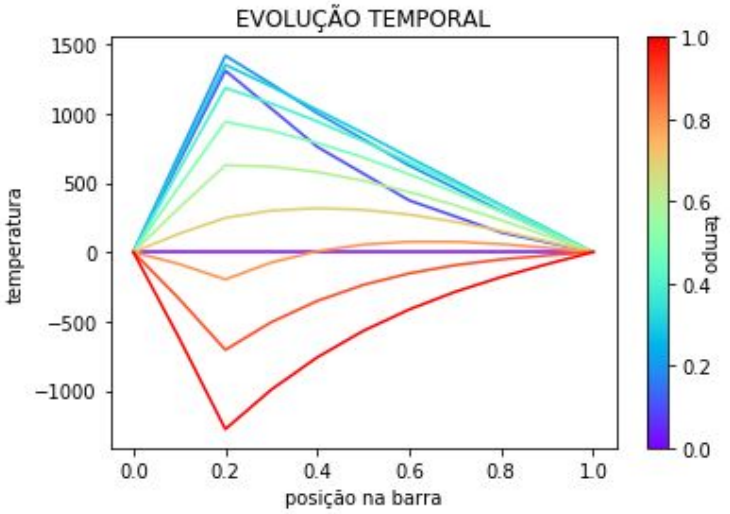


*Fig. 19 - Evolução temporal com N = 10 e M = 200 (λ = 0,50)*

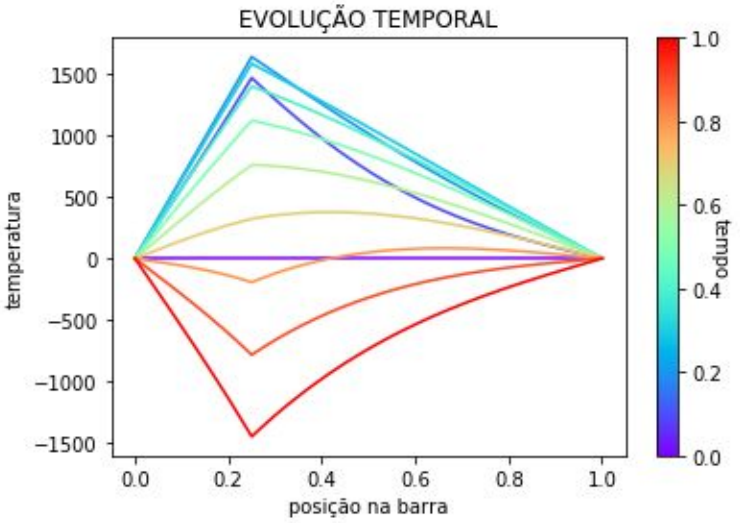


*Fig. 20 - Evolução temporal com N = 320 e M = 204800 (λ = 0,50)*

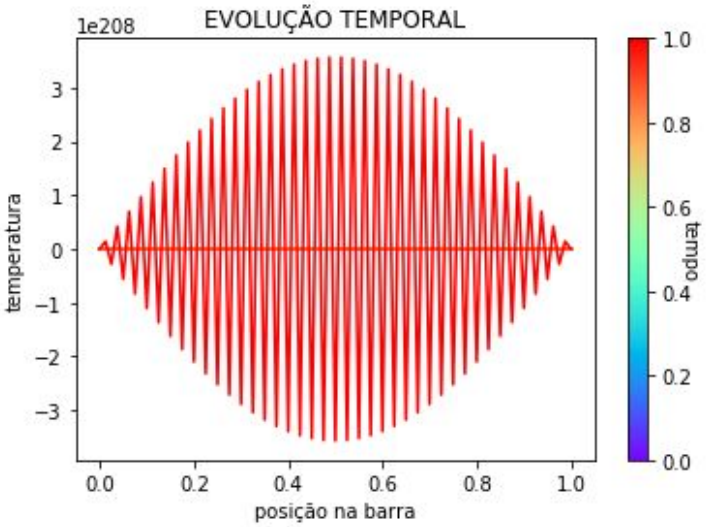


*Fig. 21 - Evolução temporal com N = 80 e M = 12550 (λ = 0,51)*

# 5. SEGUNDA TAREFA

A segunda tarefa pede a implementação dos dois método implícitos, Euler e Crank-Nicolson, para os mesmos casos citados na primeira tarefa. Nesses métodos, é necessário resolver o sistema linear com uma matriz A tridiagonal simétrica. Essa resolução, por sua vez, se dá pela decomposição $A = LDL^t$ de cada matriz.

No código, foi implementada uma rotina que calcula essa decomposição $A = LDL^t$ da matriz do sistema tridiagonal em questão. Essa rotina implementada na função *decomposicao(diagonal, subdiagonal)* recebe como entradas os vetores *diagonal* e *subdiagonal*, representando a matriz A, e fornece como saídas os vetores *vl* e *vd*, representando as matrizes L e D, respectivamente.

Depois de realizar uma série de multiplicações de matrizes menores (2x2, 3x3, 4x4…), verificamos um padrão para a decomposição. A fórmula encontrada para o cálculo de cada um desses vetores encontra-se abaixo:

$$d_i = \begin{cases} A_{ii}; & i = 1 \\ A_{ii} - d_{i-1} l_{i-1}^2; & i = 2, 3, ..., N-1 \end{cases}$$

$$l_i = \frac{A_{ij}}{d_j};\ i = 2, 3, ..., N-1 \ e \ j = i-1$$

Os valores $A_{ii}$ correspondem aos valores da matriz A armazenados no vetor *diagonal,* já os valores $A_{ij}$ correspondem aos armazenados no vetor *subdiagonal.* Além disso, por estarmos lidando com vetores, os primeiros valores sempre são armazenados começando no índice 0 do vetor e não no valor de i indicado acima.

Depois dessa rotina, é chamada outra função que resolve o sistema $LDL^t$, *solucao_LDLt(vl, vd, b).* Para tal, é necessário resolver inicialmente um sistema $L z = b.$ Em seguida, com o valor de z calculado, resolve-se outro sistema $D y = z$ . Por último, encontra-se a solução ao resolver o sistema $L^t x = y$ . Sendo x o vetor da solução, z e y vetores com soluções intermediárias e b o vetor do sistema original A x = b.

A resolução de cada um desses sistemas é bastante simples e é fácil verificar que as soluções intermediárias e finais podem ser escritas da seguinte forma:

$$z_i = \begin{cases} b_1; & i = 1 \\ b_i - l_{i-1} * z_{i-1}; & i = 2, 3, ..., N-1 \end{cases}$$

$$y_i = z_i / d_i$$

$$x_i = \begin{cases} y_{N-1}; & i = N-1 \\ y_i - l_i * x_{i+1}; & i = N-2, N-3, ..., 0 \end{cases}$$

Sendo N - 1 a última posição do vetor de tamanho N - 1 que começa em 1. No entanto, no código, os vetores iniciam com zero e não com o valor de i indicado acima, como já foi dito.

Levando em conta que o enunciado solicita que os valores de M e N sejam escolhidos em tempo de execução e fornece valores N para os testes, pede-se nesta segunda tarefa que seja considerado $\Delta t = \Delta x$, devemos efetuar o cálculo do valor de M. Sendo assim:

$$\Delta x = 1/N$$

$$\Delta t = T/M = 1/M = 1/N$$

$$\lambda = \frac{\Delta t}{\Delta x^2} = \frac{1/N}{1/N^2} = N$$

$$M = N = \lambda$$

Considerando que o número de passos da solução é o valor de M, temos abaixo como ele varia de acordo com cada valor de N e λ.

| ***N*** | ***M*** | ***λ*** |
|---|---|---|
| *10* | *10* | *10* |
| *20* | *20* | *20* |
| *40* | *40* | *40* |
| *80* | *80* | *80* |
| *160* | *160* | *160* |
| *320* | *320* | *320* |
| *640* | *640* | *640* |
| *1280* | *1280* | *1280* |

*Tabela 5 - Valores dos parâmetros*

## 5.1. Método de Euler implícito

### 5.1.1. CASO 1

Resolvendo novamente o caso 1, dessa vez pelo método de Euler implícito, para os mesmos valores de N testados, mas considerando $\Delta t = \Delta x$, os gráficos gerados também ficaram próximos da solução exata e são semelhantes entre si, convergindo mais para a solução exata com aumento do refinamento da malha. É válido lembrar que esse método é incondicionalmente convergente, e por isso não precisou ser atribuída nenhuma limitação ao $\lambda$.

Da mesma forma, optou-se por trazer dois exemplos com baixo (N=M= $\lambda$ =10) e alto refinamento (N=M= $\lambda$ =320) para ilustrar a evolução temporal de 0 a T = 1, com t variando a cada 0,1, mas os demais casos, de forma análoga, podem ser são obtidos com a execução do programa.

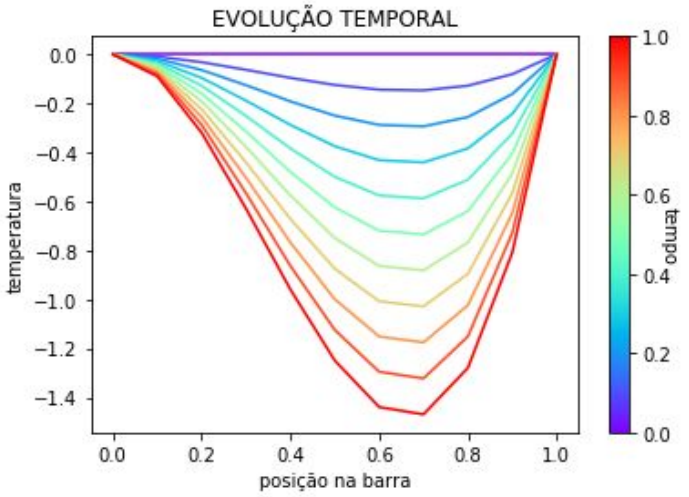


*Fig. 22- Evolução temporal com N = 10 e M = 10 (λ = 10)*

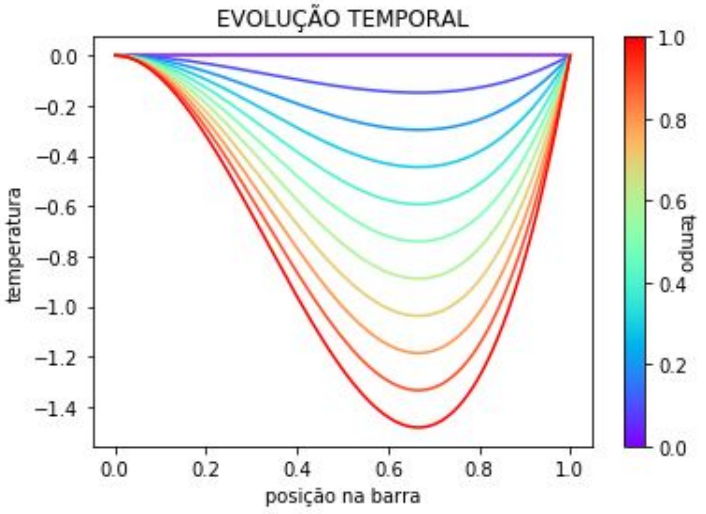


*Fig. 23 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

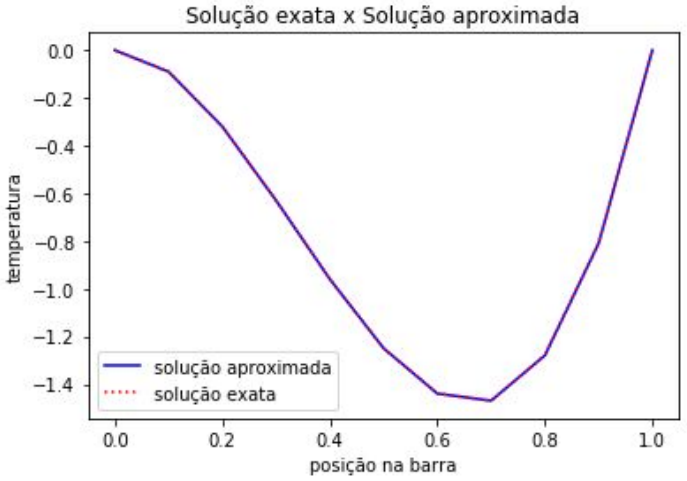


*Fig. 24 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

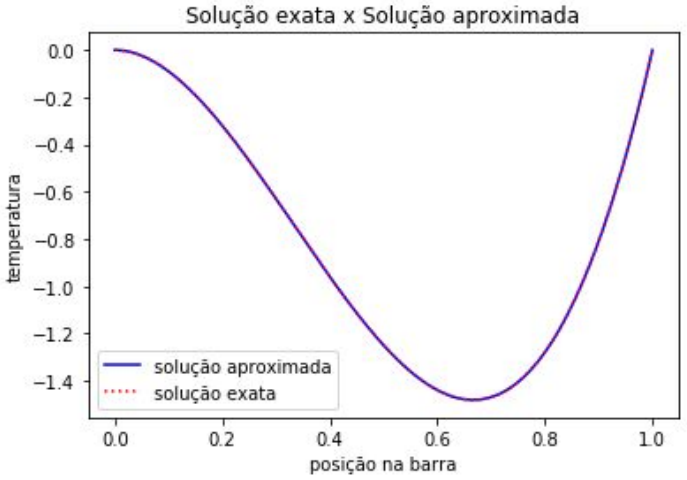


*Fig. 25 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

A tabela abaixo apresenta os valores máximos dos erros em T = 1.

| N | Erro na aproximação | Erro de truncamento |
|---|---|---|
| 10 | 6,661E-16 | 5,684E-14 |
| 20 | 2,442E-15 | 4,974E-13 |
| 40 | 1,776E-15 | 1,741E-12 |
| 80 | 5,329E-15 | 6,906E-12 |
| 160 | 8,993E-15 | 2,581E-11 |
| 320 | 2,887E-14 | 1,133E-10 |
| 640 | 5,311E-12 | 4,533E-10 |

*Tabela 6 - Valores máximos dos módulos dos erros em T = 1 no caso 1*

Como já foi demonstrado para este caso o erro é notadamente nulo, e portanto, não há fator de redução para ele, já que deve ser sempre zero, mas na prática aumenta devido aos erros de arredondamento do método ao ser refinado.

### 5.1.2. CASO 2

Para o caso 2, são feitas as mesmas considerações e análises, chegando nas mesmas conclusões de que os gráficos ficaram próximos da solução exata e são semelhantes entre si, convergindo mais para a solução exata com aumento do refinamento da malha.

Na sequência são apresentados apenas dois exemplos, um com baixo (N=M=$\lambda$=10) e outro com alto refinamento (N=M=$\lambda$=320) para ilustrar a evolução temporal de 0 a T = 1, com t variando a cada 0,1.

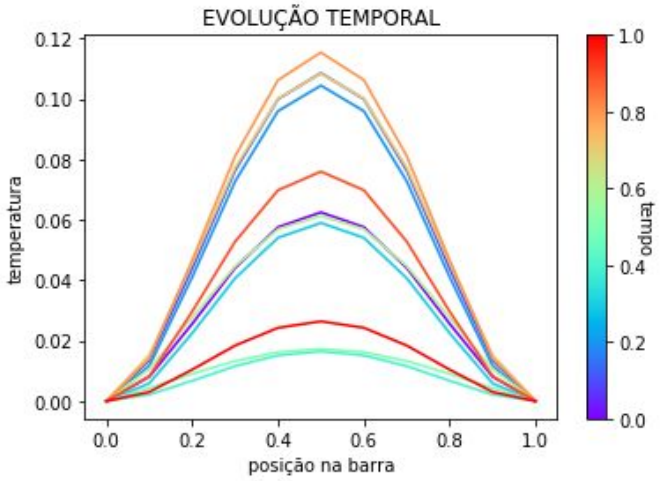


*Fig. 26- Evolução temporal com N = 10 e M = 10 (λ = 10)*

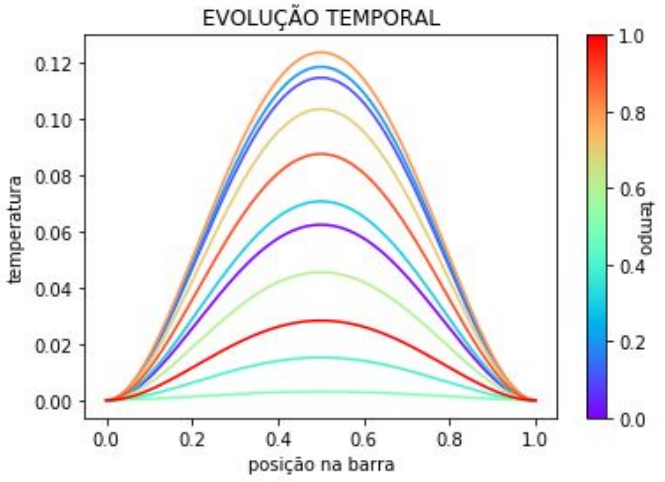


*Fig. 27- Evolução temporal com N = 320 e M = 320 (λ = 320)*

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

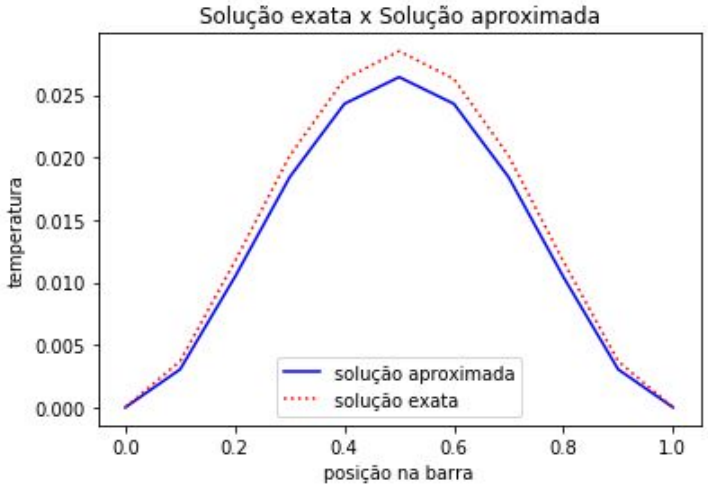


*Fig. 28- Evolução temporal com N = 10 e M = 10 (λ = 10)*

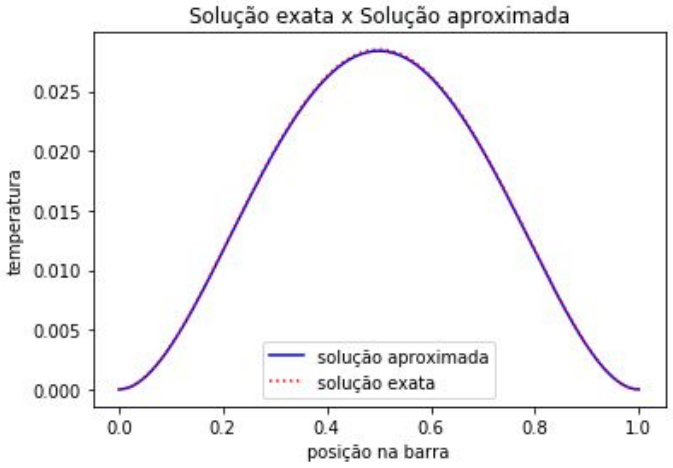


*Fig. 29 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| N | Erro na aproximação | Erro de truncamento |
|---|---|---|
| 10 | 2,079E-03 | 2,877E-01 |
| 20 | 1,496E-03 | 1,230E-01 |
| 40 | 8,785E-04 | 5,303E-02 |
| 80 | 4,737E-04 | 2,401E-02 |
| 160 | 2,457E-04 | 1,133E-02 |
| 320 | 1,251E-04 | 5,491E-03 |
| 640 | 6,309E-05 | 2,701E-03 |

*Tabela 7 - Valores máximos dos erros em T = 1 no caso 2*

Para realizar o cálculo de quanto é o fator de redução do erro no método de Euler implícito, levando em consideração que $\Delta t = \Delta x = 1/N$ e que a convergência é de ordem 2 em $\Delta x$ e de ordem 1 em $\Delta t$:

$$max_{k,i} |\tau_i^k (\Delta t,\ \Delta x)| \leq C1\Delta t + C2\Delta x^2,\ com\ C1\ e\ C2\ constantes$$

$$max_{k,i} |\tau_i^k (\Delta t,\ \Delta x)| \leq C1(\tfrac{1}{N}) + C2(\tfrac{1}{N^2})$$

Ao avaliar o erro em um caso em que N muda, temos que a variação de um caso para outro vai estar associada a um fator 1/N, já que este é maior que $1/N^2$.

Logo, quando o N dobra, é esperado que o erro se reduza pela metade. Sendo assim, o fato de redução é $\frac{1}{2}$. Essa redução é comprovada através tabela de erros acima.

### 5.1.3. CASO 3

Para o caso 3, são também feitas as mesmas considerações e análises, chegando nas mesmas conclusões.

Na sequência são apresentados apenas os dois exemplos com baixo (N=M=$\lambda$=10) e alto refinamento (N=M=$\lambda$=320) para ilustrar a evolução temporal.

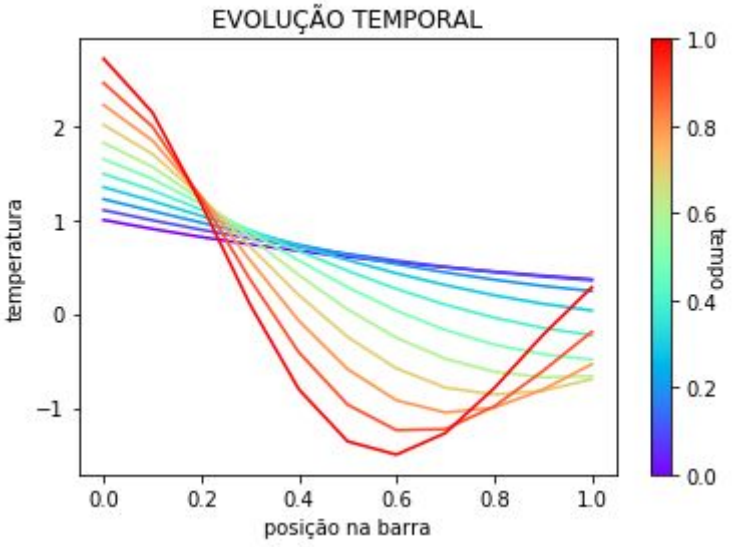


*Fig. 30 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

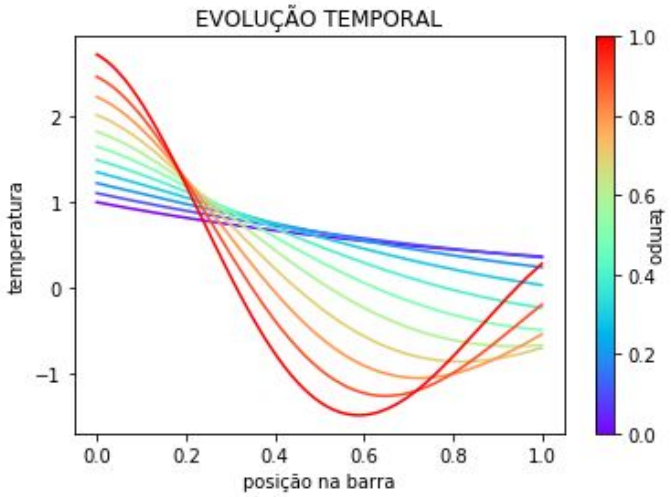


*Fig. 31- Evolução temporal com N = 320 e M = 320 (λ = 10)*

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

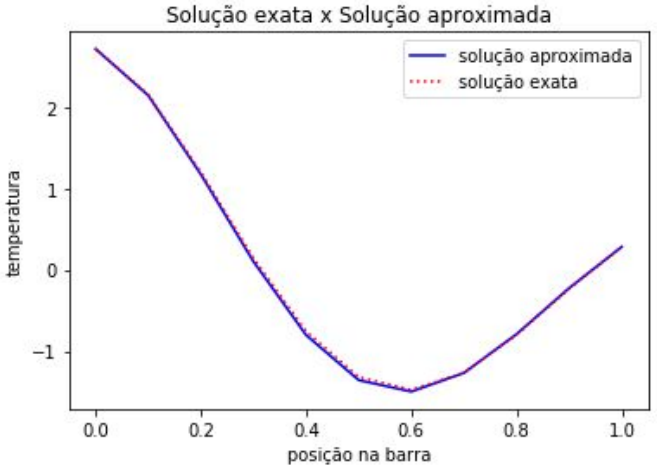


*Fig. 32 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

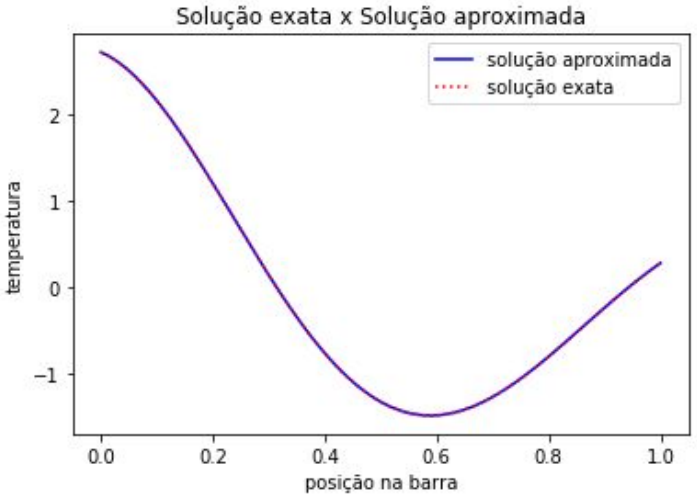


*Fig. 33 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| N | Erro na aproximação | Erro de truncamento |
|---|---|---|
| 10 | 4,497E-02 | 1,755E+00 |
| 20 | 9,371E-03 | 5,216E-01 |
| 40 | 6,276E-03 | 2,469E-01 |
| 80 | 3,609E-03 | 1,223E-01 |
| 160 | 1,929E-03 | 6,104E-02 |
| 320 | 9,968E-04 | 3,052E-02 |
| 640 | 5,065E-04 | 1,526E-02 |

*Tabela 8 - Valores máximos dos erros em T = 1 no caso 3*

Conforme descrito no caso 2, o fator de redução nesse caso também é $\frac{1}{2}$, ou seja, quando o N dobra, é esperado que o erro se reduza pela metade. Essa redução é comprovada através tabela de erros acima.

### 5.1.4. CASO 4

Para o caso 4, são também feitas as mesmas considerações e análises. Como para esse caso não se sabe a solução exata, não é possível plotar gráficos de comparação das soluções.

Na sequência são apresentados apenas os dois exemplos com baixo (N=M=$\lambda$=10) e alto refinamento (N=M=$\lambda$=320) para ilustrar a evolução temporal.

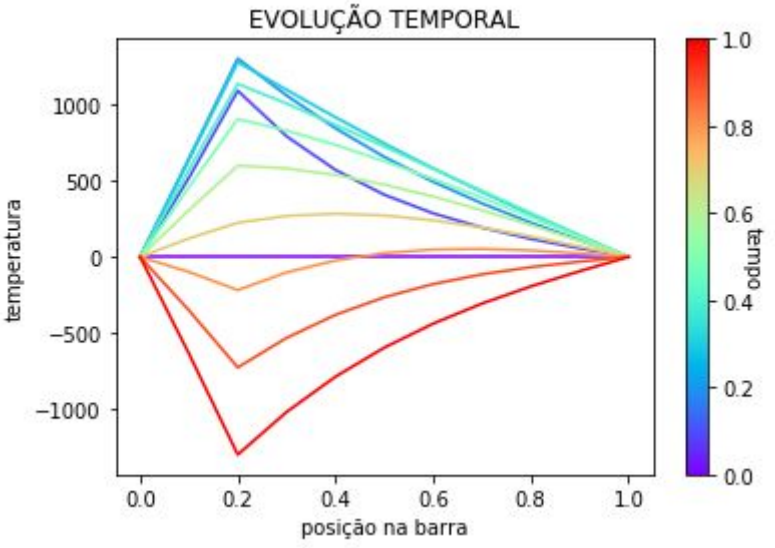


*Fig. 34 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

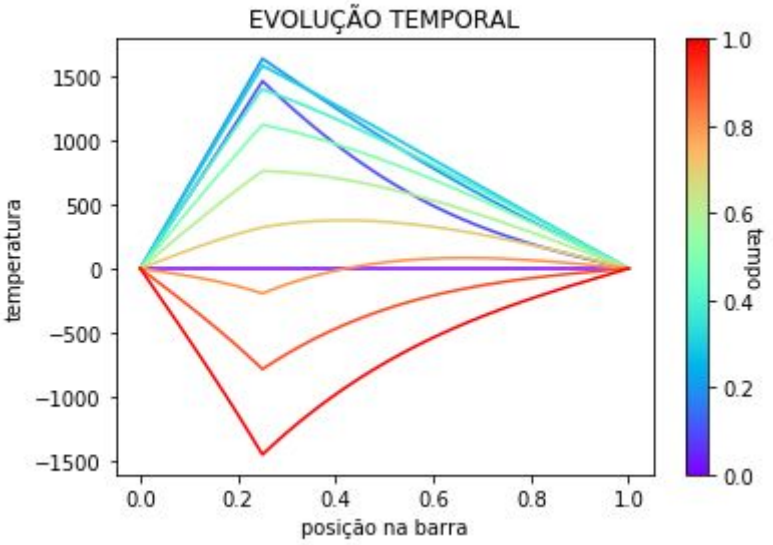


*Fig. 35 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

## 5.2. Método de Crank-Nicolson

### 5.2.1. CASO 1

Resolvendo pela última vez o caso 1, dessa vez pelo método de Crank-Nicolson, para os mesmos valores de N testados, e fazendo a mesma consideração $\Delta t = \Delta x$, os gráficos gerados também ficaram próximos da solução exata e são semelhantes entre si, convergindo mais para a solução exata com aumento do refinamento da malha. É válido lembrar que esse método também é incondicionalmente convergente, e por isso não precisou ser atribuída nenhuma limitação ao $\lambda$.

Da mesma forma, optou-se por trazer dois exemplos com baixo (N=M=$\lambda$ =10) e alto refinamento (N=M=$\lambda$=320) para ilustrar a evolução temporal de 0 a T = 1, com t variando a cada 0,1, mas os demais casos, de forma análoga, podem ser são obtidos com a execução do programa.

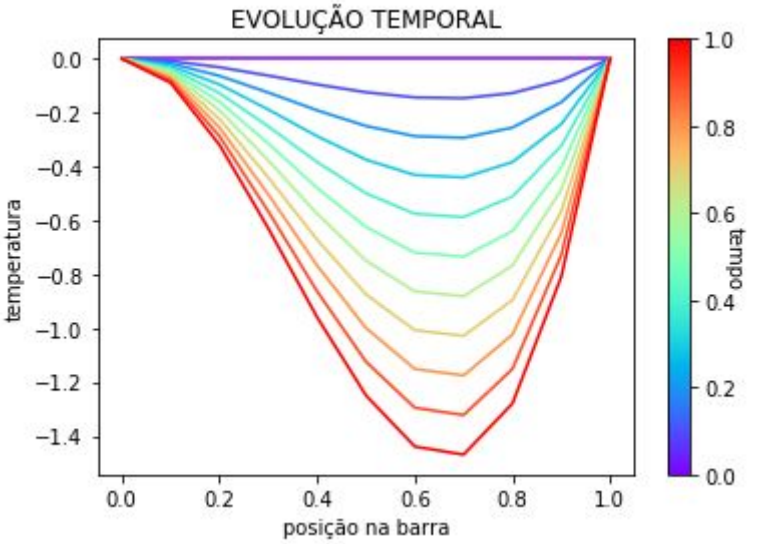


*Fig. 36 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

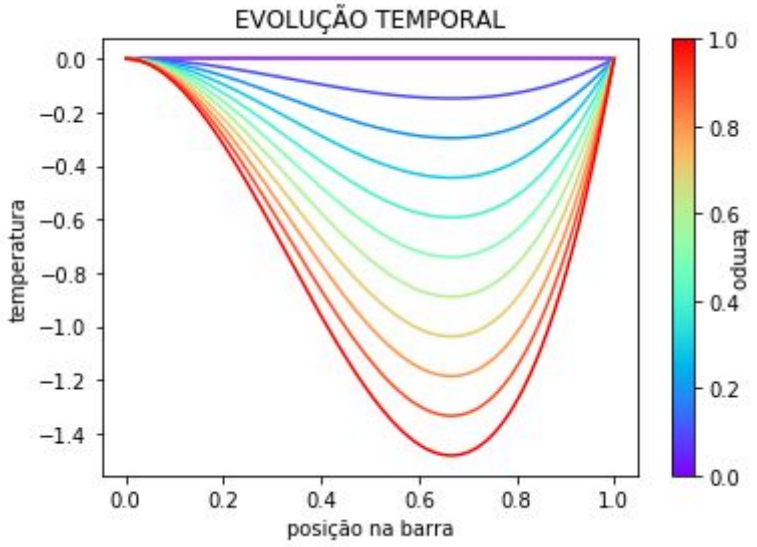


*Fig. 37 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

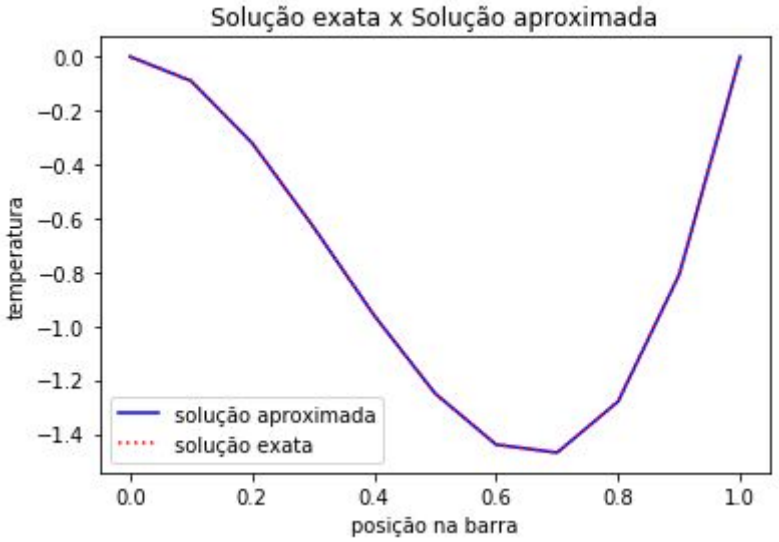


*Fig. 38 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

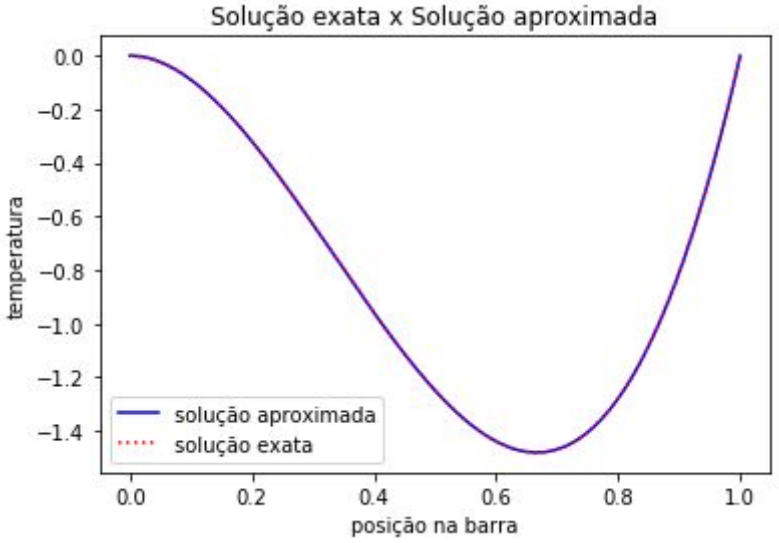


*Fig. 39 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

A tabela abaixo apresenta os valores máximos dos erros em T = 1.

| N | Erro na aproximação | Erro de truncamento |
|---|---|---|
| 10 | 6,661E-16 | 4,263E-14 |
| 20 | 8,882E-16 | 4,761E-13 |
| 40 | 5,329E-15 | 1,734E-12 |
| 80 | 1,998E-15 | 6,885E-12 |
| 160 | 1,421E-14 | 2,616E-11 |
| 320 | 3,058E-13 | 1,120E-10 |
| 640 | 2,853E-12 | 4,541E-10 |

*Tabela 9 - Valores máximos dos erros em T = 1 no caso 1*

Como já foi demonstrado para este caso o erro é notadamente nulo, e portanto, não há fator de redução para ele, já que deve ser sempre zero, mas na prática aumenta devido aos erros de arredondamento do método ao ser refinado.

### 5.2.2. CASO 2

Para o caso 2, são também feitas as mesmas considerações e análises, chegando nas mesmas conclusões.

Na sequência são apresentados apenas os dois exemplos com baixo (N=M=λ=10) e alto refinamento (N=M=λ=320) para ilustrar a evolução temporal.

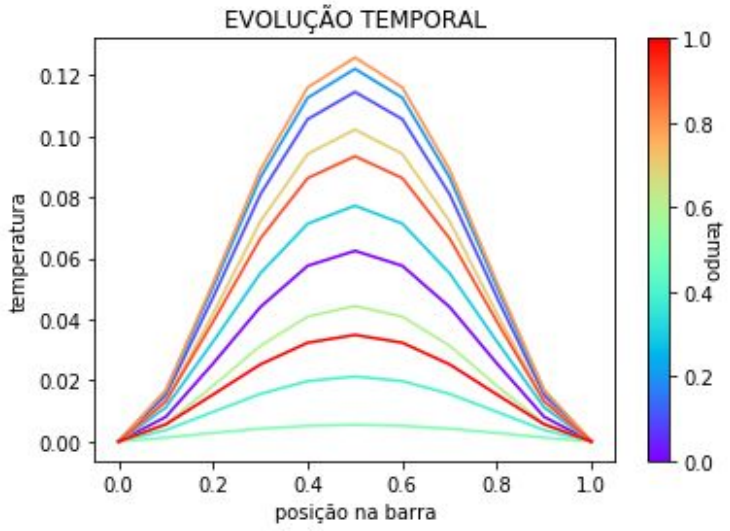


*Fig. 40 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

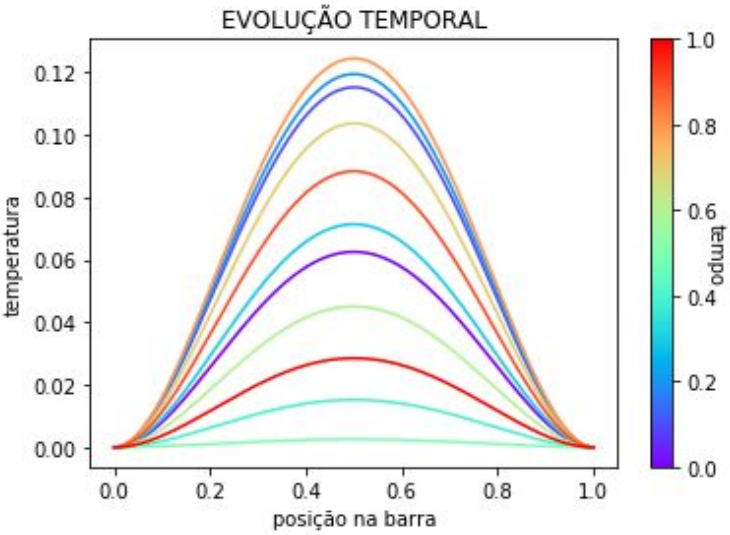


*Fig. 41 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

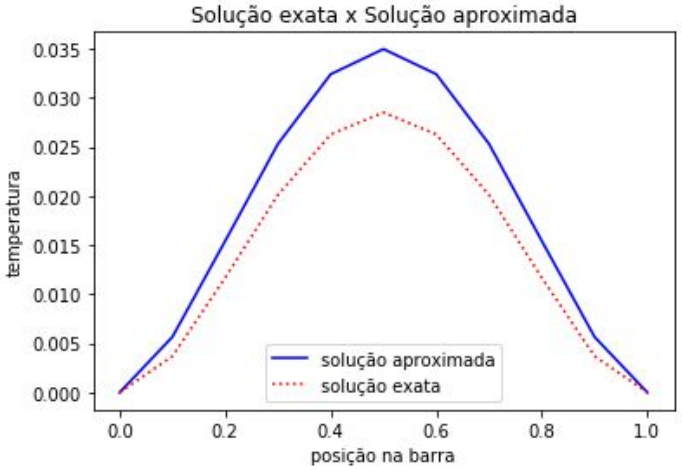


*Fig. 42 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

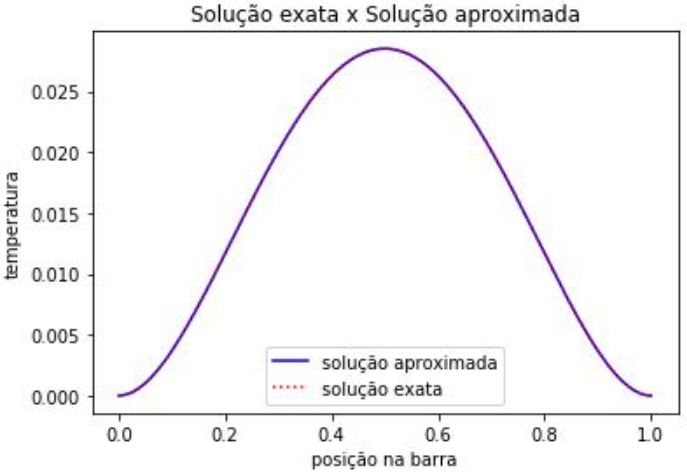


*Fig. 43 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| N | Erro na aproximação | Erro de truncamento |
|---|---|---|
| 10 | 6,453E-03 | 2,871E-02 |
| 20 | 1,569E-03 | 1,022E-02 |
| 40 | 3,893E-04 | 2,936E-03 |
| 80 | 9,716E-05 | 7,804E-04 |
| 160 | 2,428E-05 | 2,008E-04 |
| 320 | 6,069E-06 | 5,089E-05 |
| 640 | 1,517E-06 | 1,281E-05 |

*Tabela 10 - Valores máximos dos erros em T = 1 no caso 2*

Para realizar o cálculo de quanto é o fator de redução do erro no método de Euler implícito, levando em consideração que $\Delta t = \Delta x = 1/N$ e que a convergência é de ordem 2 em $\Delta x$ e em $\Delta t$:

$$max_{k,i} |\tau_i^k (\Delta t,\ \Delta x)| \leq C1\Delta t^2 + C2\Delta x^2,\ com\ C1\ e\ C2\ constantes$$

$$max_{k,i} |\tau_i^k (\Delta t,\ \Delta x)| \leq C1(\frac{1}{N^2}) + C2(\frac{1}{N^2}) \leq C'(\frac{1}{N^2})$$

Ao avaliar o erro em um caso em que N muda, temos que a variação de um caso para outro vai estar associada a um fator $1/N^2$.

Logo, quando o N dobra, é esperado que o erro se reduza para um quarto. Sendo assim, o fato de redução é $\frac{1}{4}$. Essa redução é comprovada através tabela de erros acima.

### 5.2.3. CASO 3

Para o caso 3, são também feitas as mesmas considerações e análises, chegando nas mesmas conclusões.

Na sequência são apresentados apenas os dois exemplos com baixo (N=M=$\lambda$=10) e alto refinamento (N=M=$\lambda$=320) para ilustrar a evolução temporal.

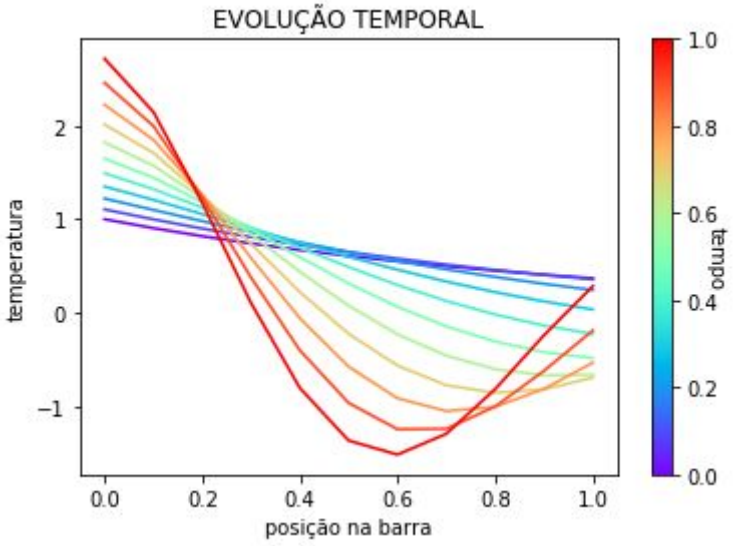


*Fig. 44 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

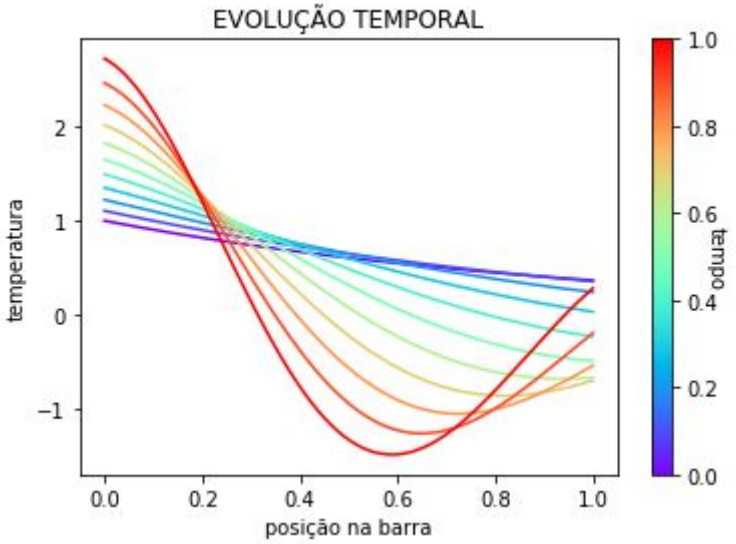


*Fig. 45 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Comparando a solução exata apresentada com a solução aproximada calculada pelo método, para os mesmos casos da evolução temporal, tem-se:

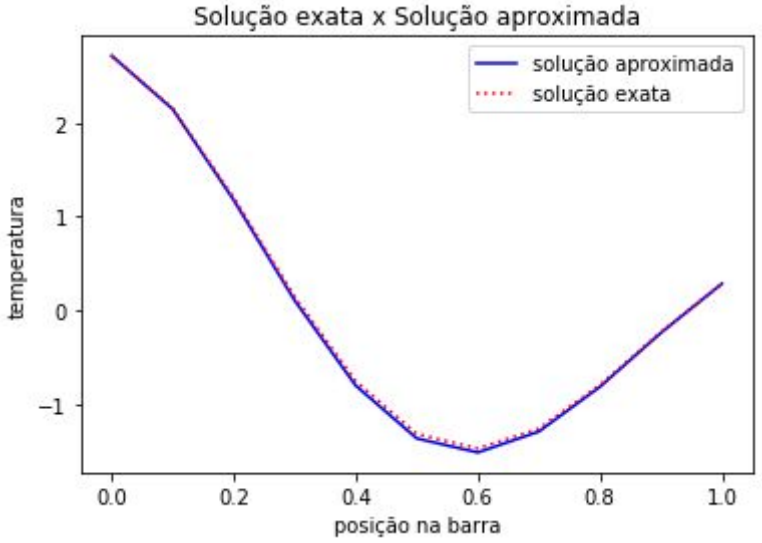


*Fig. 46 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

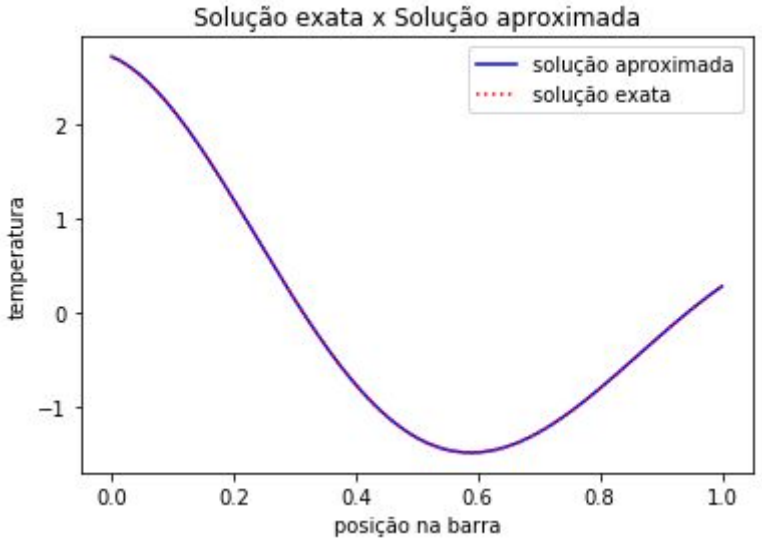


*Fig. 47 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

Executando o programa para os esse caso com todos os parâmetros solicitados, obtém-se a tabela abaixo do valor máximo dos erros em T = 1.

| N | Erro na aproximação | Erro de truncamento |
|---|---|---|
| 10 | 4,778E-02 | 1,257E+00 |
| 20 | 1,199E-02 | 2,732E-01 |
| 40 | 2,993E-03 | 6,417E-02 |
| 80 | 7,488E-04 | 1,616E-02 |
| 160 | 1,872E-04 | 4,133E-03 |
| 320 | 4,679E-05 | 1,044E-03 |
| 640 | 1,170E-05 | 2,625E-04 |

*Tabela 11 - Valores máximos dos erros em T = 1 no caso 3*

Conforme descrito no caso 2, o fator de redução nesse caso também é $\frac{1}{4}$, ou seja, quando o N dobra, é esperado que o erro se reduza pela metade. Essa redução é comprovada através tabela de erros acima.

### 5.2.4. CASO 4

Para o caso 4, são também feitas as mesmas considerações e análises. Como para esse caso não se sabe a solução exata, não é possível plotar gráficos de comparação das soluções.

Na sequência são apresentados apenas os dois exemplos com baixo (N=M=$\lambda$=10) e alto refinamento (N=M=$\lambda$=320) para ilustrar a evolução temporal.

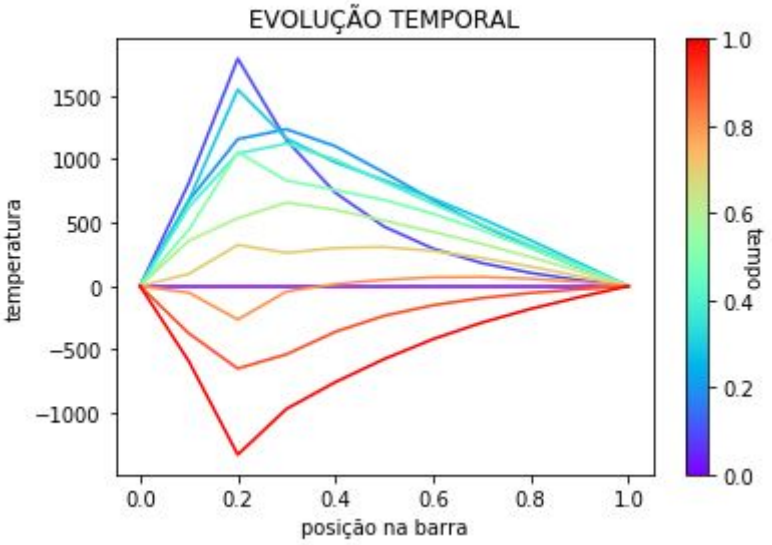


*Fig. 48 - Evolução temporal com N = 10 e M = 10 (λ = 10)*

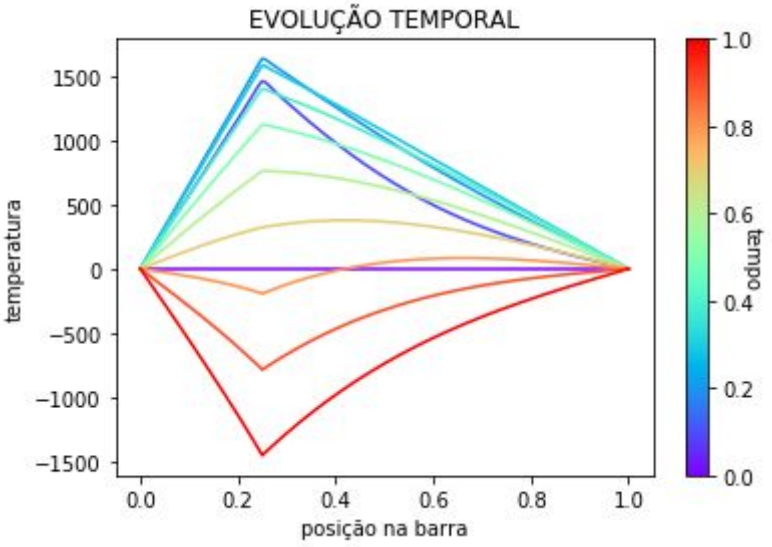


*Fig. 49 - Evolução temporal com N = 320 e M = 320 (λ = 320)*

# 6. CONCLUSÃO

O desenvolvimento do exercício de programação proposto pelo corpo docente, e discutido no presente relatório, permitiu aplicação de conhecimentos teóricos abordados no escopo da disciplina em casos possivelmente reais, ultrapassando, portanto, a barreira de aplicação teórica somente em sala de aula.

Através do exercício, foi possível aplicar três metodologias (Euler explícito, Euler implícito e Crank-Nicolson) para solucionar um problema direto envolvendo a evolução da distribuição de temperatura em uma barra sujeita a fontes de calor, a partir das condições iniciais e de contorno. Foi compreendido que o método de Euler explícito é a aplicação direta da discretização do método de aproximação numérica das derivadas parciais aproximadas por diferenças finitas.

Por outro lado, nos métodos implícitos são aplicados sistemas lineares, em que a solução calculada em um instante depende de outras variáveis nesse mesmo instante, e os sistemas podem ser resolvidos por decomposição de uma matriz. Sendo assim, por mais que sejam mais complexos, como eles são incondicionalmente convergentes e permitem uma maior liberdade na escolha dos parâmetros, e tornam-se mais relevantes para encontrar soluções de problemas reais.

Após terem sido feitas as análises de todos os casos propostos, em todos os métodos, foi possível concluir que, apesar de as soluções calculadas serem similares entre si para os mesmos parâmetros, o aumento do refinamento da malha, ou seja, o aumento do número de passos, faz com que essas soluções se aproximem cada vez mais da solução exata do problema. Com base nisso, pode ser dito que o erro diminui com o aumento do número M, caso seja respeitada, se houver, a condição de convergência. Senão, o erro cresce de maneira absurda.

Em resumo, este trabalho permitiu que fosse desenvolvida uma análise crítica de utilização de ferramentas computacionais e teóricas visando a proposição de soluções que envolvem certa complexidade o que permite obter maior facilidade quando inserida no contexto computacional (associado aos conhecimentos numéricos).

# 7. REFERÊNCIAS BIBLIOGRÁFICAS

ISAACSON, E.; KELLER, H. B. Analisys of Numerical Methods.

ENUNCIADO DO EXERCÍCIO PROGRAMA. Um problema inverso para obtenção de distribuição de Temperatura: subtítulo (se houver). MAP3121, 2020. Disponível em: https://www.ime.usp.br/~map3121/. Acesso em: 2020.

# 8. ANEXO DO CÓDIGO

Aqui se encontra em anexo o código escrito pelos alunos.

```
#############################################################################
#                        EP1 - MAP3121                                      #
#        UM PROBLEMA INVERSO PARA OBTENÇÃO DE TEMPERATURA - PARTE 1         #
#############################################################################
#       ALUNOS:                                                             #
#             ADRIANO CARVALHO E SOUSA      (NUSP: 10773291)              #
#             MARIA TEREZA FERREIRA SILVA   (NUSP: 10769726)              #
#                                                                           #
#############################################################################

# Importando pacotes para trabalhar com funções e gráficos

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors

# Parâmetros globais da solução

t_m = 1  # T
p = 0.25  # Posição da fonte de calor pontual

def main():

    # Cabeçalho
    print("EP1 - MAP3121")
    print("UM PROBLEMA INVERSO PARA OBTENÇÃO DE TEMPERATURA - PARTE 1")
    print("by Adriano e Maria Tereza")
    print()

    # Loop de excecução
    fim = False

    while fim == False:

        # Interação com usuário

        # Seleciona qual dos casos do enunciado
        print("Parâmetros para a solução do problema:")
        print()

        print("Caso 1:")
        print ("T = 1 ; f(t,x) = 10x^2(x-1)-60xt+20t ; u0(x) = 0 ; g1(t) = 0 ; g2(t) = 0")
        print()

        print("Caso 2:")
        print ("T = 1 ; f(t,x) = 10cos(10t) x^2 (1-x)^2 - (1+sin(10t)) (12x^2 - 12x + 2) ; u0(x) = x^2 (1-x)^2 ; g1(t) = 0 ; g2(t) = 0")
        print()

        print("Caso 3:")
        print ("T = 1 ; f(t,x) = -5exp(t-x)((2t+x)sin(5tx) - 5t^2 cos(5tx)) ; u0(x) = exp(-x) ; g1(t) = exp(t) ; g2(t) = exp(t-1)cos(5t)")
        print()

        print("Caso 4:")
        print("T = 1 ; fonte pontual em x = %.2f com intensidade variando em t ; u0(x) = 0 ; g1(t) = 0 ; g2(t) = 0" % p)

        # Lê o caso selecionado
        check = True
        while(check == True):
            # Repete até ter uma entrada válida
            caso = int(input("Qual caso deseja selecionar? "))
            if caso >= 1 and caso <= 4:
                # Verifica se a entrada é valida
                check = False
```

```
            print()

        # Seleciona N e M
        print("Selecione os valores de N e M do problema:")
        n = int(input("Valor de N: "))
        m = int(input("Valor de M: "))
        print()

        # Calcula lambda
        calcula_lambda(n, m)
        print("O valor de lambda é %.2f" % lamb)
        print()

        # Seleciona o método de resolução
        print("Selecione o método de resolução:")
        print()
        print("1) Método de Euler explícito")
        print("2) Método de Euler implícito")
        print("3) Método de Crank-Nicolson")

        # Lê o método a ser utilizado
        check = True
        while(check == True):
            # Repete até ter uma entrada válida
            metodo = int(input("Qual método deseja utilizar? "))
            if metodo >= 1 and metodo <= 3:
                # Verifica se a entrada é valida
                check = False

        # Resolve o sistema de acordo com os parâmetros solicitados
        u = resolve(n, m, caso, metodo)

        # Gera o gáfico com os resultados
        resultados(u)

        # Verificando se o caso tem a solução exata conhecida
        if caso == 1 or caso == 2 or caso == 3:
            # Verifica a necessidade de se informar do erro
            sel = input("Deseja comparar a solução exata e a aproxiamda em T = 1 (s/n)? ")
            if (sel == 's'):
                compara(caso, metodo, n, m, u[-1])

        # Verifica a necesssidade de uma nova excecução
        sel = input("Deseja excecutar novamente (s/n)? ")
        if not (sel == 's'):
            fim = True

    return

def calcula_lambda(n, m):

    # Esta função calcula lambda a partir de n e m

    # Definindo variáveis globais que serão usadas em quase todas as funções
    global delta_x
    global delta_t
    global lamb

    # Calculando delta t e delta x
    delta_x = 1 / n
    delta_t = t_m / m

    # Calculando lambda
    lamb = delta_t / (delta_x ** 2)

    return

def resolve(n, m, caso, metodo):

    # Esta função decide qual o método de resolução será utilizado
```

```
    if metodo == 1:
        # Equação discretizada
        u = resolve_ee(n, m, caso)

    elif metodo == 2:
        # Método de Euler implícito
        u = resolve_ei(n, m, caso)

    elif metodo == 3:
        # Método de Crank-Nicolson
        u = resolve_cn(n, m, caso)

    else:
        # Número do método inválido
        u = False

    return u

def resultados(u):

    valores_x = np.linspace(0, 1, num = len(u[0]))

    plt.figure()

    norm = colors.Normalize(vmin=0, vmax=1)
    cmap = plt.cm.rainbow

    escala, grafico = plt.subplots()

    for t in range(0, 11):
        grafico.plot(valores_x, u[t], color=cmap(norm(t/10)))

    plt.title('EVOLUÇÃO TEMPORAL')
    plt.xlabel('posição na barra')
    plt.ylabel('temperatura')

    sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
    cbar = escala.colorbar(sm)
    cbar.set_label('tempo', rotation=270)

    plt.show()

    return

def resolve_ee(n, m, caso):

    # Esta função resolve o problema pelo método de Euler explícito

    # Obtendo as condições do problema solicitado
    g1 = calcula_g1(caso, m)
    g2 = calcula_g2(caso, m)

    # Criando vetor atual para solução e atribuíndo a ele os valores de u0
    u_atual = calcula_u0(caso, n)

    # Criando a matriz com as 11 soluções solicitadas
    u = []
    u.append(u_atual.copy())  # Incluindo a solução em t = 0

    # Aplicando o método de solução da equação
    for k in range(1, m + 1):

        # Percorrendo todos os instantes de tempo
        u_anterior = u_atual.copy()  # Atualiza o vetor
        u_atual[0] = g1[k] # Contorno em x = 0 -- primeira posição do vetor
        u_atual[-1] = g2[k] # Contorno em x = 1 -- última posição do vetor

        for i in range(1, n):

            # Aplicando equação da solução
```

```
            u_atual[i] = u_anterior[i] + delta_t * (((u_anterior[i - 1] - 2 * u_anterior[i] + u_anterior[i + 1]) / delta_x ** 2) + func_f(caso,
delta_x * i , delta_t * (k - 1)))

        for j in range(1, 11):

            # Verifica se o resultado é um dos que entrará no gráfico
            if k == j * m // 10:
                u.append(u_atual.copy())

    return u

def resolve_ei(n, m, caso):

    # Esta função resolve o problema pelo método de Euler implícito

    # Obtendo as condições do problema solicitado
    g1 = calcula_g1(caso, m)
    g2 = calcula_g2(caso, m)

    # Criando matriz A a partir de 2 vetores
    diagonal_mA = [(1 + 2 * lamb) for i in range(n - 1)]
    subdiagonal_mA = [(-lamb) for i in range(n - 1)]

    # Decompondo matriz A
    vetor_l, vetor_d = decomposicao(diagonal_mA, subdiagonal_mA)

    # Criando vetor atual para solução e atribuíndo a ele os valores de u0
    u_atual = calcula_u0(caso, n)

    # Criando a matriz com as 11 soluções solicitadas
    u = []
    u.append(u_atual.copy())  # Incluindo a solução em t = 0

    # Aplicando o método de solução da equação
    for k in range(1, m + 1):

        u_anterior = u_atual.copy()  # Atualiza o vetor

        # Gerando vetor b
        b = [(u_anterior[i] + delta_t * func_f(caso, delta_x * i, delta_t * k)) for i in range(1, n)]
        b[0] += (lamb * g1[k])
        b[-1] += (lamb * g2[k])

        # Aplicando condições de contorno e resolvendo o sistema
        u_atual[0] = g1[k]  # Contorno em x = 0 -- primeira posição do vetor
        u_atual[-1] = g2[k]  # Contorno em x = 1 -- última posição do vetor
        u_atual[1 : n] = solucao_LDLt(vetor_l, vetor_d, b)  # Resolvendo o sistema

        for j in range(1, 11):
            # Verifica se o resultado é um dos que entrará no gráfico
            if k == j * m // 10:
                u.append(u_atual.copy())

    return u

def resolve_cn(n, m, caso):

    # Esta função resolve o problema pelo método de Crank-Nicolson

    # Obtendo as condições do problema solicitado
    g1 = calcula_g1(caso, m)
    g2 = calcula_g2(caso, m)

    # Criando matriz A a partir de 2 vetores
    diagonal_mA = [(1 + lamb) for i in range(n - 1)]
    subdiagonal_mA = [(-lamb / 2) for i in range(n - 1)]

    # Decompondo matriz A
    vetor_l, vetor_d = decomposicao(diagonal_mA, subdiagonal_mA)

    # Criando vetor atual para solução e atribuíndo a ele os valores de u0
```

```
    u_atual = calcula_u0(caso, n)

    # Criando a matriz com as 11 soluções solicitadas
    u = []
    u.append(u_atual.copy())  # Incluindo a solução em t = 0

    # Aplicando o método de solução da equação
    for k in range(1, m + 1):

        u_anterior = u_atual.copy()  # Atualiza o vetor

        # Gerando o vetor b
        b = [(u_anterior[i] + ((u_anterior[i - 1] - 2 * u_anterior[i] + u_anterior[i + 1]) * lamb / 2) + ((func_f(caso, delta_x * i, delta_t * k) +
func_f(caso, delta_x * i, delta_t * (k - 1))) * delta_t / 2)) for i in range(1, n)]
        b[0] += (lamb * g1[k] / 2)
        b[-1] += (lamb * g2[k] / 2)

        # Aplicando condições de contorno e resolvendo o sistema
        u_atual[0] = g1[k]  # Contorno em x = 0 -- primeira posição do vetor
        u_atual[-1] = g2[k]  # Contorno em x = 1 -- última posição do vetor
        u_atual[1 : n] = solucao_LDLt(vetor_l, vetor_d, b)  # Resolvendo o sistema

        for j in range(1, 11):
            # Verifica se o resultado é um dos que entrará no gráfico
            if k == j * m // 10:
                u.append(u_atual.copy())

    return u


def calcula_u0(caso, n):

    # Esta função calcula os valores de u0 (temperatura inicial)

    u0 = [0 for i in range(n + 1)]

    if caso == 2:
        for i in range(len(u0)):
            # Percorre cada valor discreto de x
            x = delta_x * i
            u0[i] = x ** 2 * (1 - x) ** 2
    elif caso == 3:
        for i in range(len(u0)):
            # Percorre cada valor discreto de x
            x = delta_x * i
            u0[i] = np.exp(-x)
    else:
        for i in range(len(u0)):
            # Percorre cada valor discreto de x
            x = delta_x * i
            u0[i] = 0

    return u0

def calcula_g1(caso, m):

    # Esta função calcula os valores de g1 (condição de contorno)

    g1 = [0 for k in range(m + 1)]

    if caso == 3:
        for k in range(len(g1)):
            # Percorre cada valor discreto de t
            t = delta_t * k
            g1[k] = np.exp(t)
    else:
        for k in range(len(g1)):
            # Percorre cada valor discreto de t
            t = delta_t * k
            g1[k] = 0
```

```
    return g1

def calcula_g2(caso, m):

    # Esta função calcula os valores de g2 (condição de contorno)

    g2 = [0 for k in range(m + 1)]

    if caso == 3:
        for k in range(len(g2)):
            # Percorre cada valor discreto de t
            t = delta_t * k
            g2[k] = np.exp(t - 1) * np.cos(5 * t)
    else:
        for k in range(len(g2)):
            # Percorre cada valor discreto de t
            t = delta_t * k
            g2[k] = 0

    return g2

def func_f(caso, x, t):

    # Esta função calcula os valores de f (função da fonte de calor)

    if caso == 1:
        # Calcula os valores de f do caso 1
        f = 10 * x ** 2 * (x - 1) - 60 * x * t + 20 * t
    elif caso == 2:
        # Calcula os valores de f do caso 2
        f = 10 * np.cos(10 * t) * x ** 2 * (1 - x) ** 2 - (1 + np.sin(10 * t)) * (12 * x ** 2 - 12 * x + 2)
    elif caso == 3:
        # Calcula os valores de f do caso 3
        f = -5 * np.exp(t - x) * ((2 * t + x) * np.sin(5 * t * x) - 5 * t ** 2 * np.cos(5 * t * x))
    elif caso == 4:
        # Calcula os valores de f do caso 4
        if x >= (p - delta_x / 2) and x <= (p + delta_x / 2):
            f = 10000 * (1 - 2 * t ** 2) * 1 / delta_x
        else:
            f = 0

    return f

def decomposicao(diagonal, subdiagonal):

    # Esta função decompõem a matriz A no produto de matrizes LDLt

    vd =  [0 for i in range(len(diagonal))]  # Matriz diagonal D representada por apenas um vetor
    vl =  [0 for i in range(len(diagonal))]  # Matriz L que tem 1s na diagonal principal e o restante dos valores serão armazenados
nesse vetor

    for i in range(len(vd)):
        # Definimos um caso inicial, em seguida, calcula-se o restante a partir dele
        if i == 0:
            # Caso inicial
            vd[0] = diagonal[0]
            vl[0] = subdiagonal[0] / vd[0]
        else:
            # Restante dos valores
            vd[i] = diagonal[i] - vd[i - 1] * vl[i - 1] ** 2
            vl[i] = subdiagonal[i] / vd[i]

    return vl, vd

def solucao_LDLt(vl, vd, b):

    # Esta função resolve o sistema pelo método LDLt

    # Soluções final (x) e intermediárias (y e z)
    x = [0 for i in range(len(b))]
    y = [0 for i in range(len(b))]
```

```
    z = [0 for i in range(len(b))]

    # Resolvendo Lz = b:
    z[0] = b[0]
    for i in range(1, len(b)):
        z[i] = b[i] - vl[i - 1] * z[i - 1]

    # Resolvendo Dy = z:
    for i in range(len(b)):
        y[i] = z[i] / vd[i]

    # Finalmente, resolvendo Ltx = y:
    x[-1] = y[-1]
    for i in reversed(range(0, len(b) - 1)):
        x[i] = y[i] - vl[i] * x[i + 1]

    return x

def compara(caso, metodo, n, m, s_aprox):

    # Esta função calula os erros em T = 1 e plota os gráficos da solução exata e aproximada nesse instante

    valores_x = np.linspace(0, 1, num = len(s_aprox))  # Valores de x para plotar o gráfico

    s_exata = []  # Solução exata da equação segundo as condições solicitadas (em T = 1, ou seja, k = m)
    s_exata_proximo = []  # Solução exata da equação para k = m + 1, necessária para o cálculo do erro de truncamento em T = 1

    if caso == 1:

        for i in range(len(valores_x)):
            # Preenchendo o vetor com os valores da solução para o caso 1
            s_exata.append(10 * (valores_x[i] ** 2) * (valores_x[i] - 1))
            s_exata_proximo.append(10 * (m + 1) * delta_t * (valores_x[i] ** 2) * (valores_x[i] - 1))

    if caso == 2:

        for i in range(len(valores_x)):
            # Preenchendo o vetor com os valores da solução para o caso 2
            s_exata.append((1 + np.sin(10)) * valores_x[i] ** 2 * (1 - valores_x[i]) ** 2)
            s_exata_proximo.append((1 + np.sin(10 * (m + 1) * delta_t)) * valores_x[i] ** 2 * (1 - valores_x[i]) ** 2)

    if caso == 3:

        for i in range(len(valores_x)):
            # Preenchendo o vetor com os valores da solução para o caso 3
            s_exata.append(np.exp(1 - valores_x[i]) * np.cos(5 * valores_x[i]))
            s_exata_proximo.append(np.exp((m + 1) * delta_t - valores_x[i]) * np.cos(5 * (m + 1) * delta_t * valores_x[i]))

    # Chamando as funções que calculam os erros
    et = erro_trun(s_exata, s_exata_proximo, metodo, caso)
    ea = erro_aprox(s_exata, s_aprox)

    # Gerando gráfico comparativo entre as soluções exata e aproximada

    plt.figure()

    plt.plot(valores_x, s_aprox, color = 'b', label = 'solução aproximada')
    plt.plot(valores_x, s_exata, 'k:', color = 'r', label = 'solução exata')

    plt.title('Solução exata x Solução aproximada')
    plt.xlabel('posição na barra')
    plt.ylabel('temperatura')
    plt.legend()

    plt.show()

    # Imprimimindo os valores calculados para os erros nesse caso

    print()
    print("O módulo do erro de truncamento é: %g" % et)
    print("O módulo do erro na aproximação da solução exata é: %g" % ea)
```

```
    return

def erro_trun(k, kpp, metodo, caso):

    # Esta função calcula o erro de truncamento em T = 1

    erro = 0
    if metodo == 1:
        for i in range(1, len(k) - 1):
            # Calculando o erro de truncamento para o método de euler explícito
            tmp = abs(((kpp[i] - k[i]) / delta_t) - ((k[i - 1] - 2 * k[i] + k[i + 1]) / delta_x ** 2) - func_f(caso, delta_x * i, t_m))
            if tmp > erro:
                # Buscando o valor máximo
                erro = tmp
    if metodo == 2:
        for i in range(1, len(k) - 1):
            # Calculando o erro de truncamento para o método de euler implícito
            tmp = abs(((kpp[i] - k[i]) / delta_t) - ((kpp[i - 1] - 2 * kpp[i] + kpp[i + 1]) / delta_x ** 2) - func_f(caso, delta_x * i, t_m + delta_t))
            if tmp > erro:
                # Buscando o valor máximo
                erro = tmp
    if metodo == 3:
        for i in range(1, len(k) - 1):
            # Calculando o erro de truncamento para o método de Crank-Nicolson
            tmp = abs(((kpp[i] - k[i]) / delta_t) - (((k[i - 1] - 2 * k[i] + k[i + 1]) + (kpp[i - 1] - 2 * kpp[i] + kpp[i + 1])) / (2 * delta_x ** 2)) -
(func_f(caso, delta_x * i , t_m + delta_t) + func_f(caso, delta_x * i , t_m)) / 2)
            if tmp > erro:
                # Buscando o valor máximo
                erro = tmp

    return erro

def erro_aprox(exata, aproximada):

    # Esta função calcula o erro de aproximação da solucão exata em T = 1

    erro = 0
    for i in range(len(exata)):
        tmp = abs(exata[i] - aproximada[i])  # Erro de aproximação dado pela diferença entre a solução exata e a aproximada
        if tmp > erro:
            # Buscando o valor máximo do erro
            erro = tmp
    return erro

main()
```