

MAP3121 - Métodos Numéricos e Aplicações

**Exercício Programa 1: Um problema inverso para obtenção de
distribuição de Temperatura (Parte 1)**



Professor Teórico:
Clodoaldo Grotta Ragazzo

Integrantes:
Marina Botelho de Mesquita - 10771156
Murilo Costa Campos de Moura - 10705763

Data:
31/03/2020

SUMÁRIO

Introdução e Conceitos Iniciais	3
Primeira Tarefa	4
Introdução ao Método Explícito	4
Item a	5
Item b	10
Item c	15
Segunda Tarefa	18
Introdução ao Método Implícito	18
Item a - Funções de Decomposição de Matriz e Resolução de Sistema	18
Item b - Método de Euler Implícito	22
Equação 1: $u(t,x)=(1+\text{sen}(10t))x^2(1-x)^2$	23
Equação 2: $u(t,x)=et-x\cos(5tx)$	25
Equação 3:	26
Item c - Método de Crank-Nicolson	27
Equação 1: $u(t,x)=(1+\text{sen}(10t))x^2(1-x)^2$	29
Equação 2: $u(t,x)=et-x\cos(5tx)$	30
Equação 3	32
Considerações finais	34
Referências Bibliográficas	34

1. Introdução e Conceitos Iniciais

Este trabalho tem por objetivo o estudo da evolução da distribuição de temperatura em uma barra sujeita a fontes de calor, a partir de uma dada distribuição inicial.

A primeira parte, que será o foco deste relatório e motivação para o Exercício Programa 1, é o estudo do problema direto. Já segunda parte, que será tratada futuramente no Exercício Programa 2, é o estudo do problema inverso.

Problemas inversos são opostos aos problemas diretos. Ou seja, ao passo que o problema direto busca estudar o efeito tendo em vista uma causa, o inverso busca verificar a causa partindo-se do efeito.

Sabe-se que a evolução da distribuição de temperatura pode ser descrita pela seguinte equação diferencial parcial, na qual t é a variável no tempo, x é a variável no espaço (comprimento da barra normalizado para 1) e a função f descreve as fontes de calor:

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

Além disso, as condições de contorno da equação também são conhecidas e estão explicitadas abaixo:

$$\begin{aligned} u(0, x) &= u_0(x) \quad \text{em } [0, 1] \\ u(t, 0) &= g_1(t) \quad \text{em } [0, T] \\ u(t, 1) &= g_2(t) \quad \text{em } [0, T] \end{aligned}$$

O objetivo do nosso estudo é discretizar a equação diferencial parcial acima para que possamos aproximar-a numericamente, bem como observar o erro na aproximação. Para conseguir tal aproximação, vamos nos basear nas expansões de Taylor para aproximar as derivadas parciais por diferenças finitas. Para a discretização definiremos $x_i = i\Delta x_i$, $i = 0, \dots, N$ com $\Delta x = 1/N$ e $t_k = k\Delta t$, $k = 1, \dots, M$ com $\Delta t = T/M$. Além disso a notação utilizada para a aproximação de $u(t_k, x_i)$ será u_i^k .

Assim a condição inicial é:

$$u_i^0 = u_0(x_i), \quad i = 0, \dots, N$$

E as condições na fronteira da barra no instante t_k são dadas por:

$$u_0^k = g_1(t_k) \quad \text{e} \quad u_N^k = g_2(t_k), \quad k = 1, \dots, M$$

2. Primeira Tarefa

2.1. Introdução ao Método Explícito

O primeiro método que utilizaremos para resolver o problema direto será o Método Explícito. Para este método, a distribuição de temperatura nos pontos da barra que não pertençam a fronteira é dada por:

$$u_i^{k+1} = u_i^k + \Delta t \left(\frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2} + f(x_i, t_k) \right), \quad i = 1, \dots, N-1, \text{ e } k = 0, \dots, M-1$$

Nota-se que a aproximação descrita acima se trata de um método explícito iterativo, no qual são utilizados valores de temperatura no instante anterior k para posição em questão i , para posição anterior $i-1$ e para posição seguinte $i+1$ com o intuito de determinar a temperatura desejada u_i^{k+1} . Por se tratar de uma aproximação, é importante levar em consideração o erro, oriundo da diferença entre a função exata e a aproximação, como explicitado abaixo:

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

Nos testes referentes a este relatório, o cálculo do erro será realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste ao qual diz respeito. O valor do erro máximo será o considerado como erro do teste.

Neste modelo, temos que se Δt e Δx tenderem a zero a aproximação converge para a equação exata, uma vez que o erro também tende a zero. Para o exercício, estamos sob hipótese de que:

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

Assim sendo, esse método se apresenta como condicionalmente convergente, uma vez que só se a condição acima for respeitada, a solução é estável.

Para o Método Explícito, os diferentes testes descritos nos próximos itens foram implementados com a seguinte função presente no programa:

```

def explicito(N,M,teste):
    """Fornece a solução aproximada usando o método explícito

    Parameters:
    N (int): número de passos em x
    M (int): número de passos em t
    teste (int): qual a equação que deverá ser resolvida (conforme está no LEIAME.txt)

    Returns:
    grafico(array): vetor com o valor da solução aproximada a cada 0.1s

"""

```

2.2. Item a

Iniciaremos os estudos do Método Explícito tomando como exemplo a seguinte solução exata, suas condições de fronteira e inicial e a seguinte fonte:

$$\begin{aligned}
u(t, x) &= 10tx^2(x - 1) \\
u_0(x) &= u(0, x) = 0 \\
g_1(t) &= u(t, 0) = 0 \\
g_2(t) &= u(t, 1) = 0 \\
f(t, x) &= u_t(t, x) - u_{xx}(t, x) = 10x^2(x - 1) - 60xt + 20t
\end{aligned}$$

O erro de truncamento para este exemplo pode ser calculado da seguinte forma:

$$\begin{aligned}
\tau(\Delta t, \Delta x) &= \max_{k,i} |\tau_i^k(\Delta t, \Delta x)| \leq C_1 \Delta t + C_2 \Delta x^2 \\
C_1 &= \max_D \left| \frac{u_{tt}(t,x)}{2} \right| \quad \text{e} \quad C_2 = \max_D \left| \frac{u_{xxxx}(t,\bar{x})}{4!} \right|
\end{aligned}$$

Uma vez que:

$$u_{tt}(t, x) = 0 \quad \text{e} \quad u_{xxxx}(t, x) = 0$$

Temos que:

$$\tau(\Delta t, \Delta x) = 0$$

Para realizar os primeiros testes do Método Explícito, vamos considerar a solução exata, condições de fronteira e função da fonte descritas a seguir:

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

$$u_0(x) = u(0, x) = x^2(1 - x)^2$$

$$g_1(t) = u(t, 0) = 0$$

$$g_2(t) = u(t, 1) = 0$$

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

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

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

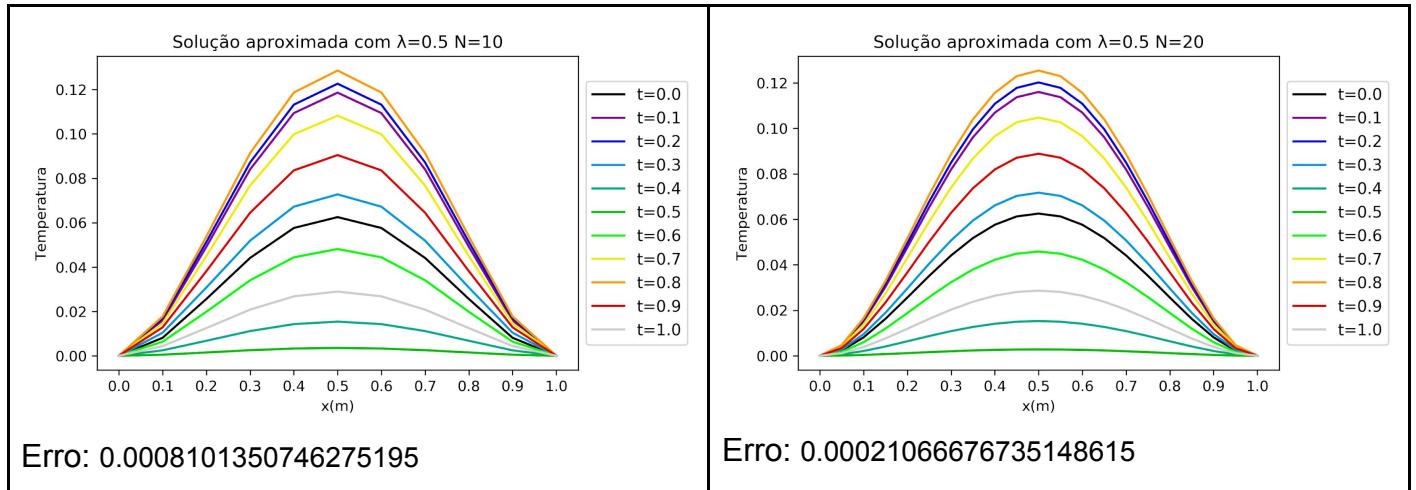
Percebe-se que a solução exata nesse caso é $u(t, x) = (1 + \sin(10t))x^2(1 - x)^2$, uma vez que corresponde às condições impostas pela equação diferencial parcial do calor. Nessas condições, realizaremos testes para $N = 10, 20, 40, 80, 160$ e 320 e para $\lambda = 0.5$ e $\lambda = 0.25$.

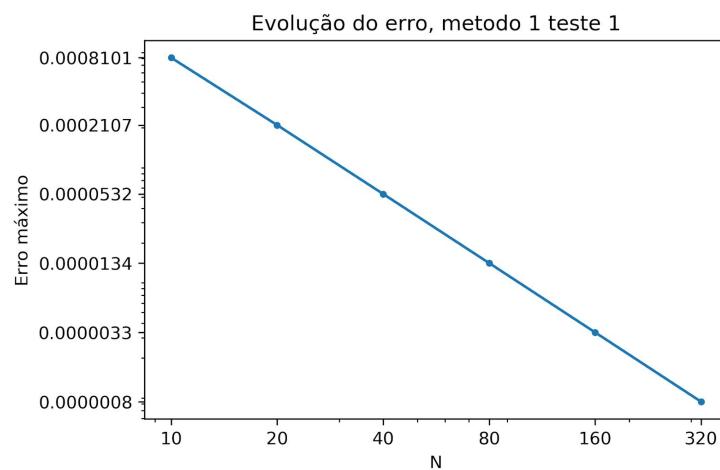
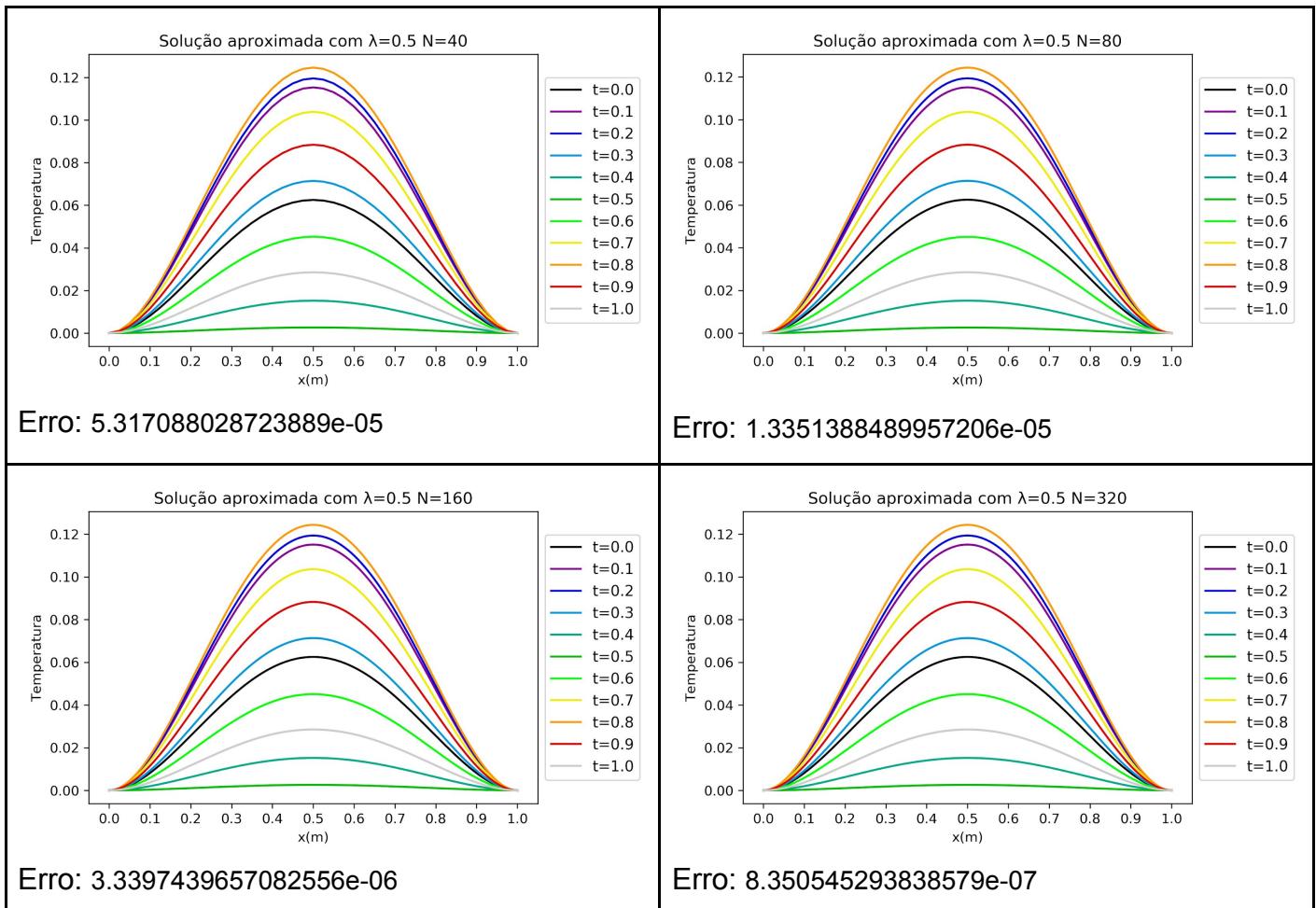
O cálculo do erro é realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste. O valor retornado é o máximo calculado em cada teste. O cálculo utilizado para os testes deste item é:

$$e_i^k = |(1 + \sin(10t))x^2(1 - x)^2 - u_i^k|$$

Os gráficos referentes à variação de temperatura ao longo da barra para cada teste e o comportamento do erro podem ser observados a seguir:

Para $\lambda = 0.5$:

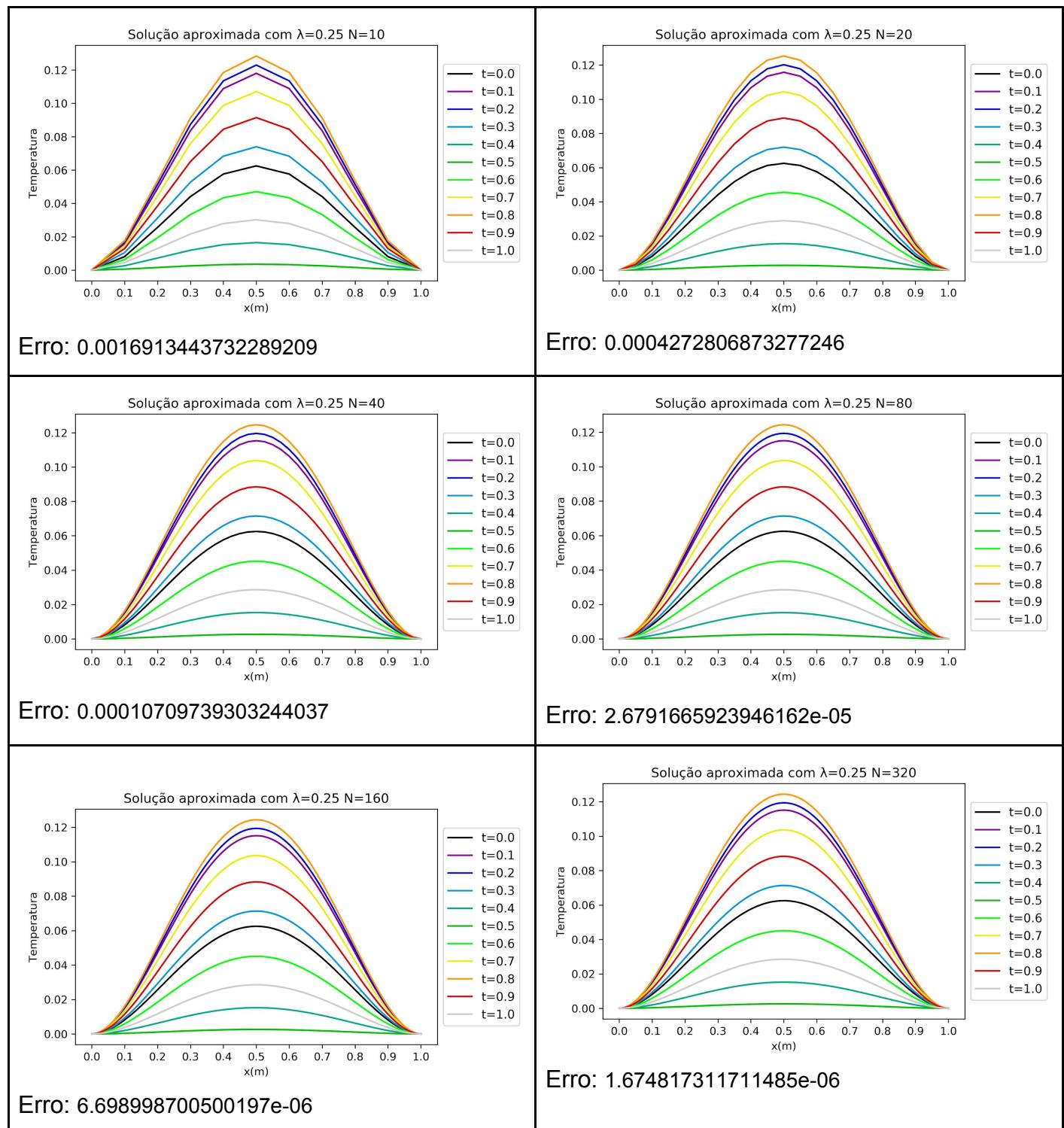


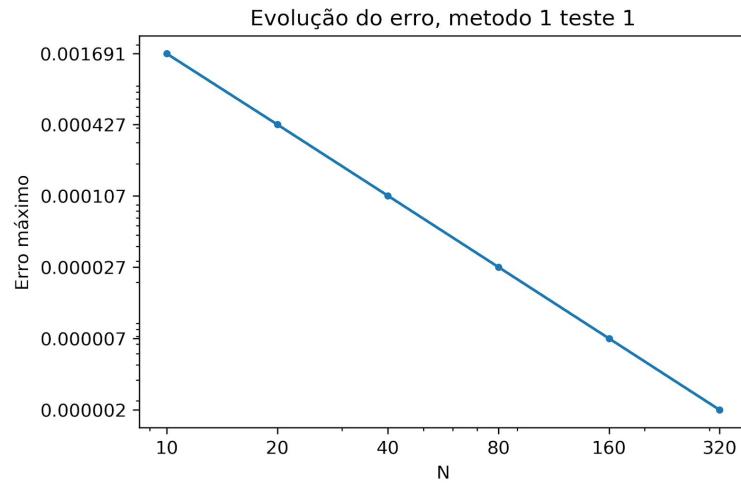


O fator de redução do erro a cada refinamento da malha foi: 3.84557605; 3.96207033; 3.98242328; 3.99772816 e 3.99943219.

Pode-se perceber que o valor se aproxima de 4, o que é coerente, dado que o erro de truncamento é de ordem 1 em Δt e quando N dobrou, Δt é 4 vezes menor.

Para $\lambda = 0.25$:

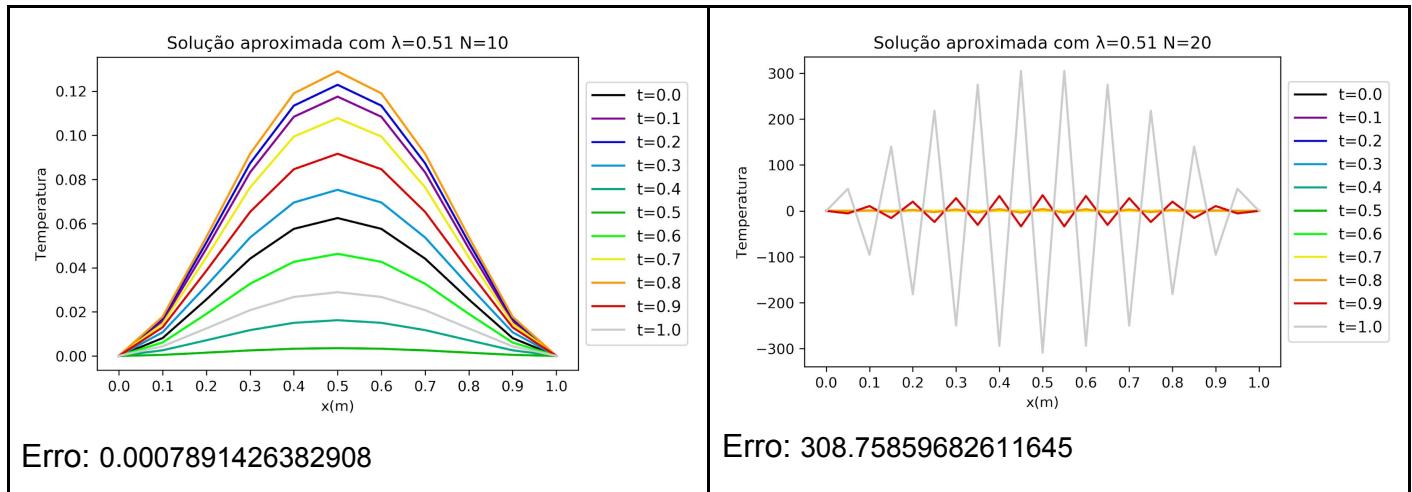


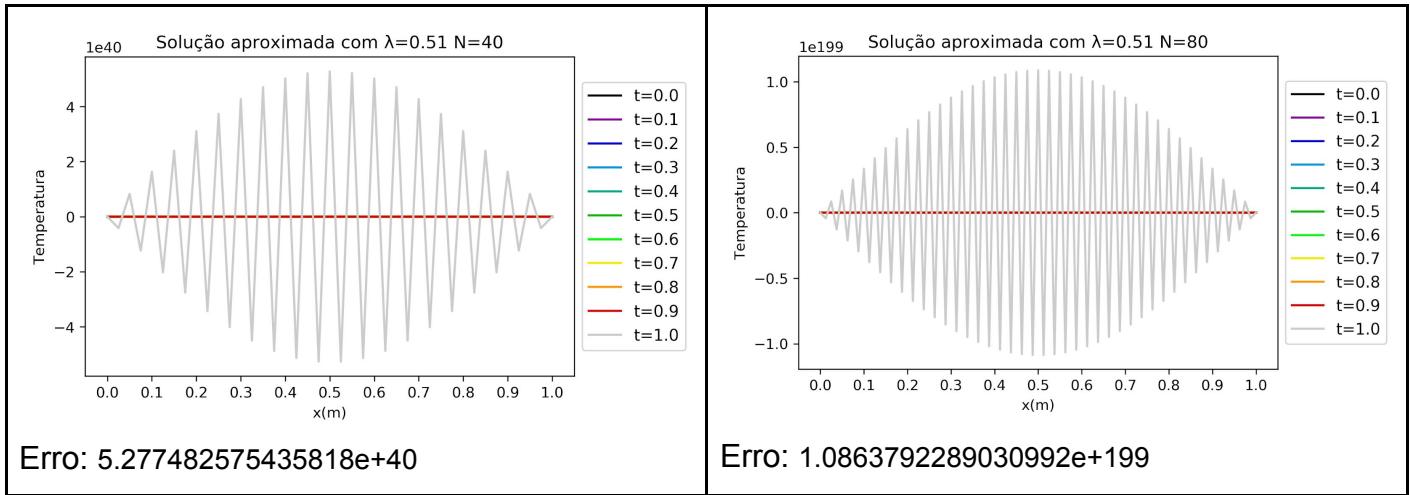


Fator de redução do erro a cada refinamento da malha: 3.95839181; 3.98964601
3.99741447; 3.9993538 e 3.99983846]

Pode-se perceber que o valor se aproxima de 4, o que faz sentido, dado que o erro de truncamento é de ordem 1 em Δt e quando N dobra, Δt é 4 vezes menor.

Ao realizar os mesmos testes para os diversos valores de N , porém com $\lambda = 0.51$, o Método Explícito fica instável, por não ser respeitada a condição de convergência descrita anteriormente. Assim sendo, o erro passa a crescer drasticamente, como apresentado pelos gráficos a seguir:





O número de passos (M) necessários para realizar cada teste depende de N, segundo a fórmula explicitada a seguir.

$$M = \frac{N^2}{\lambda}$$

Nota-se que ao dobrar N mantendo constante o valor de λ , o número de passos é quadruplicado. Dessa forma, para valores de $N = 640$ e para $\lambda = 0.5$ e $\lambda = 0.25$ temos o explicitado a seguir:

$$M = \frac{640^2}{0.5} = 819200$$

$$M = \frac{640^2}{0.25} = 1638400$$

2.3. Item b

Para uma nova sequência de testes, tomamos como solução exata:

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

A partir das definições descritas na introdução, podemos determinar $u_0(x)$, $g_1(t)$, $g_2(t)$ e $f(t, x)$ como:

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

$$g_1(t) = u(t, 0) = e^t \cos(0) = e^t$$

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

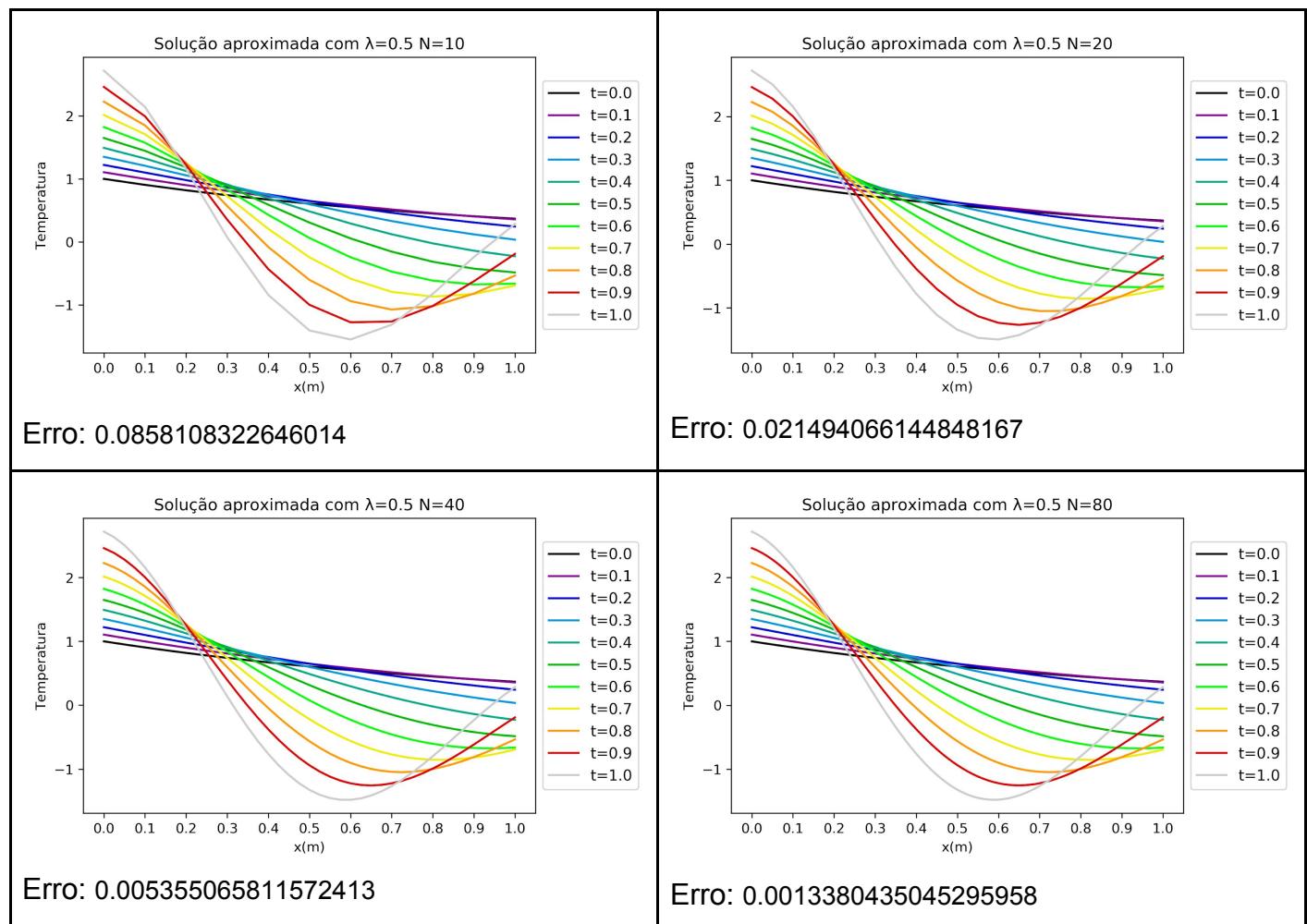
$$f(t, x) = u_t(t, x) - u_{xx}(t, x) = e^{t-x} [25t^2 \cos(5tx) - \sin(5tx)(10t + 5x)]$$

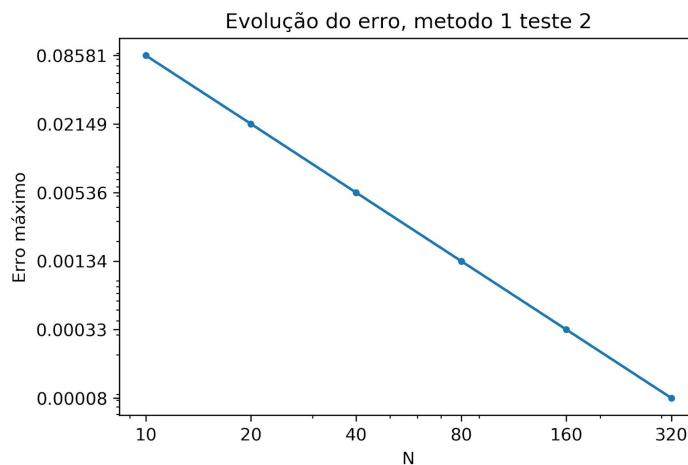
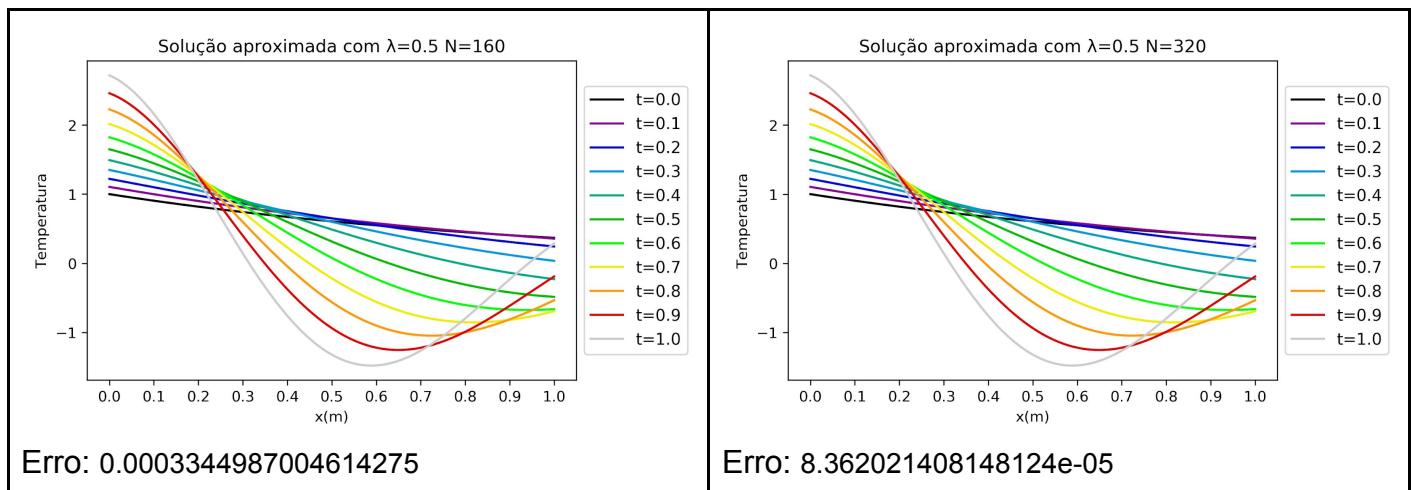
Nessas condições, mais uma vez, realizaremos testes para $N = 10, 20, 40, 80, 160$ e 320 e para $\lambda = 0.5$ e $\lambda = 0.25$. O cálculo do erro é realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste. O valor retornado é o máximo calculado em cada teste. O cálculo utilizado para os testes deste item é:

$$e_i^k = |e^{1-x} \cos(5x) - u_i^k|$$

Os gráficos referentes à variação de temperatura ao longo da barra para cada teste e comportamento do erro podem ser observados a seguir:

Para $\lambda = 0.5$:

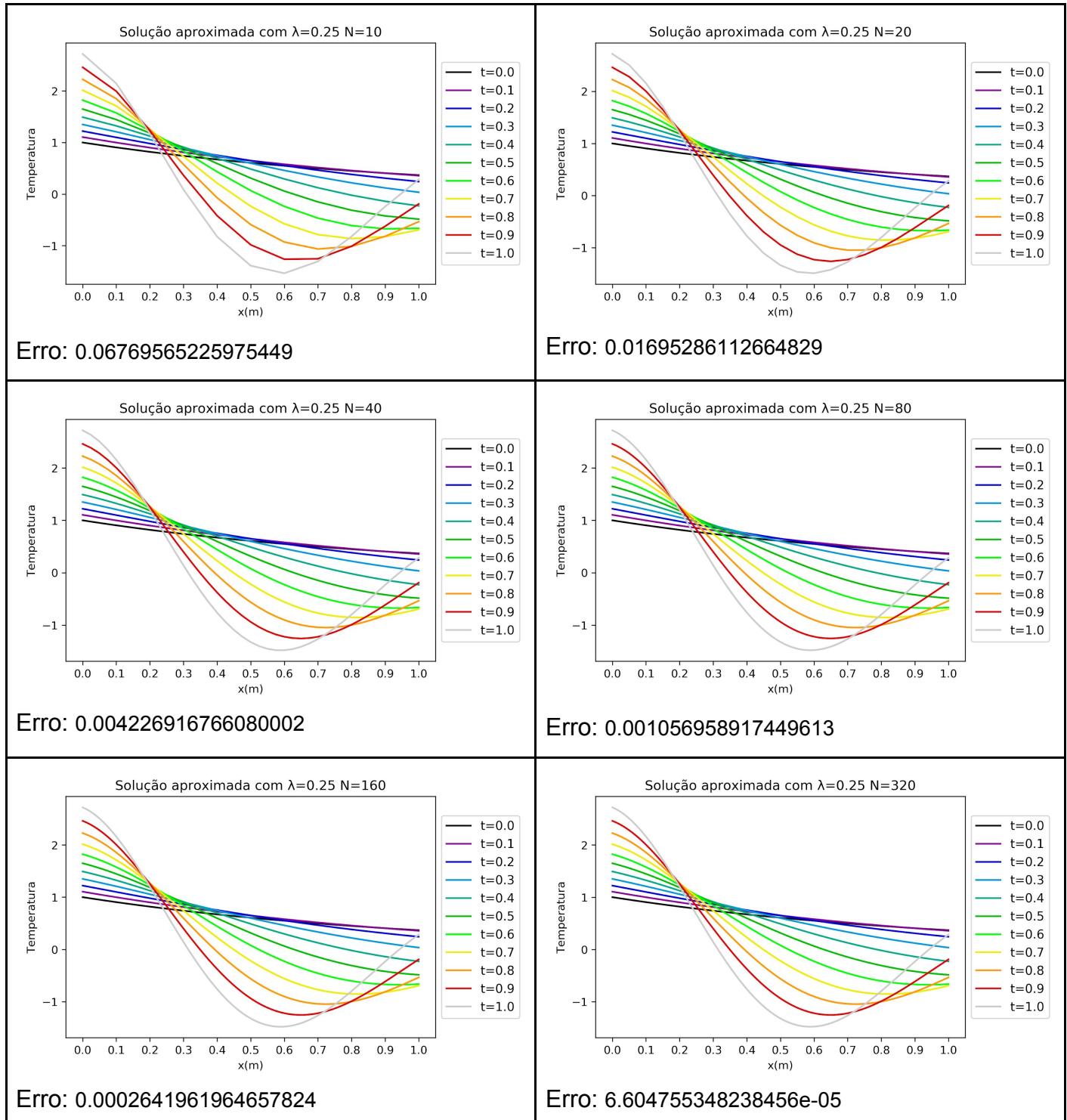


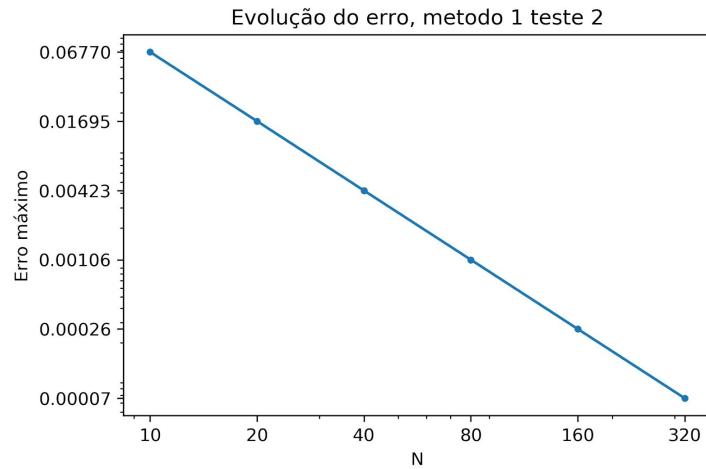


Fator de redução do erro a cada refinamento da malha: 3.99230335; 4.01378188; 4.00216121; 4.0001456 e 4.00021339.

Pode-se perceber que o valor se aproxima de 4, o que faz sentido, dado que o erro de truncamento é de ordem 1 em Δt e quando N dobra, Δt é 4 vezes menor.

Para $\lambda = 0.25$:

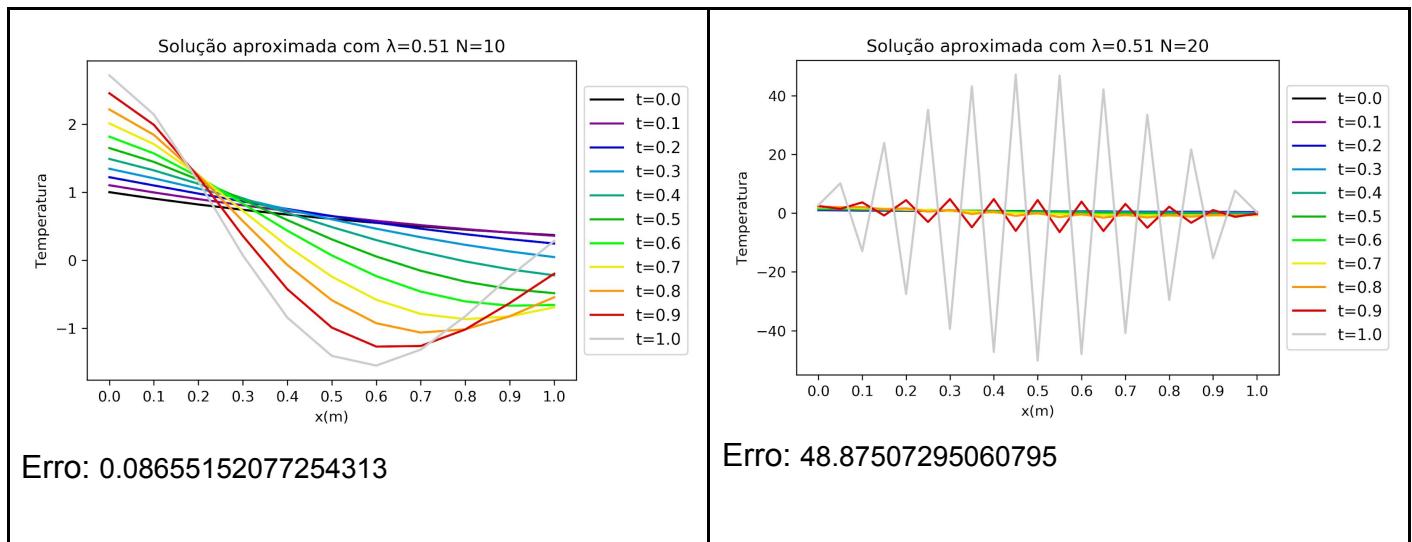


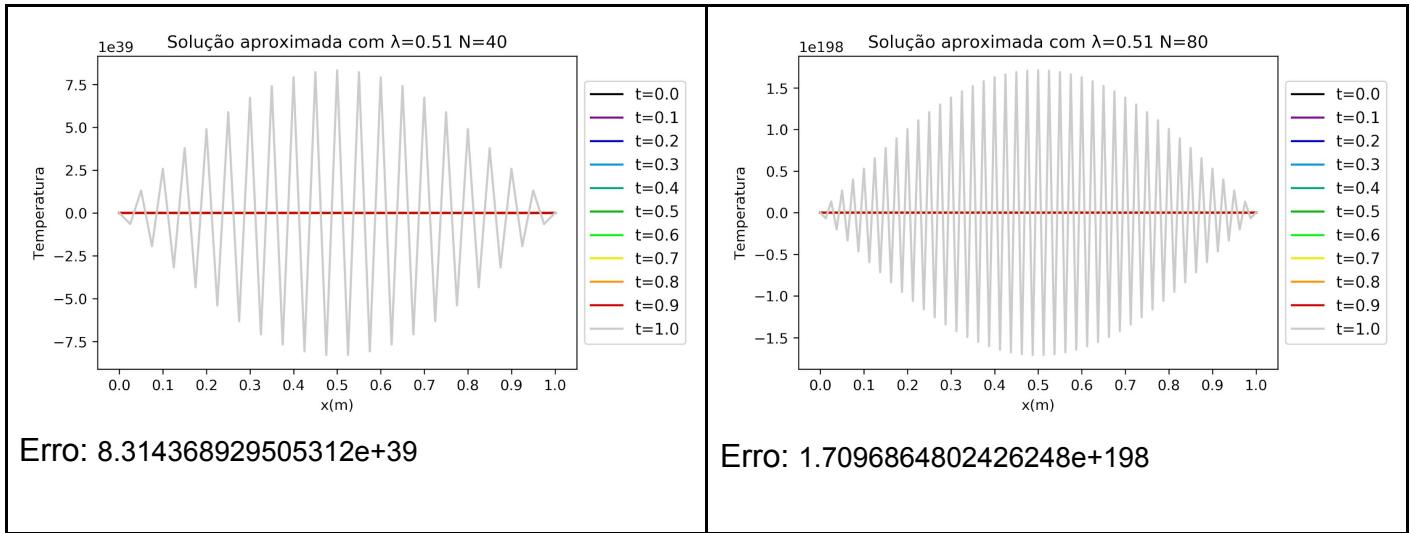


Fator de redução do erro a cada refinamento da malha: 3.99316975; 4.01069197; 3.99913062; 4.0006591 e 4.00009058.

Pode-se perceber que o valor se aproxima de 4, o que faz sentido, dado que o erro de truncamento é de ordem 1 em Δt e quando N dobrou, Δt é 4 vezes menor.

Ao realizar os mesmos testes para os diversos valores de N , porém com $\lambda = 0.51$, o Método Explícito fica instável. Assim sendo, o erro passa a crescer drasticamente, como apresentado pelos gráficos a seguir:





2.4. Item c

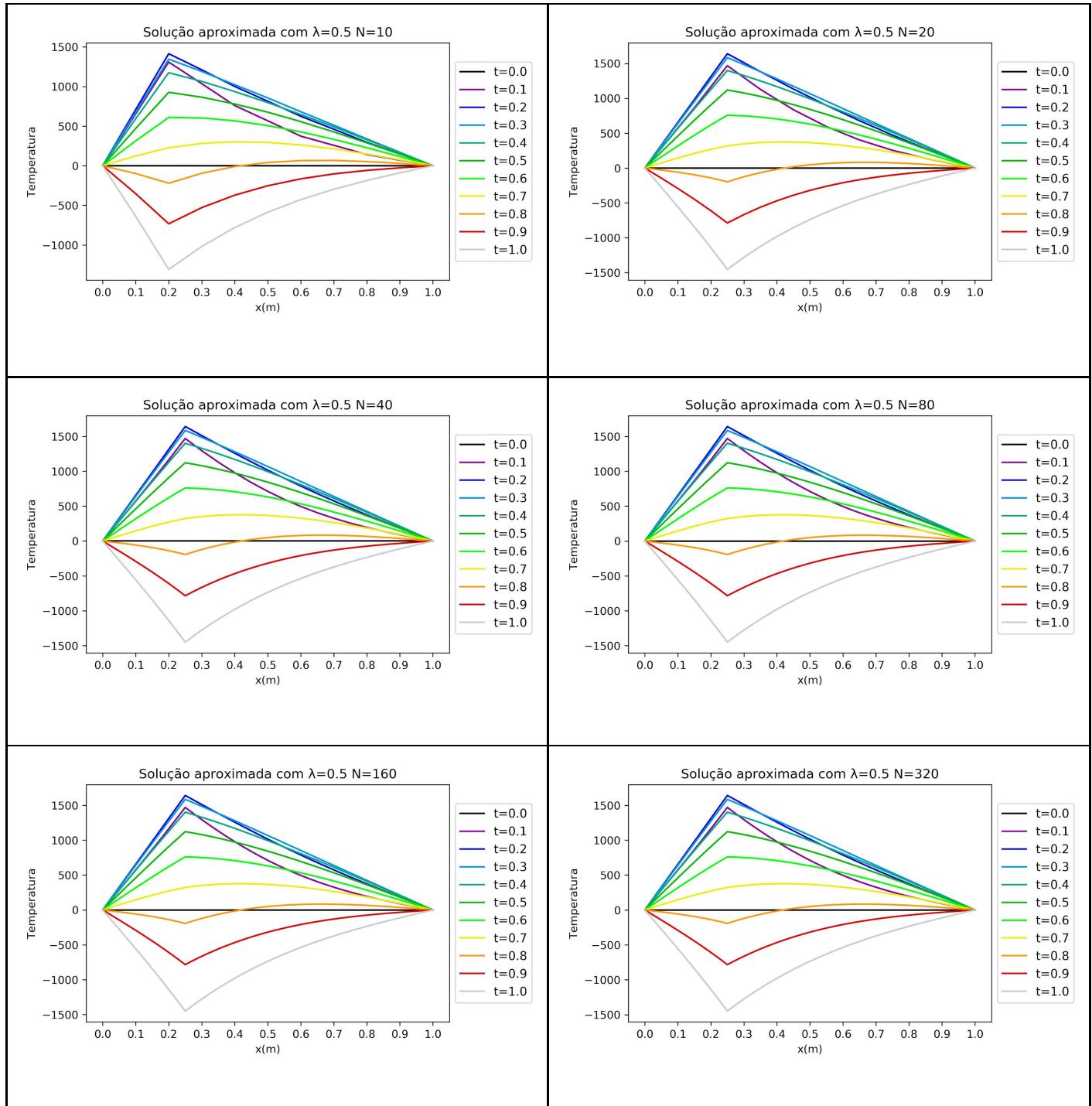
A fonte pontual considerada agora é definida como $f(t, x) = r(t)g_h(x)$. Nessa equação, a intensidade da fonte varia em função do tempo e é definida como $r(t) = 10000(1 - 2t^2)$ e $g_h(x)$ é definida como:

$$g_h(x) = \frac{1}{h}, \quad \text{se } p - h/2 \leq x \leq p + h/2 \quad \text{e} \quad g_h(x) = 0 \quad \text{caso contrario}$$

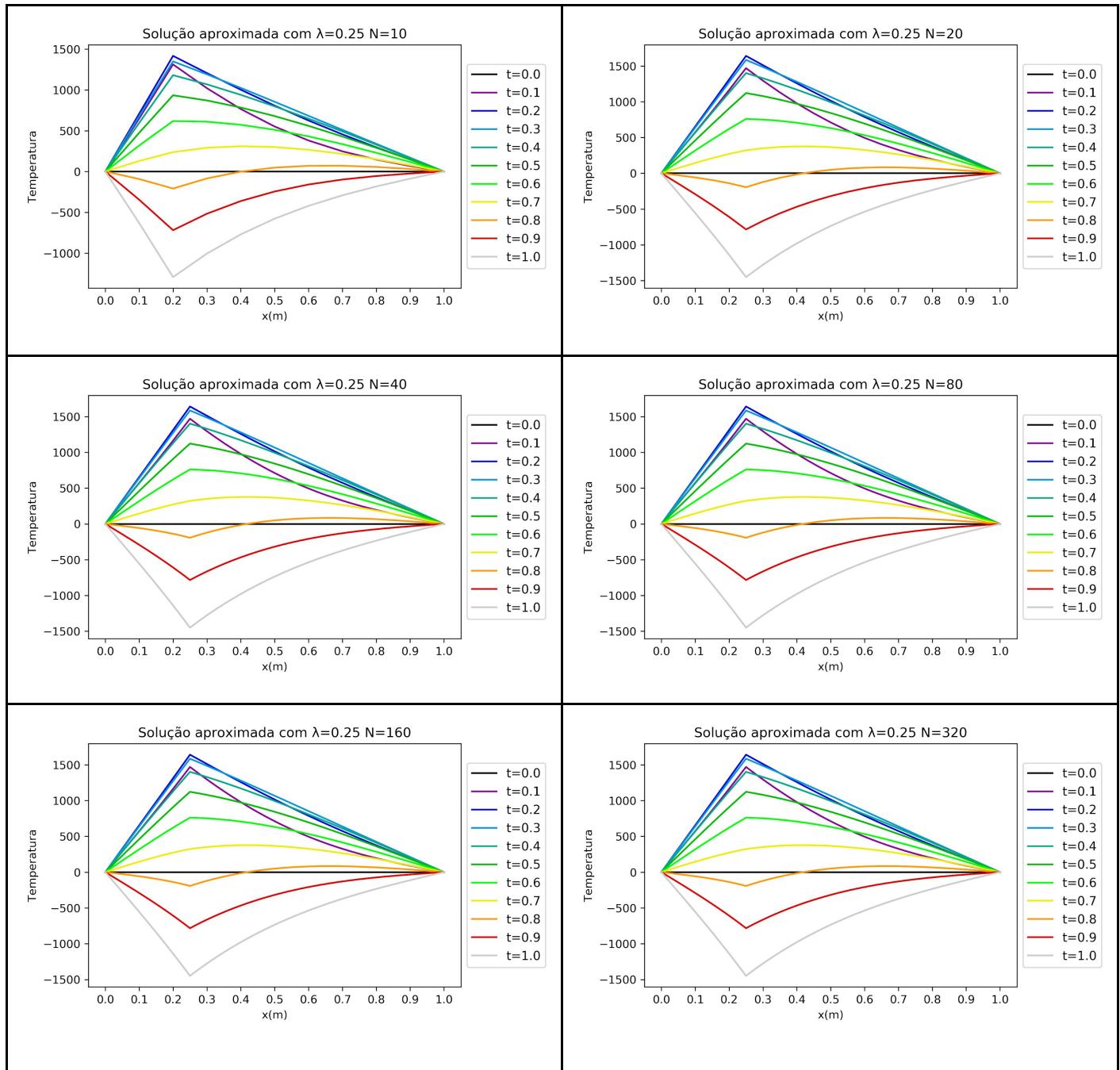
Trabalhando com a fonte em $p=0.25$, tal abordagem foi implementada no código descrito a seguir, que guarda o resultado da posição no vetor booleano p que, por sua vez, é utilizado para definir a fonte pontual.

```
elif teste == 3:
    #condição inicial u0
    grafico=np.array([0]*(N+1))
    linha_ant=grafico
    #vetor booleano que localiza o ponto p
    p=np.logical_and(x_vetor >= 0.25-delta_x/2, x_vetor <= 0.25+delta_x/2)
    for i,t in enumerate(t_vetor):
        if t>0:
            x=linha_ant+lamb*(np.roll(linha_ant,1)-2*linha_ant+np.roll(linha_ant,-1))+delta_t/delta_x*p*10000*(1-2*t**2)
            #condição nas fronteiras
            x[0]=0
            x[-1]=0
            linha_ant=x
            #solução a cada 0.1s
            if posicoes[0][i]==1:
                grafico=np.vstack((grafico,x))
```

Para $\lambda = 0.5$:



Para $\lambda = 0.25$:

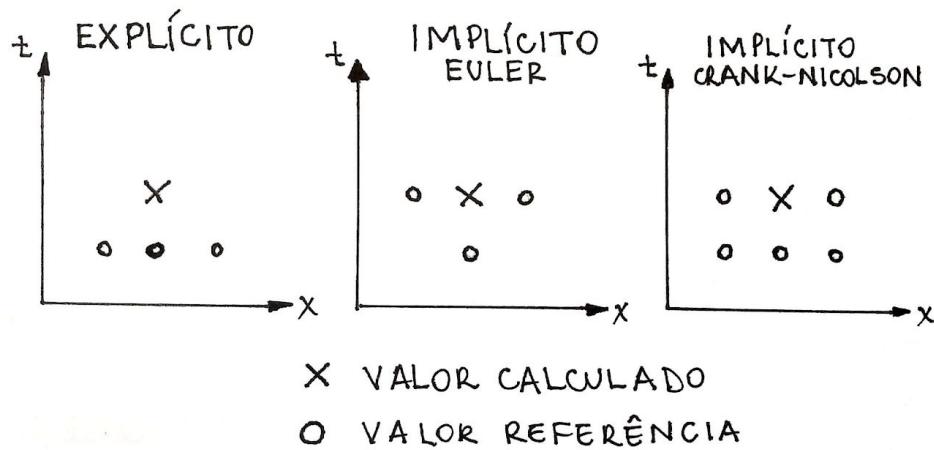


3. Segunda Tarefa

3.1. Introdução ao Método Implícito

Após realizar os testes com o método explícito, percebeu-se que este não era apropriado, uma vez que requer um número de passos demasiadamente grande. Assim, propõe-se o Método Implícito, mais eficiente nesse sentido.

No Método Explícito, como demonstrado no item 2.1, a solução da equação de calor em um ponto depende apenas de valores referentes a pontos da barra no instante anterior. Já nos métodos Implícito de Euler e Crank-Nicolson, a solução depende não só de valores referentes ao instante anterior, como também depende de valores referentes ao mesmo instante em que se está calculando a solução, assim como explicado pelo croqui abaixo. Dessa forma, nos métodos Implícitos de Euler e Crank-Nicolson, ocorre uma interdependência entre valores em um instante para definição de valores no mesmo instante. Essa interdependência torna necessário a resolução de um sistema de equações a cada novo instante de tempo.



3.2. Item a - Funções de Decomposição de Matriz e Resolução de Sistema

Tendo em vista que para o Método Implícito é preciso calcular diversos sistemas de equações do tipo $Ax = b$, é interessante desenvolver um programa capaz de eficientemente fazer a seguinte decomposição:

$$Ax = LDL^T x = b.$$

A decomposição acima é chamada de decomposição LDL^T e é uma variação de Fatoração de Cholesky. A Fatoração de Cholesky, por sua vez, é a

decomposição de uma matriz hermitiana (matriz igual a sua conjugada transposta) em uma matriz triangular inferior e sua matriz conjugada transposta e é escrita como $A = LL^*$. A Fatoração de Cholesky é utilizada com o intuito de obter soluções numéricas eficientes. A Decomposição LDL é uma variação da de Cholesky devido ao demonstrado a seguir:

$$A = LDL^* = LDD^*L^* = (LD)(LD)^*$$

A Decomposição LDL resulta em três matrizes: uma triangular (L), uma diagonal (D) e a conjugada transposta da primeira (L^*). Para casos de matrizes reais, como ocorre no problema tratado neste relatório, a matriz conjugada transposta é apenas uma matriz transposta (L^t). Assim a decomposição é escrita como: $A = LDL^t$.

O algoritmo para a decomposição $A = LDL^t$ para matrizes tridiagonais está explicitado a seguir:

$$A = LDL^t = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & l_{N-1} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & d_{N-1} \end{bmatrix} \begin{bmatrix} 1 & l_2 & 0 & 0 \\ 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & l_{N-1} \\ 0 & \dots & 0 & 1 \end{bmatrix} = \begin{bmatrix} d_1 & & & \\ l_2d_1 & l_2^2d_1 + d_2 & & \\ 0 & l_3d_2 & l_3^2d_2 + d_3 & \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \dots & 0 & l_{N-1}d_{N-1} & l_{N-1}^2d_{N-2} + d_{N-1} \end{bmatrix} \quad simetrico$$

Assim, generalizando, as entradas da matriz A podem ser definidas como dois vetores. O primeiro representando a diagonal principal (p) e o segundo representando cada uma das sub diagonais (s):

$$\begin{aligned} p &= (d_1, \quad l_2^2d_1 + d_2, \quad l_3^2d_2 + d_3, \dots, \quad l_{N-1}^2d_{N-2} + d_{N-1}) \\ s &= (l_2d_1, \quad l_3d_2, \dots, \quad l_{N-1}d_{N-2}) \end{aligned}$$

Dispondo dessa definição, as entradas do vetor d , que representa a matriz D , e do vetor l , que representa as matrizes L e L^t , podem ser calculadas da seguinte maneira:

$$\begin{cases} d_1 = p_1, \quad j = 1 \\ d_j = p_j - l_j^2 d_{j-1}, \quad j \geq 2 \\ l_j = \frac{1}{d_{j-1}} s_{j-1}, \quad j \geq 2 \end{cases}$$

A função desenvolvida para realização desta decomposição é:

```

def decompoem_matriç(diag_A,subdiag_A):
    """Calcula a decomposição LDLt de uma matriz tridiagonal simétrica A

    Parameters:
    diag_A (array): diagonal principal da matriz A
    subdiag_A (array): subdiagonal da matriz A

    Returns:
    L(array): matriz L da decomposição armazenada em 1 vetor
    D(array): matriz D da decomposição armazenada em 1 vetor
    """
    D=np.empty(len(diag_A))
    L=np.empty(len(subdiag_A))

    for i in range(len(diag_A)):
        if i == 0:
            D[i]=diag_A[i]
            L[i]=subdiag_A[i]/D[i]
        elif i>0 and i < len(diag_A)-1:
            D[i]=diag_A[i]-L[i-1]**2*D[i-1]
            L[i]=subdiag_A[i]/D[i]
        else:
            D[i]=diag_A[i]-L[i-1]**2*D[i-1]

    return L,D

```

Além da decomposição de matriz, para desenvolvimento do estudo do método implícito, também é necessário desenvolver uma forma eficiente de resolver sistemas que ja estejam decompostos na forma $(LDL^t)x = b$. O seguinte código tem o propósito de suprir tal necessidade:

```

def resolve_sistema(L,D,b):
    """Calcula a solução de um sistema linear do tipo LDLt*x=b

    Parameters:
    L(array): matriz L da decomposição armazenada em 1 vetor
    D(array): matriz D da decomposição armazenada em 1 vetor
    b(array): matriz do lado direito do sistema

    Returns:
    x(array): vetor contendo a solução do sistema
    """
    z=np.empty(len(b))
    for i in range(len(z)):
        if i == 0:
            z[0]=b[0]
        else:
            z[i]=b[i]-L[i-1]*z[i-1]

    y=z/D
    x=np.empty(len(b))

    for i in range(len(b)):
        if i == 0:
            x[-1]=y[-1]
        else:
            x[-1-i]=y[-1-i]-L[-i]*x[-i]

    return x

```

Este código foi estruturado baseado em um algoritmo que faz com que, partindo do que conhecemos, possamos destrinchar o sistema aos poucos com o intuito de buscar o que realmente desejamos, no caso x em $Ax = b$. Para tal, partiremos da decomposição LDL^t , ou seja, $LDL^tx = b$.

$$LDL^tx = b$$

$$DL^tx = z$$

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & l_{N-1} & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{bmatrix}$$

$$\left\{ \begin{array}{l} z_1 = b_1 \\ z_2 = b_2 - l_2 z_1 \\ z_3 = b_3 - l_3 z_2 \\ \vdots \\ z_{N-1} = b_{N-1} - l_{N-1} z_{N-2} \end{array} \right.$$

Assim, definimos e conhecemos o z em $DL^tx = z$. A partir disso podemos definir y com $L^tx = y$ como demonstrado a seguir:

$$DL^tx = z$$

$$L^tx = y$$

$$\begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & d_{N-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{N-1} \end{bmatrix}$$

$$\left\{ \begin{array}{l} y_1 = \frac{z_1}{d_1} \\ y_2 = \frac{z_2}{d_2} \\ y_3 = \frac{z_3}{d_3} \\ \vdots \\ y_{N-1} = \frac{z_{N-1}}{d_{N-1}} \end{array} \right.$$

De forma análoga a anterior, agora conhecendo y , podemos definir x em $L^tx = y$:

$$L^t x = y$$

$$\begin{bmatrix} 1 & l_2 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & l_{N-1} \\ 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix}$$

$$\left\{ \begin{array}{l} x_1 = y_1 - l_2 x_2 \\ x_2 = y_2 - l_3 x_3 \\ x_3 = y_3 - l_4 x_4 \\ \vdots \\ x_{N-1} = y_{N-1} \end{array} \right.$$

Importante ressaltar que esse último sistema deve ser resolvido a partir do x da posição final, ou seja começando por $x_{N-1} = y_{N-1}$. Isso porque, para o cálculo de determinado x , é necessário o valor do x seguinte.

3.3. Item b - Método de Euler Implícito

O primeiro método implícito que abordaremos será o de Euler. Nesse método, a distribuição de temperatura em uma barra em função da posição na barra e em função do tempo é dada por:

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

Repetiremos os testes realizados na Primeira Tarefa, aplicando o Método de Euler Implícito para as três diferentes equações como nos itens (2.2), (2.3) e (2.4). Mais uma vez realizaremos os testes para cada uma das equações para $N = 10, 20, 40, 80, 160$ e 320 , mas, agora, utilizaremos $\lambda = N$.

Para realizar os diferentes testes para Método de Euler Implícito descritos nos próximos itens, a seguinte função foi desenvolvida no programa:

```
def euler(N,M,teste):
    """Fornece a solução aproximada usando o método de Euler implícito

    Parameters:
    N (int): número de passos em x
    M (int): número de passos em t
    teste (int): qual a equação que deverá ser resolvida (conforme está no LEIAME.txt)

    Returns:
    grafico(array): matriz (M+1)x(N+1) com o valor da solução aproximada a cada ponto
    """

```

Verificamos que o Método de Euler Implícito é incondicionalmente estável e convergente, uma vez que é possível determinar o erro sem colocar restrições para valores de Δt e Δx , assim como demonstrado em seguida:

$$\begin{aligned} e_i^{k+1} &= e_i^k + \lambda(e_{i-1}^{k+1} - 2e_i^{k+1} + e_{i+1}^{k+1}) + \Delta t \tau_i^{k+1} \\ (1 + 2\lambda) |e_i^{k+1}| &\leq |e_i^k| + \lambda(|e_{i-1}^{k+1}| + |e_{i+1}^{k+1}|) + \Delta t |t_i^{k+1}| \\ \max_i : (1 + 2\lambda) \|e^{k+1}\| &\leq \|e^k\| + 2\lambda \|e^{k+1}\| + \Delta t \tau(\Delta t, \Delta x) \\ \|e^{k+1}\| &\leq \|e^k\| + \Delta t \tau(\Delta t, \Delta x) \end{aligned}$$

A seguinte fórmula foi utilizada para verificar a ordem de convergência (p) em cada teste:

$$p \approx \frac{\log (e_{\text{novo}}/e_{\text{velho}})}{\log (\Delta x_{\text{novo}}/\Delta x_{\text{velho}})}$$

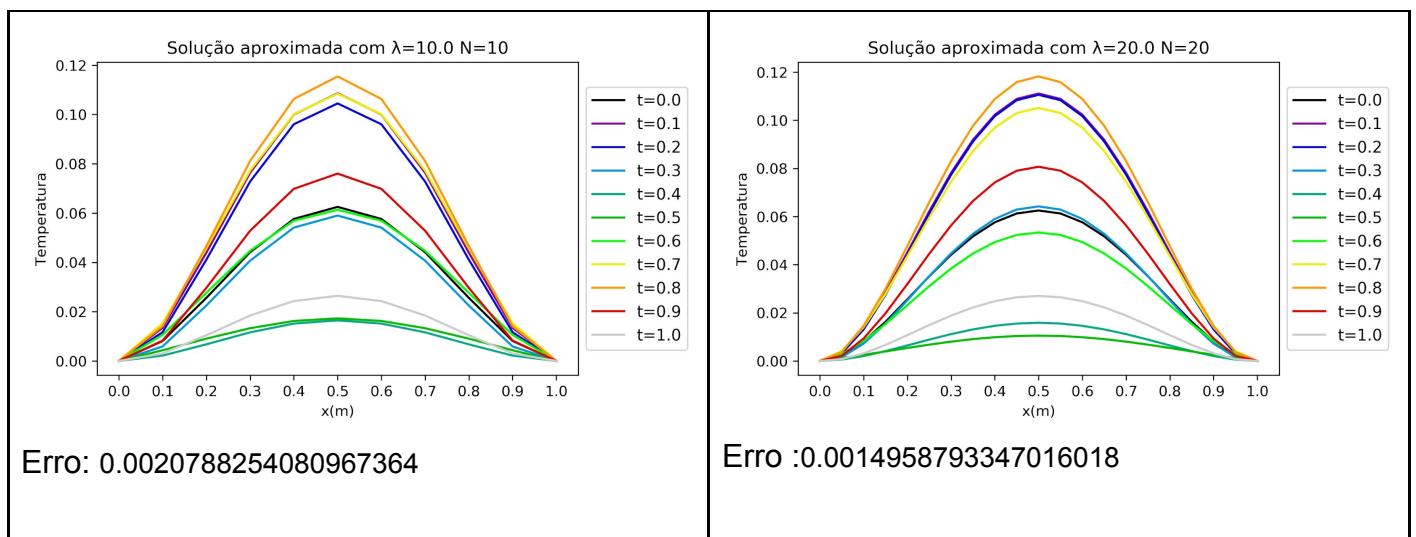
Com os testes referentes à equação 1 foram obtidos os seguintes resultados para p a cada refinamento da malha: 0.47477479; 0.76784182; 0.89111172; 0.94732166 e 0.97410106.

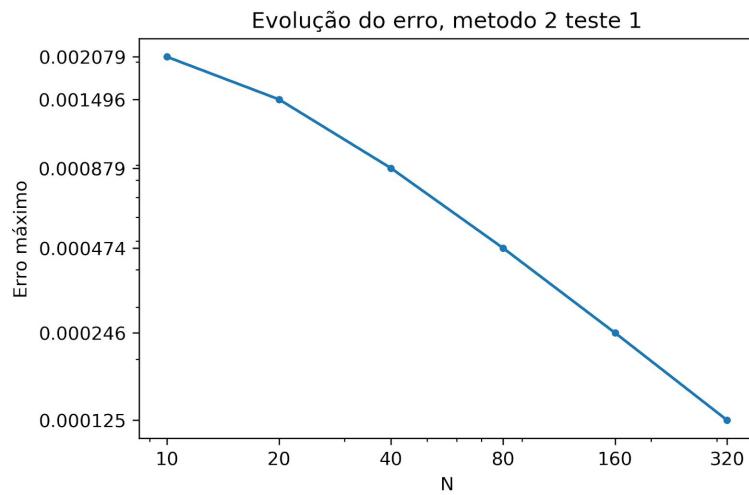
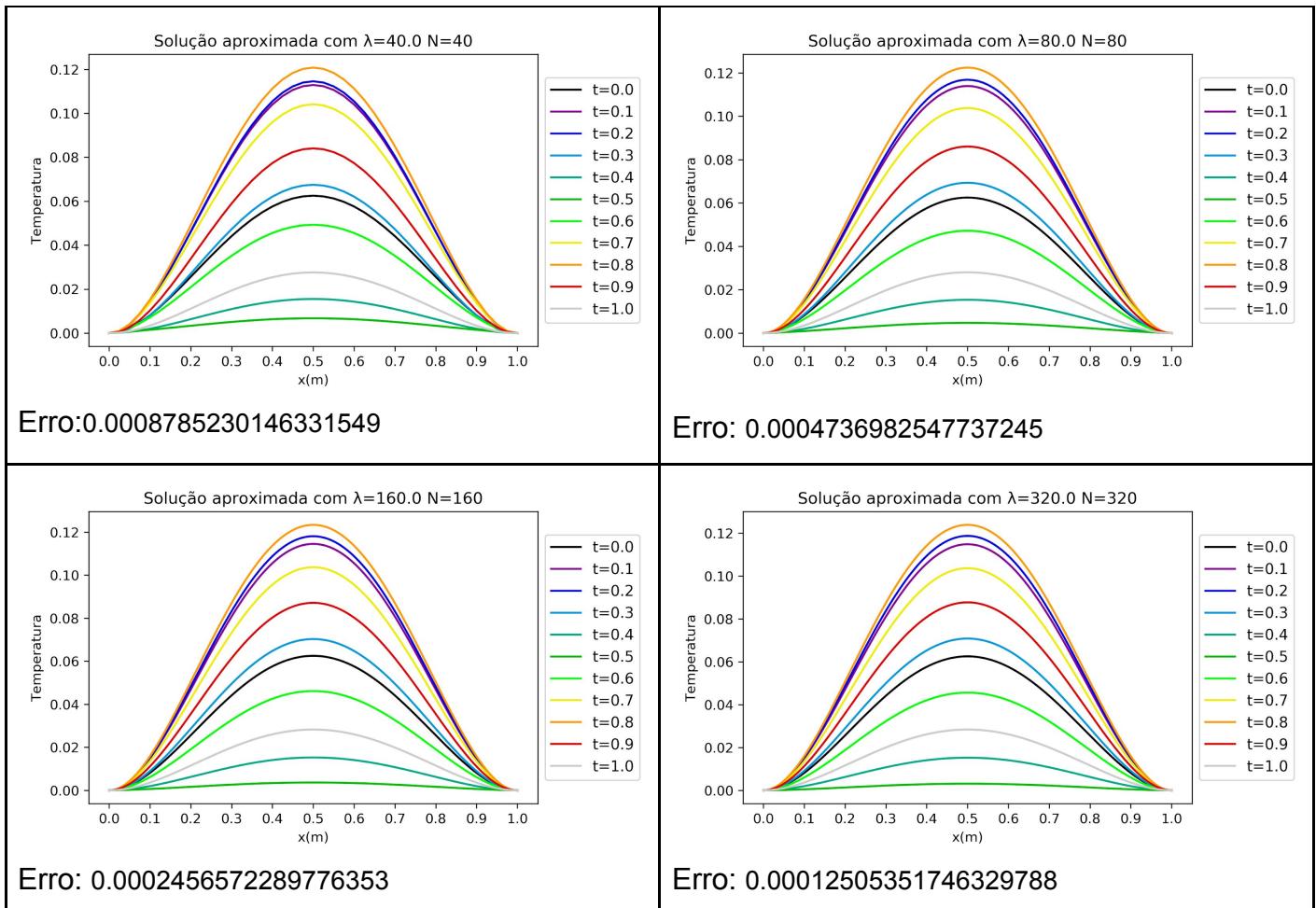
Com os testes referentes à equação 2 foram obtidos os seguintes resultados para p a cada refinamento da malha: 2.2625453; 0.5785364; 0.7980071; 0.90370288 e 0.95271325.

Percebe-se que os valores estimados cada vez mais se aproximam de 1, o valor da ordem de convergência em Δt que restringe a redução do erro.

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

Utilizando $\Delta t = \Delta x$:





O cálculo do erro é realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste. O valor retornado é o máximo calculado em cada teste. O cálculo utilizado para os testes deste item é:

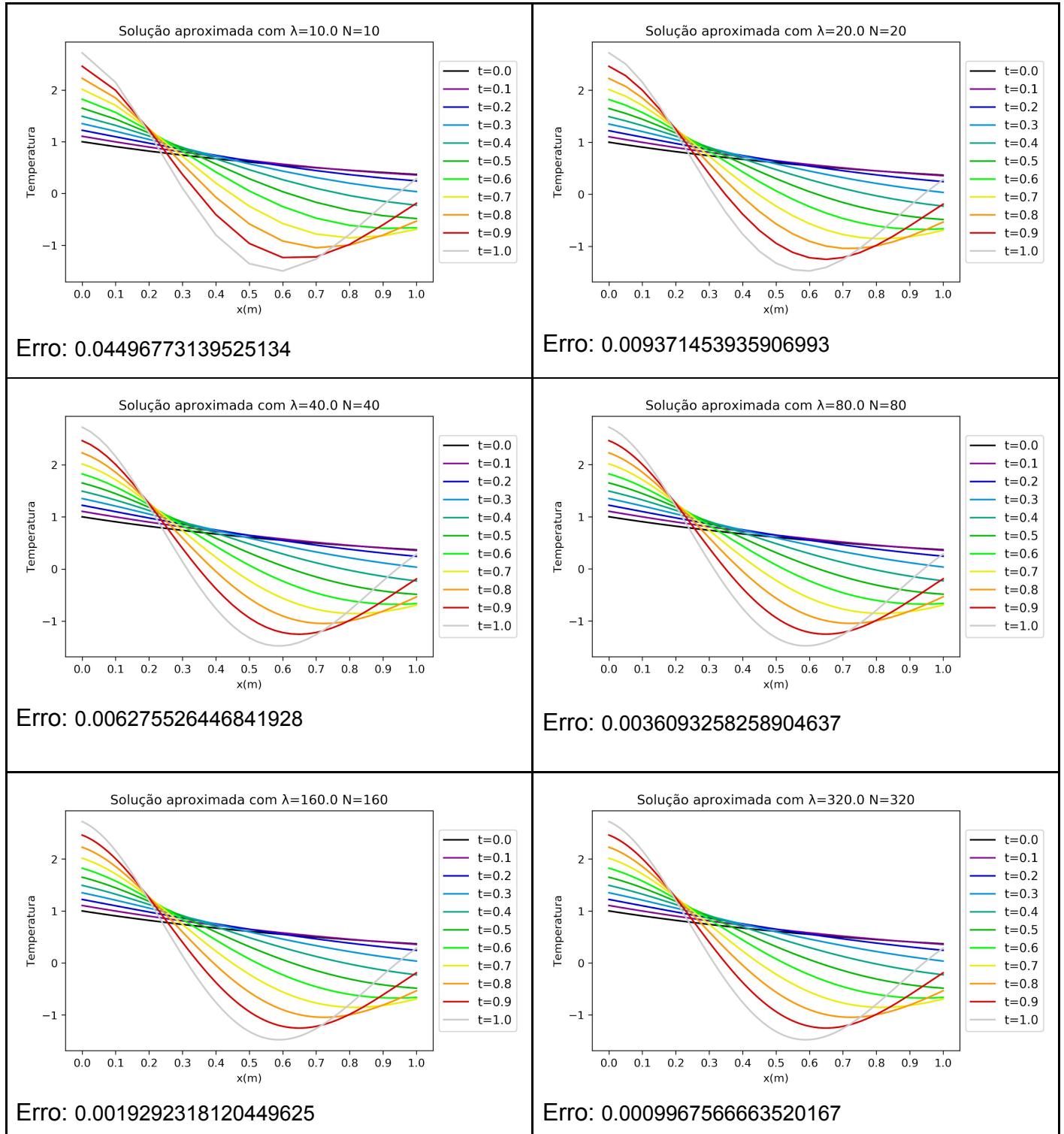
$$e_i^k = |(1 + \sin(10))x^2(1 - x)^2 - u_i^k|$$

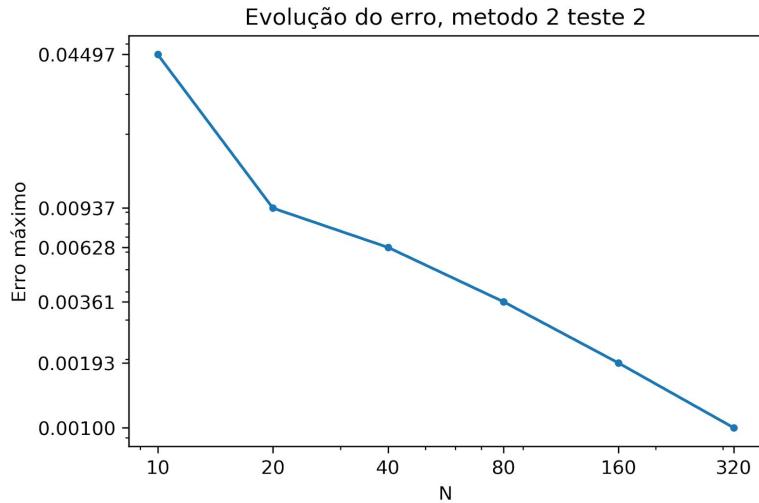
Fator de redução do erro a cada refinamento da malha: 1.38970127; 1.70272071; 1.85460471; 1.9282895 e 1.96441679.

Pode-se perceber que o valor se aproxima de 2, o que faz sentido, dado que o erro de truncamento é de ordem 1 em Δt e quando N dobra, Δt é 2 vezes menor.

3.3.2. Equação 2: $u(t, x) = e^{t-x} \cos(5tx)$

Utilizando $\Delta t = \Delta x$:





O cálculo do erro é realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste. O valor retornado é o máximo calculado em cada teste. O cálculo utilizado para os testes deste item é:

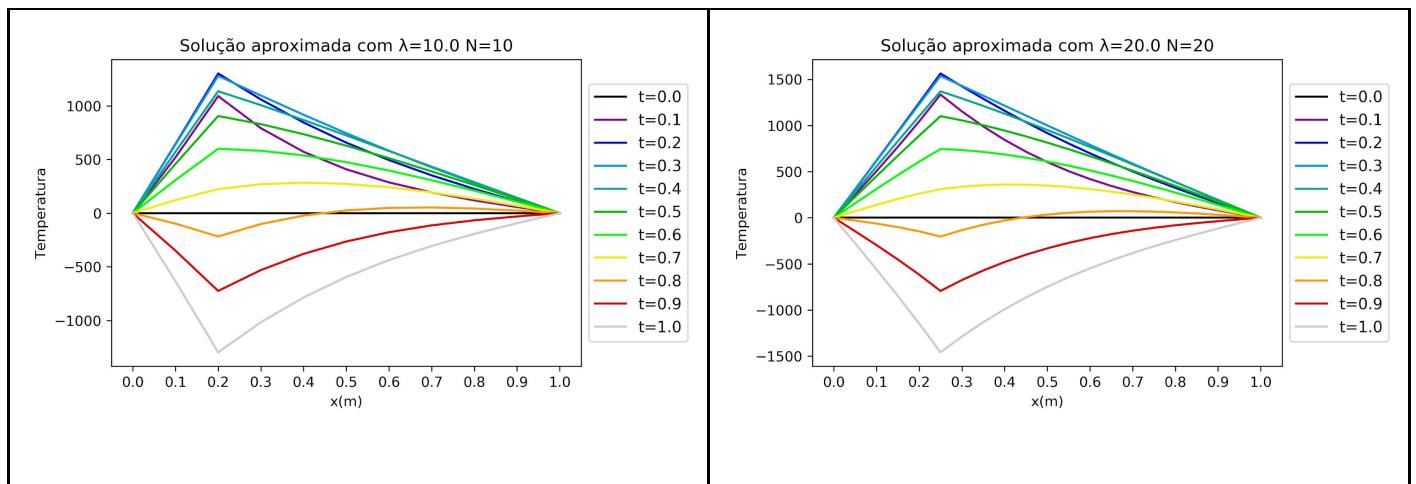
$$e_i^k = |e^{1-x} \cos(5x) - u_i^k|$$

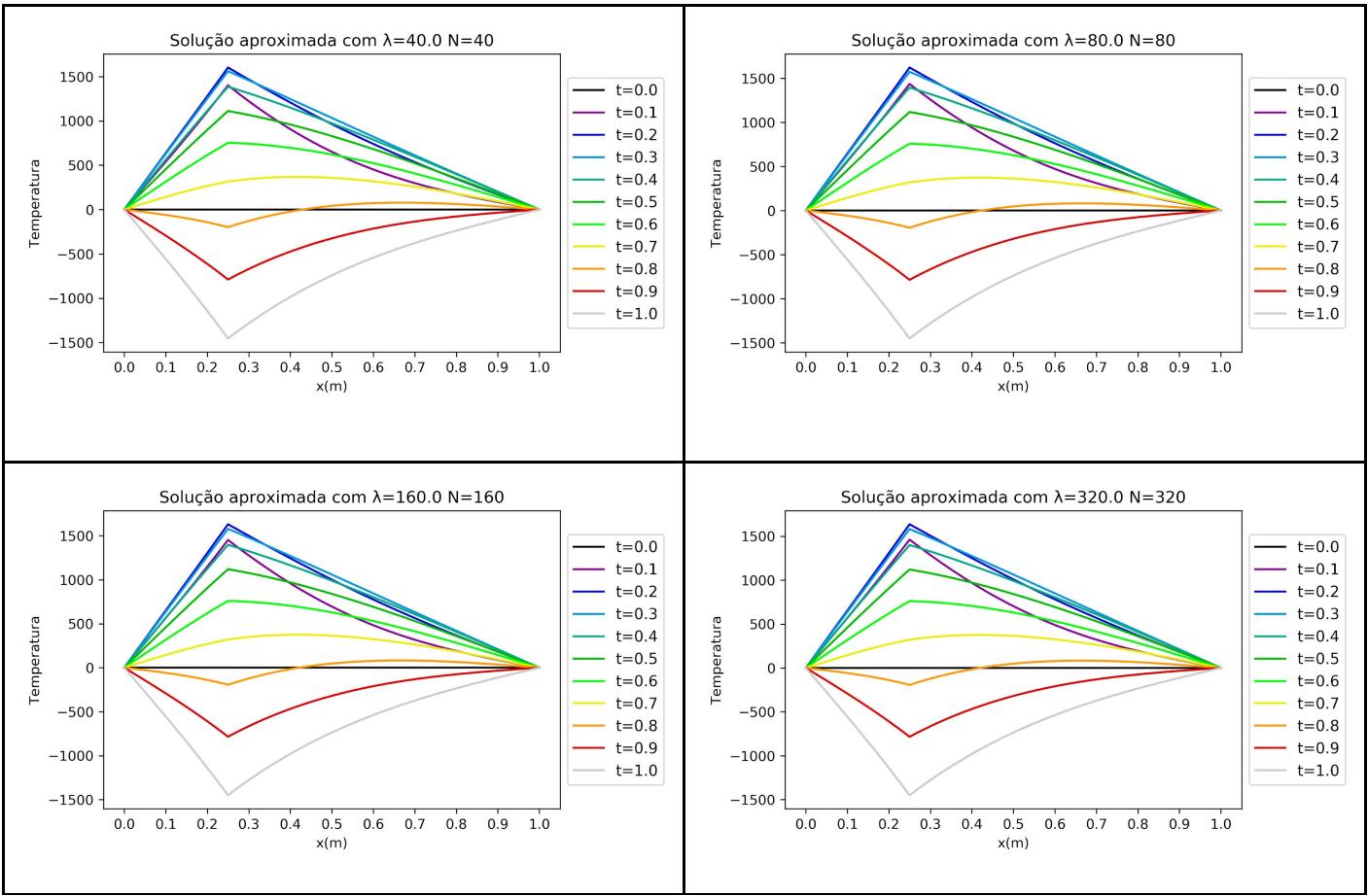
Fator de redução do erro a cada refinamento da malha: 4.79837298; 1.49333351; 1.73869768; 1.87086166 e 1.93550931.

Pode-se perceber que o valor se aproxima de 2, o que faz sentido, dado que o erro de truncamento é de ordem 1 em Δt e quando N dobrou, Δt é 2 vezes menor.

3.3.3. Equação 3:

Utilizando $\Delta t = \Delta x$:





3.4. Item c - Método de Crank-Nicolson

Método de Crank-Nicolson é o segundo método implícito que abordaremos, nele a distribuição de temperatura em uma barra em função da posição da barra e em função do tempo é dada por:

$$u_i^{k+1} = u_i^k + \frac{\lambda}{2}((u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) + (u_{i-1}^k - 2u_i^k + u_{i+1}^k)) + \frac{\Delta t}{2}(f(x_i, t_k) + f(x_i, t_{k+1})), \quad i = 1, \dots, N-1 \quad \text{e} \quad k = 0, \dots, M-1$$

Para a resolução dos sucessivos sistemas lineares, partindo da equação acima, foi possível determinar b como:

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

Analogamente ao processo realizado para o Método de Euler Implícito, realizaremos os testes da Primeira Tarefa, aplicando o Método Crank-Nicolson para as três diferentes equações como nos itens (2.2), (2.3) e (2.4). Serão realizados os testes para $N = 10, 20, 40, 80, 160$ e 320 , mas, agora, utilizaremos $\lambda = N$.

A função do programa utilizada para realizar os diferentes testes para o Método Crank-Nicolson descritos nos próximos itens é a destacada a seguir:

```
def crank(N,M,teste):
    """Fornece a solução aproximada usando o método de Crank-Nicolson

    Parameters:
    N (int): número de passos em x
    M (int): número de passos em t
    teste (int): qual a equação que deverá ser resolvida (conforme está no LEIAME.txt)

    Returns:
    grafico(array): matriz (M+1)x(N+1) com o valor da solução aproximada a cada ponto
    """

```

Um ponto interessante a ser analisado no Método Crank-Nicolson, e a razão pelo qual ele é utilizado, é o seu erro de truncamento. O processo de sua análise está descrito a seguir, note que as equações 5, 7 e 8 se referem às equações do enunciado do Exercício Programa:

$$\tau(\Delta t, \Delta x) = \frac{u_i^{k+1} - u_i^k}{\Delta t} - \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{2\Delta x^2} - \frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{2\Delta x^2} - \frac{(f_i^{k+1} + f_i^k)}{2}$$

Eq. 7: $u_t(t_{k+\frac{1}{2}}, x_i) = \frac{u(t_{k+1}, x_i) - u(t_k, x_i)}{\Delta t} + O(\Delta t^2)$

Eq. 8: $u_{xx}(t_k, x_i) = \frac{u(t_k, x_{i-1}) - 2u(t_k, x_i) + u(t_k, x_{i+1})}{\Delta x^2} + O(\Delta x^2)$

Eq. 8: $u_{xx}(t_{k+1}, x_i) = \frac{u(t_{k+1}, x_{i-1}) - 2u(t_{k+1}, x_i) + u(t_{k+1}, x_{i+1})}{\Delta x^2} + O(\Delta x^2)$

Eq. 5: $u_{xx}(t_{k+\frac{1}{2}}, x_i) = \frac{u_{xx}(t_k, x_i) + u_{xx}(t_{k+1}, x_i)}{2} + O(\Delta t^2)$

Eq. 5: $f(t_{k+\frac{1}{2}}, x_i) = \frac{f(t_k, x_i) + f(t_{k+1}, x_i)}{2} + O(\Delta t^2)$

$$\tau(\Delta t, \Delta x) = u_t(t_{k+\frac{1}{2}}, x_i) - O(\Delta t^2) - u_{xx}(t_{k+\frac{1}{2}}, x_i) - O(\Delta x^2) - f(t_{k+\frac{1}{2}}, x_i)$$

$$\tau(\Delta t, \Delta x) = O(\Delta t^2) + O(\Delta x^2)$$

Dessa forma, ao centrar a discretização em $t_{k+\frac{1}{2}}$ conseguimos obter convergência de ordem 2 em Δt e Δx

A seguinte fórmula foi utilizada para verificar a ordem de convergência (p) em cada teste:

$$p \approx \frac{\log(e_{novo}/e_{velho})}{\log(\Delta x_{novo}/\Delta x_{velho})}$$

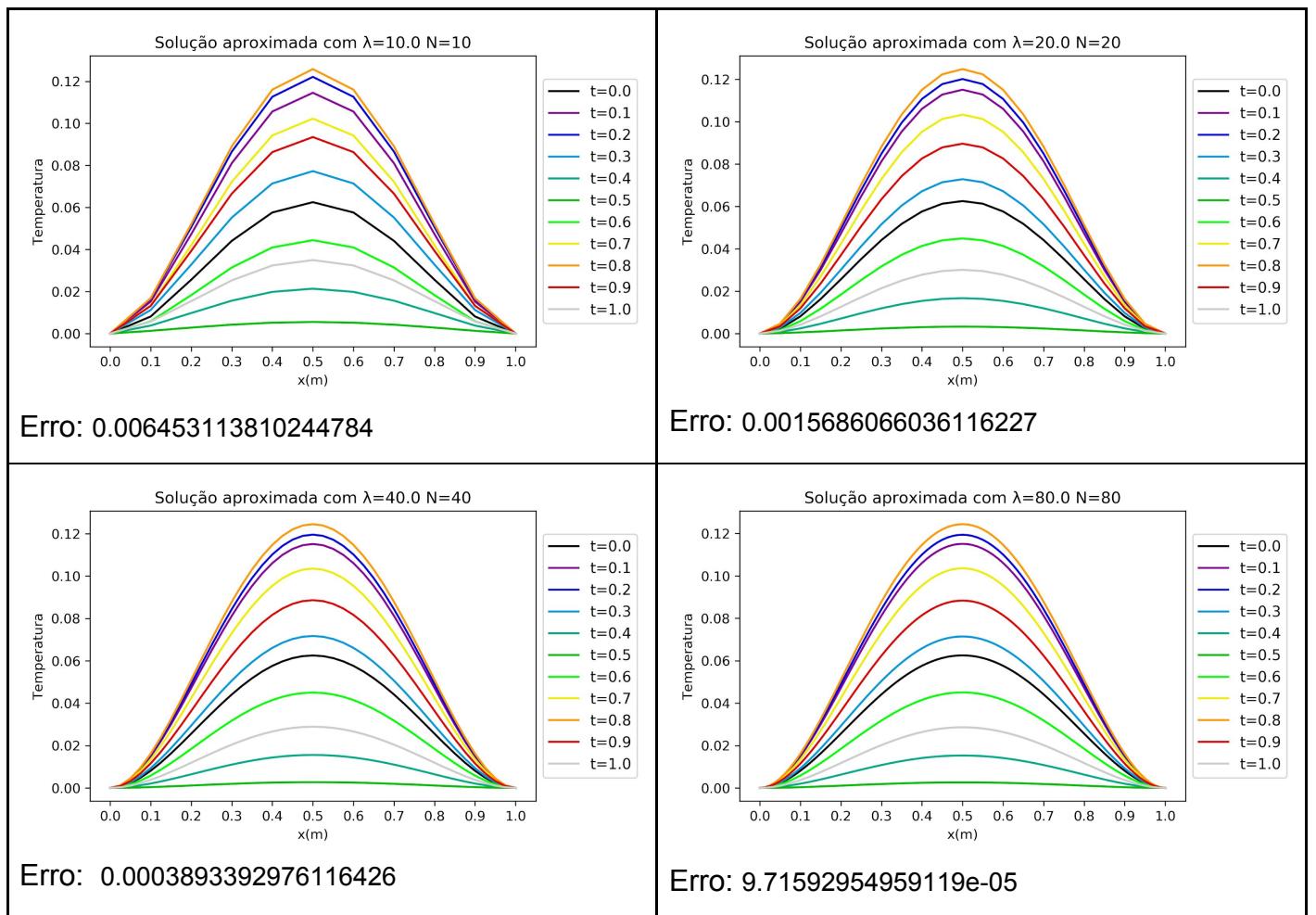
Com os testes referentes à equação 1 foram obtidos os seguintes resultados para p a cada refinamento da malha: 2.04051189; 2.0103837; 2.00260404; 2.00065157 e 2.00016293.

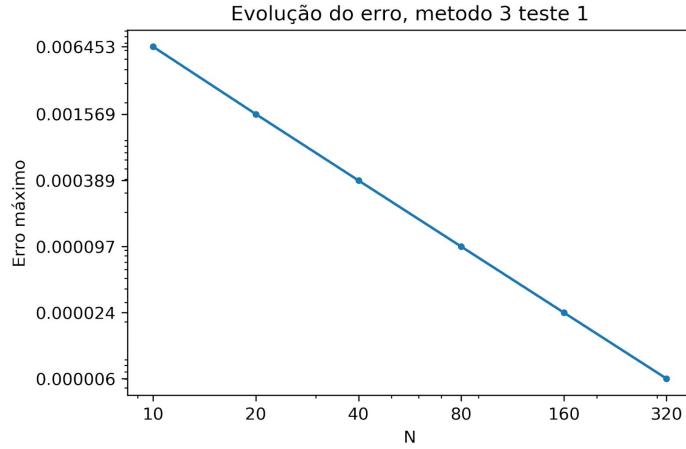
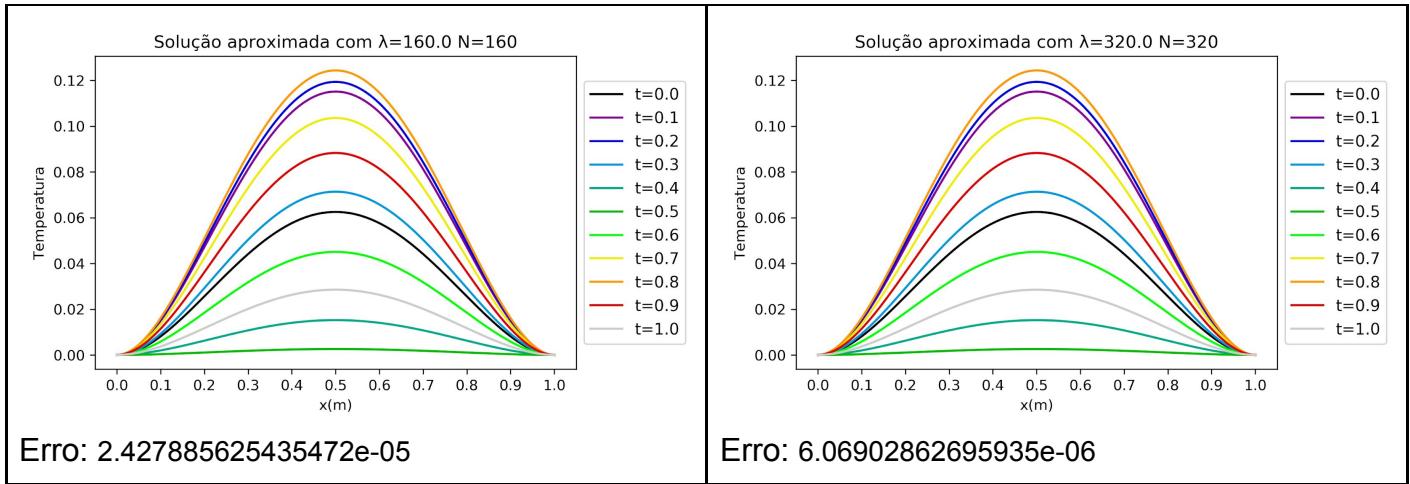
Com os testes referentes à equação 2 foram obtidos os seguintes resultados para p a cada refinamento da malha: 1.99455172; 2.00232834; 1.9988719; 2.00014024 e 2.00003506.

Percebe-se que os valores estimados cada vez mais se aproximam de 2, que é a ordem de convergência tanto de Δt , quanto de Δx .

3.4.1. Equação 1: $u(t,x) = (1 + \operatorname{sen}(10t))x^2(1-x)^2$

Utilizando $\Delta t = \Delta x$:





O cálculo do erro é realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste. O valor retornado é o máximo calculado em cada teste. O cálculo utilizado para os testes deste item é:

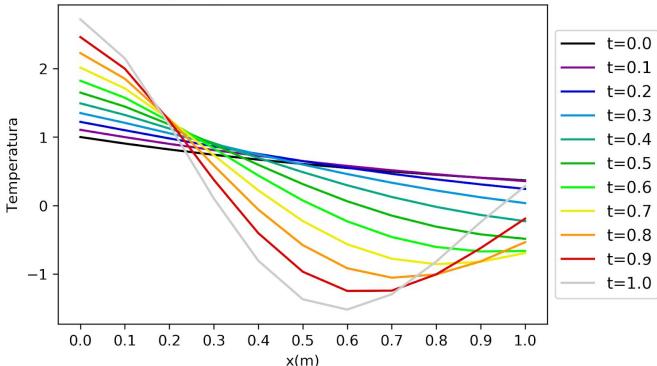
$$e_i^k = |(1 + \sin(10))x^2(1 - x)^2 - u_i^k|$$

Fator de redução do erro a cada refinamento da malha: 4.11391473; 4.0288936; 4.00722644; 4.00180694 e 4.00045176.

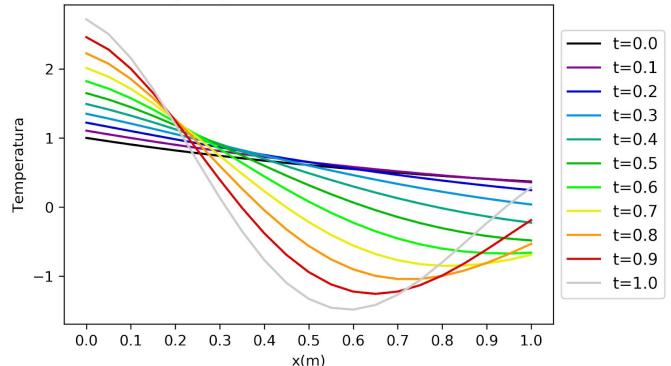
Pode-se perceber que o valor se aproxima de 4, o que faz sentido, dado que o erro de truncamento é de ordem 2 tanto em Δt quanto em Δx e ambos caem pela metade a cada teste feito.

3.4.2. Equação 2: $u(t, x) = e^{t-x} \cos(5tx)$

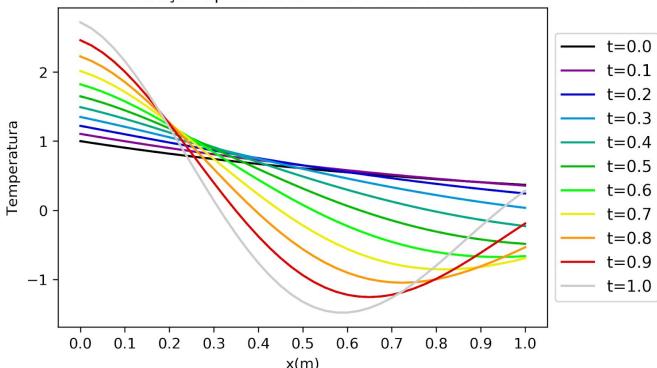
Utilizando $\Delta t = \Delta x$:

Solução aproximada com $\lambda=10.0$ N=10

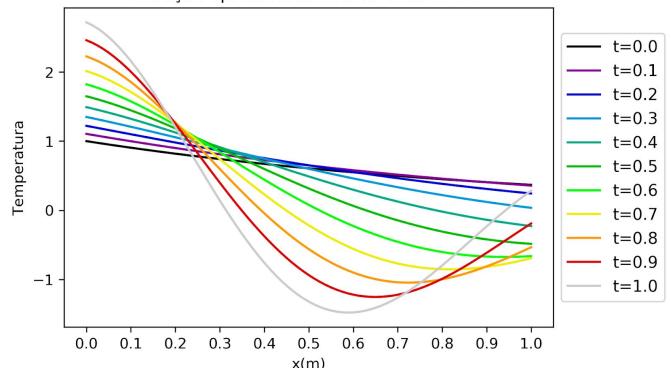
Erro: 0.047779438080028935

Solução aproximada com $\lambda=20.0$ N=20

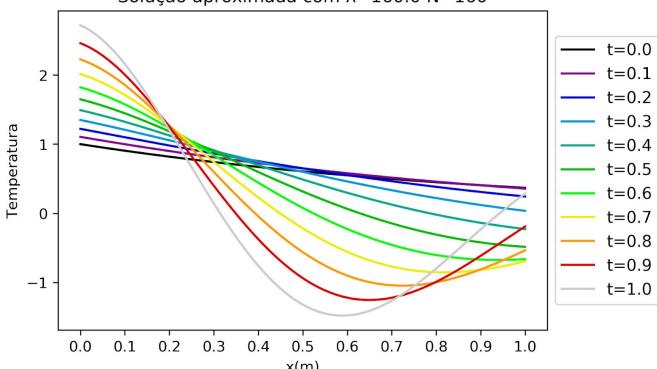
Erro: 0.01199005405793363

Solução aproximada com $\lambda=40.0$ N=40

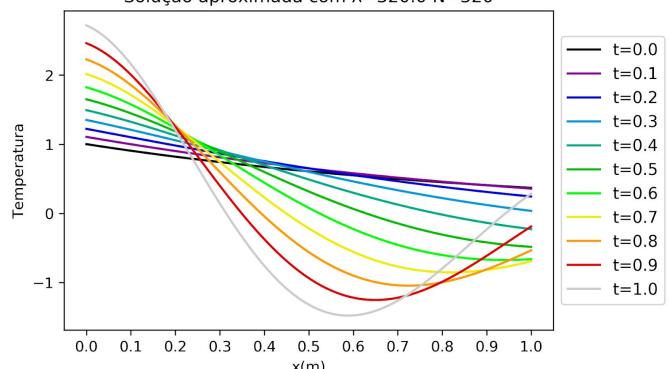
Erro: 0.00299267979185025

Solução aproximada com $\lambda=80.0$ N=80

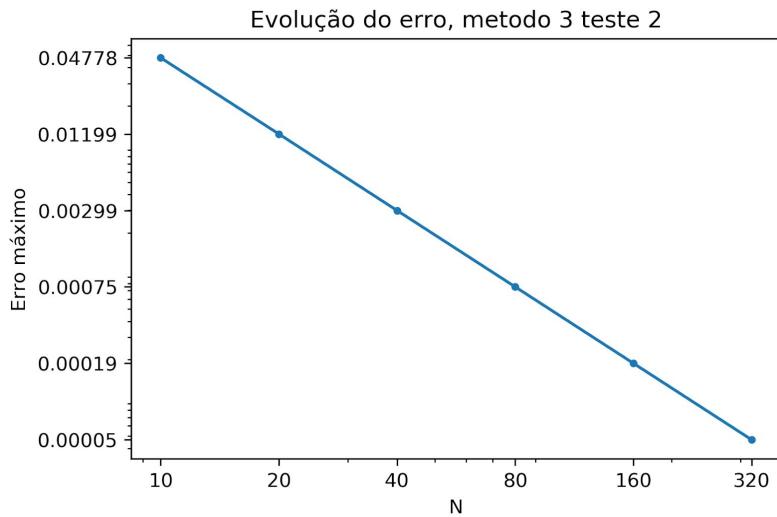
Erro: 0.0007487551993312636

Solução aproximada com $\lambda=160.0$ N=160

Erro: 0.0001871706044498378

Solução aproximada com $\lambda=320.0$ N=320

Erro: 4.679151400521775e-05



O cálculo do erro é realizado sempre em $t = 1$ e corresponde a diferença em módulo entre a solução exata e os valores obtidos no teste. O valor retornado é o máximo calculado em cada teste. O cálculo utilizado para os testes deste item é:

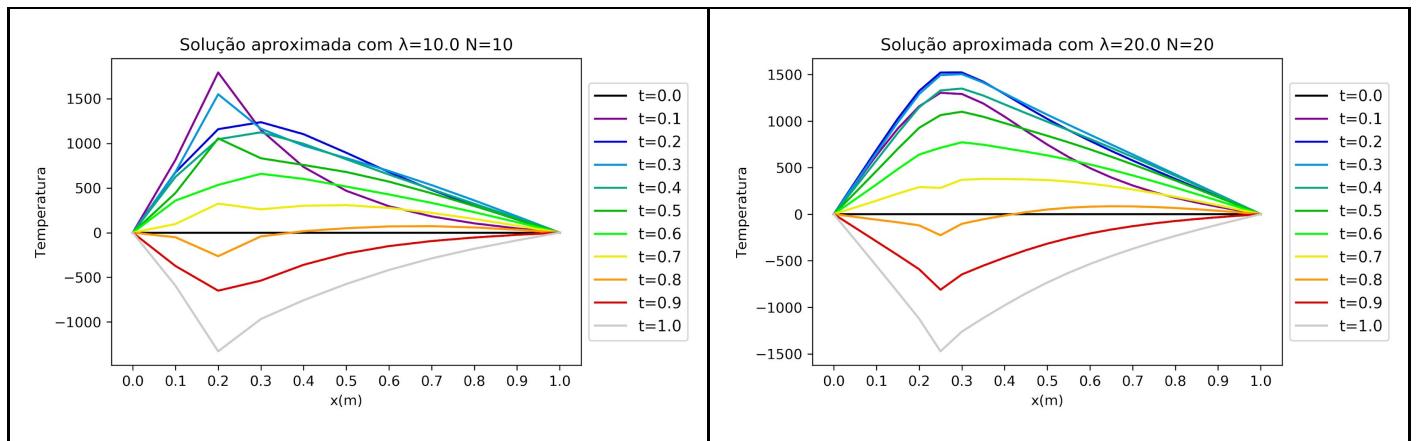
$$e_i^k = |e^{1-x} \cos(5x) - u_i^k|$$

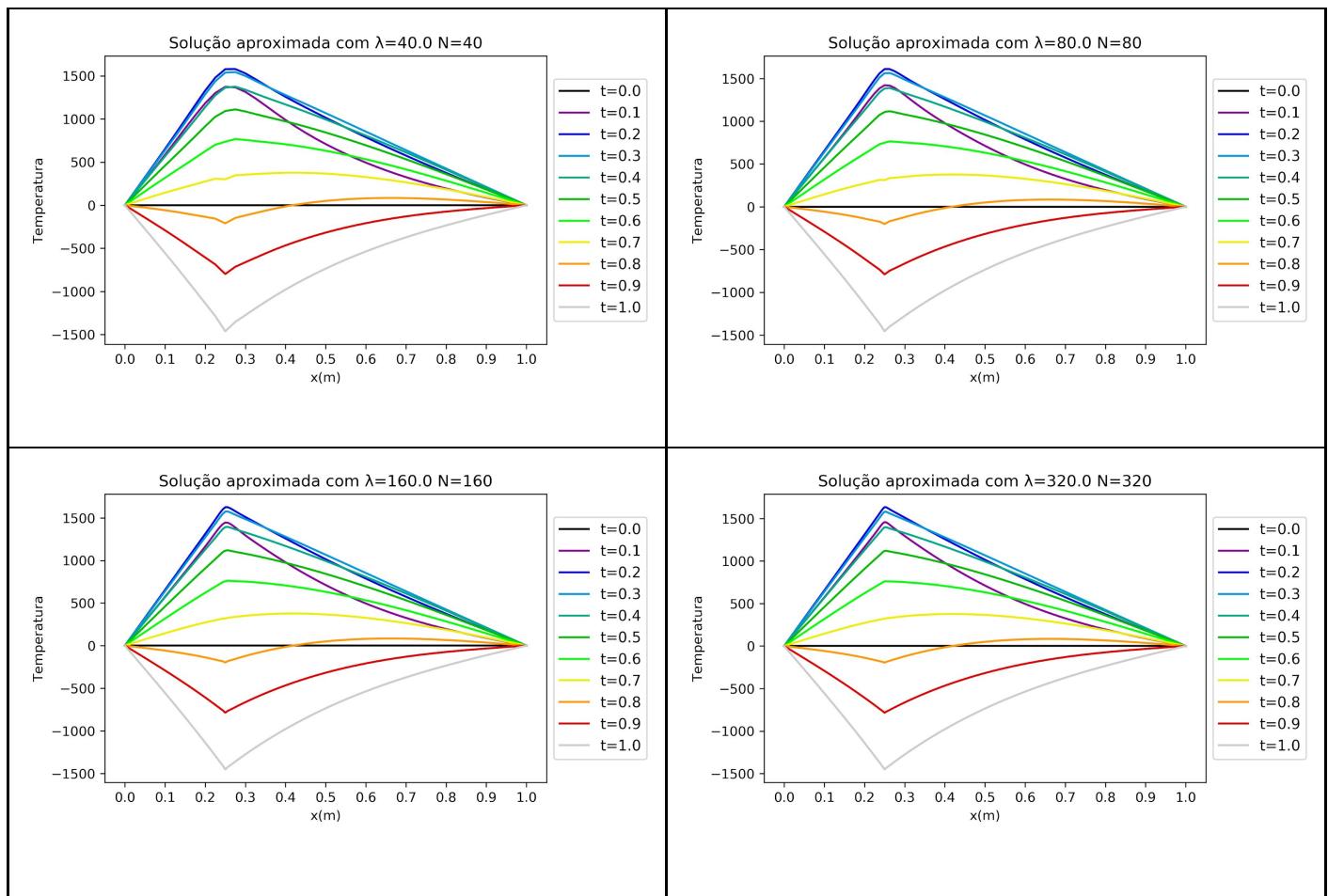
Fator de redução do erro a cada refinamento da malha: 3.98492266; 4.00646073; 3.99687347; 4.00038885 e 4.00009721.

Pode-se perceber que o valor se aproxima de 4, o que faz sentido, dado que o erro de truncamento é de ordem 2 tanto em Δt quanto em Δx e ambos caem pela metade a cada teste feito.

3.4.3. Equação 3

Utilizando $\Delta t = \Delta x$:





4. Considerações finais

Por meio deste Exercício Programa, conseguimos perceber o poder que as aproximações por diferenças finitas têm, permitindo resolver equações de funções desconhecidas tendo em mãos somente as condições de contorno. A partir do desenvolvimento da solução em Python, percebemos um resultado interessante: o método explícito, apesar de permitir que os valores de temperatura para um certo t_k sejam calculados diretamente, tem um tempo de execução muito maior que os métodos implícitos (Euler e Crank-Nicolson), que necessitam resolver um sistema linear a cada t_k , uma operação numericamente intensiva.

Outro aspecto positivo dos métodos implícitos em relação ao explícito, é o fato de serem absolutamente convergentes. Isso os difere do método explícito em que a convergência está restrita a casos em que $\lambda \leq 0.5$.

Também foi interessante perceber como a norma do erro tem um comportamento bem semelhante ao que é previsto pela teoria, com os valores da ordem de convergência e taxa de redução do erro se aproximando mais dos valores teóricos a cada refinamento da malha.

5. Referências Bibliográficas

BURDEN, R. & FAIRES, D. Análise Numérica, Cengage Learning, 7th ed., 2008.