Universidade de São Paulo

Escola Politécnica

MAP3121 - Métodos Numéricos e Aplicações (2020)

Exercício Programa 2

Nomes	NUSP
Elias Daleffi Rodrigues Rayes	10823848
Vinícius Marchioli	10774232

Sumário

1 Tarefa A

Na primeira tarefa, a partir do desenvolvido no Exercício Programa 1, será empregado o método de Crank-Nicolson, cujo termo geral é descrito pela relação mostrada em (1), para resolução numérica da equação diferencial de calor dada por (2), (3), (4) e (5) com condição inicial e de fronteira nulas $\big(u_0(x) = g_1(t) = g_2(t) = 0\big)$.

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

$$u_t(t,x) = u_{xx}(t,x) + f(t,x) \ \text{ em } \ [0,T] \times [0,1] \quad (2)$$

$$u(0,x) = u_0(x) \ \text{ em } \ [0,1] \quad (3)$$

$$u(t,0) = g_1(t) \ \text{ em } \ [0,T] \quad (4)$$

$$u(t,1) = g_2(t) \ \text{ em } \ [0,T] \quad (5)$$

Dado um conjunto de pontos $p_1, p_2, \cdots, p_{nf}$, deve-se gerar os vetores $u_k(T, x_i),\ i = 1, 2, \cdots, N-1$ que se referem às distribuições de temperatura na barra, no instante T $(t = 1)$, a partir da ação de cada fonte de calor pontual da forma $f(t,x) = r(t)g_h^k(x),\ k = 1, 2, \cdots, nf$, sendo $r(t) = 10 \cdot (1 + \cos(5t))$, com $g_h^k(x)$ e $f(t,x)$ nas formas (6) e (7).

$$g_h^k(x) = \begin{cases} \dfrac{1}{h}, & \text{se } \ p - \dfrac{h}{2} \leq x \leq p + \dfrac{h}{2} \\ 0, & \text{caso contrário.} \end{cases} \quad (6)$$

$$f(t,x) = \begin{cases} \dfrac{10 \cdot (1 + \cos(5t))}{\Delta x}, & \text{se } \ p - \dfrac{\Delta x}{2} \leq x \leq p + \dfrac{\Delta x}{2} \\ 0, & \text{caso contrário.} \end{cases} \quad (7)$$

É relevante ressaltar que, no escopo deste exercício, obter todas as distribuições de temperatura $u_k(T, x_i)$ individualmente, resultantes da ação de uma única fonte, corresponde a encontrar um conjunto de vetores linearmente independentes entre si que compõem uma base, cuja dimensão é o número total de fontes pontuais. O agregado de tais fontes corresponde à uma forçante da forma $f(t, x) = r(t) \sum_{k=1}^{nf} a_k g_h^k(x)$. Os coeficientes a_k representam as contribuições de cada fonte para a temperatura final e serão determinados ao fim do exercício programa.

Em relação ao que já foi implementado no exercício programa 1, foi necessário, simplesmente, ajustar a função $f(t, x)$ do método de Crank-Nicolson às condições do enunciado (considerar $r(t) = 10 \cdot (1 + \cos(5t))$) e encontrar a distribuição final de calor para cada p. Assim, para a solução desse passo, foi utilizada a decomposição $A = LDL^t$ para uma matriz A esparsa, tridiagonal e positivamente definida e, a partir disso, pode-se calcular, de modo mais eficiente, a solução para o sistema linear que descreve o termo geral (1) para cada instante de tempo. Ao se fazer esse procedimento para cada fonte e extrair das soluções o instante final, encontra-se a base desejada, sobre qual deve-se projetar $u_T(x)$.

2 Tarefa B

Quando conhecida a distribuição de temperatura $u_T = r(t) \sum_{k=1}^{nf} a_k u_k(T, x)$, no instante T, é de interesse determinar os coeficientes a_k que minimizam a expressão (8). Dessa forma, deve-se resolver um problema de melhor aproximação, ou seja, de mínimos quadrados. É importante ressaltar que os extremos do intervalo têm seus valores conhecidos pelas condições de contorno impostas e, por isso, não necessitam ser contemplados pela expressão (8).

$$E_2 = \sqrt{\Delta x \sum_{i=1}^{N-1} \left(u_T(x_i) - \sum_{k=1}^{nf} a_k u_k(T, x_i) \right)^2} \tag{8}$$

Em posse dos vetores que compõem a base, para a solução do problema de minimização de distâncias e obtenção dos coeficientes (intensidade de cada fonte), deve-se montar e resol-

ver o sistema representado em (9). Para tal, é necessário implementar a rotina que calcula o produto interno, mostrado em (10).

$$\begin{bmatrix} \langle u_1, u_1 \rangle & \langle u_2, u_1 \rangle & \cdots & \langle u_{nf}, u_1 \rangle \\ \langle u_1, u_2 \rangle & \langle u_2, u_2 \rangle & \cdots & \langle u_{nf}, u_2 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle u_1, u_{nf} \rangle & \langle u_2, u_{nf} \rangle & \cdots & \langle u_{nf}, u_{nf} \rangle \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nf} \end{bmatrix} = \begin{bmatrix} \langle u_T, u_1 \rangle \\ \langle u_T, u_2 \rangle \\ \vdots \\ \langle u_T, u_{nf} \rangle \end{bmatrix} \tag{9}$$

$$\langle u, v \rangle = \sum_{i=1}^{N-1} u(x_i) v(x_i) \tag{10}$$

3 Tarefa C

Ao contrário do exercício programa 1, cujo sistema a ser resolvido era constituído por uma matriz tridiagonal simétrica, as matrizes que compõem o sistema (9), não são esparsas. Por conseguinte, para a otimização do tempo e dos recursos computacionais empregados, é feita decomposição[2] LDL^t sendo L uma matriz triangular inferior e D uma matriz diagonal, como visto em (11).

$$L = \begin{bmatrix} 1 & & & & & \\ l_{2,1} & \ddots & & & & \\ & \ddots & 1 & & & \\ \vdots & \dots & l_{j+1,j} & 1 & & \\ & & \vdots & \ddots & \ddots & \\ l_{N,1} & \dots & l_{N,j} & \dots & l_{N,N-1} & 1 \end{bmatrix} \quad D = \begin{bmatrix} d_1 & 0 & & \dots & & 0 \\ 0 & d_2 & & & & \\ & & \ddots & & & \vdots \\ \vdots & & & d_j & & \\ & & & & \ddots & 0 \\ 0 & & \dots & & 0 & d_N \end{bmatrix} \tag{11}$$

$$D_j = A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2 D_k, \tag{12}$$

$$L_{ij} = \frac{1}{D_j} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} D_k \right) \quad \text{para } i > j. \tag{13}$$

A solução da equação matricial pode ser feita em três partes, resolvendo sistemas lineares triangulares simples em cada passo. Podemos, então, decompor $A = LDL^t$ em:

$$Lz = b$$

$$Dy = z$$

$$L^t x = y$$

4 Testes

Após a implementação das rotinas descritas nas tarefas, foram executados quatro diferentes testes para a comprovação do correto funcionamento do programa. A persistência das saída de cada teste foi feita através de arquivos *.txt* cujos nomes seguem a convenção Output + < letra do teste > + < valor de N >. Por exemplo, o arquivo nomeado *OutputC2048.txt* refere-se ao teste C, com $N = 2048$. Em todos os testes, foi considerado $r(t) = 10 \cdot (1 + \cos(5t))$ e $T = 1$.

O *script* de MATLAB utilizado para gerar as curvas encontra-se no Apêndice A.

4.1 Teste A

Primeiramente, considerou-se uma única fonte pontual ($nf = 1$), $N = 128$, $p = 0,35$ e definiu-se $u_T(x_i) = 7 \cdot u_1(x_i)$, de modo que $u_1(x_i)$ é a solução do método de Crank-Nicolson para uma única fonte pontual, localizada em p.

Como, de antemão, conhece-se a contribuição da única fonte pontual existente ($a_1 = 7$), essa tarefa tem como objetivo comprovar o funcionamento adequado do código desenvolvido. Após a resolução de um sistema trivial 1×1, obteve-se o resultado esperado $a_1 = 7$. A figura 1 mostra que a curva conhecida u_T e a curva reconstruída $7 \cdot u_1$ são, de fato, iguais. O erro obtido ($1,142742 \cdot 10^{-15}$), pela equação (8), deve-se apenas à aproximações numéricas na resolução do sistema.

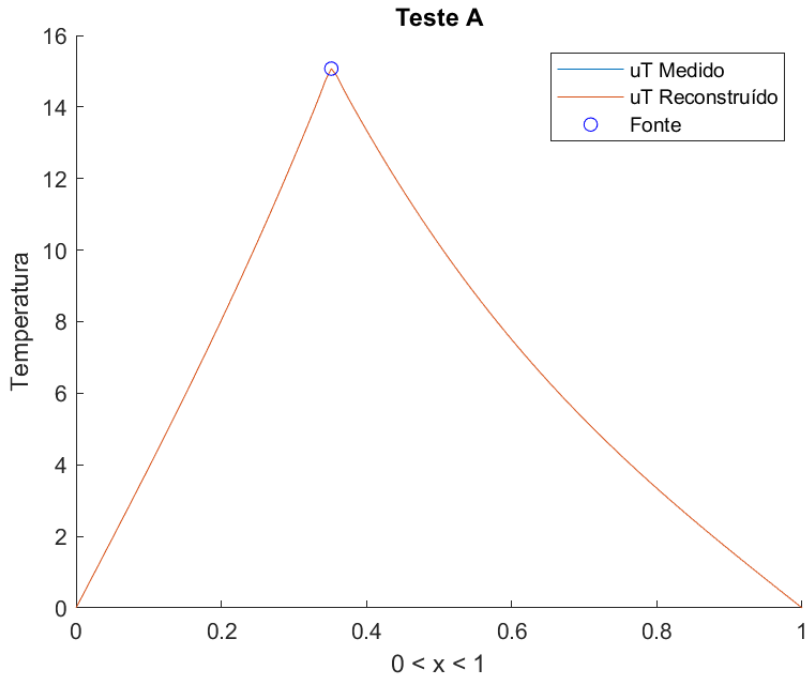


Figura 1: Distribuição de temperatura do teste A, com destaque na fonte pontual

4.2 Teste B

No teste B, ainda com $N = 128$, há um aumento no número de fontes. Agora, têm-se $nf = 4$ com $p_1 = 0,15$, $p_2 = 0,3$, $p_3 = 0,7$ e $p_4 = 0,8$. Novamente, é definido $u_T(x_i) = 2,3 \cdot u_1(T, x_i) + 3,7 \cdot u_2(T, x_i) + 0,3 \cdot u_3(T, x_i) + 4,2 \cdot u_4(T, x_i)$, de modo que a solução $a = [a_1, a_2, a_3, a_4]$ já é conhecida.

Nesse caso, almeja-se testar e comprovar o funcionamento da rotina de decomposição LDL^t e resolução de sistemas maiores, como o mostrado em (14). O erro E_2 obtido foi de $3,780624 \cdot 10^{-15}$ e deve-se, simplesmente, a arredondamentos sucessivos. A figura 2 mostra as curvas obtidas.

$$\begin{bmatrix} \langle u_1, u_1 \rangle & \langle u_2, u_1 \rangle & \langle u_3, u_1 \rangle & \langle u_4, u_1 \rangle \\ \langle u_1, u_2 \rangle & \langle u_2, u_2 \rangle & \langle u_3, u_2 \rangle & \langle u_4, u_2 \rangle \\ \langle u_1, u_3 \rangle & \langle u_2, u_3 \rangle & \langle u_3, u_3 \rangle & \langle u_4, u_3 \rangle \\ \langle u_1, u_4 \rangle & \langle u_2, u_4 \rangle & \langle u_3, u_4 \rangle & \langle u_4, u_4 \rangle \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \langle u_T, u_1 \rangle \\ \langle u_T, u_2 \rangle \\ \langle u_T, u_3 \rangle \\ \langle u_T, u_4 \rangle \end{bmatrix} \tag{14}$$

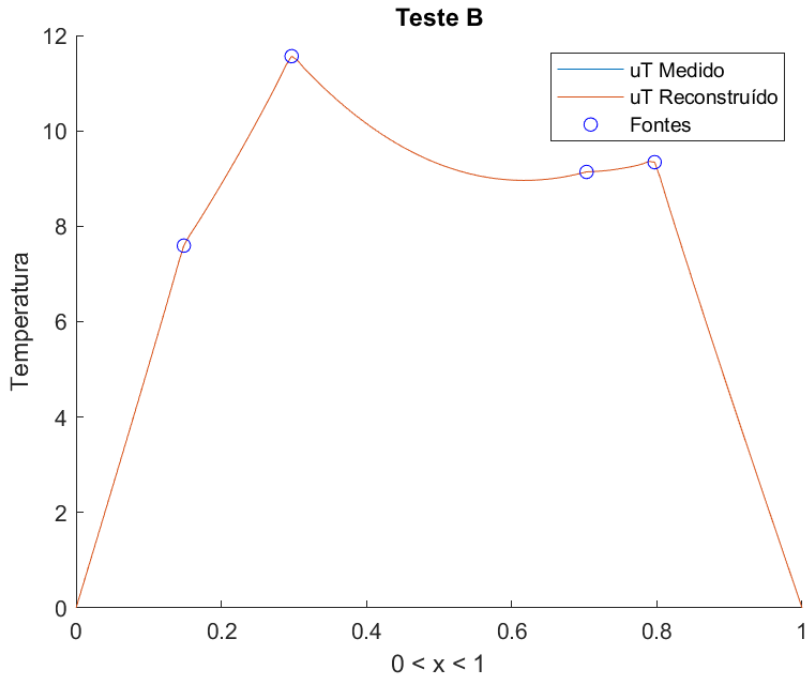


Figura 2: Distribuição de temperatura do teste B

4.3 Teste C

A partir de um arquivo (*teste.txt*), que fornece a posição de $nf = 10$ fontes pontuais e a distribuição de temperatura $u_T(x_i)$ em $T = 1$, deve-se encontrar os coeficientes a_k que correspondem às intensidades de cada fonte, construindo a solução que resolve o problema inverso.

O arquivo fornece dados para uma malha de discretização $\Delta x = \dfrac{1}{2048}$ e, para os casos em que $N < 2048$, pontos igualmente espaçados são armazenados, desde que N escolhido pelo usuário seja um submúltiplo de 2048.

As figuras 4a, 4b, 4c, 4d e 4e mostram as curvas geradas para os casos em que $N = 128,\ 256,\ 512,\ 1024$ e 2048, respectivamente. Não é possível, a partir de $N = 256$, notar diferenças visuais significativas entre as curvas.

Os erros E_2 obtidos pela equação (8) variam entre $2,445340 \cdot 10^{-2}$ e $3,591445 \cdot 10^{-12}$ e podem ser vistos na tabela 1 e na figura 3.

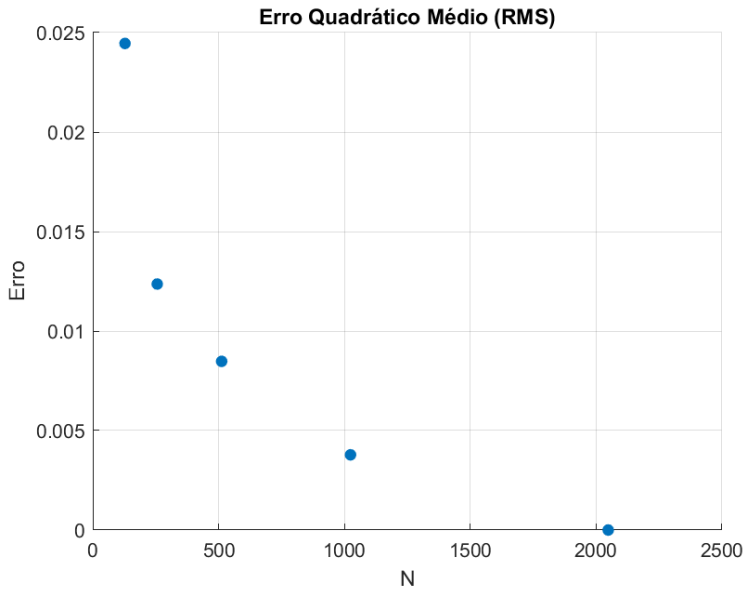


Figura 3: Erros do teste C

N	Erro máximo
128	$2,445340 \cdot 10^{-2}$
256	$1,236346 \cdot 10^{-2}$
512	$8,476628 \cdot 10^{-3}$
1024	$3,779310 \cdot 10^{-3}$
2048	$3,591445 \cdot 10^{-12}$

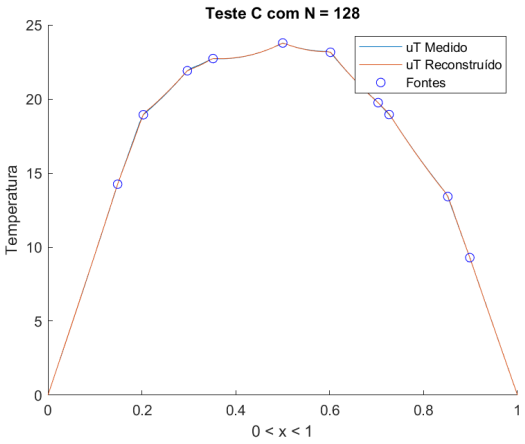
Tabela 1: Erros do teste C

Os coeficientes a_k podem ser visualizados na tabela 2. Vê-se convergência dos coeficientes conforme refina-se a malha, com o aumento de N, uma vez que os valores de erro (figura 3 e tabela 1) decrescem consideravelmente.

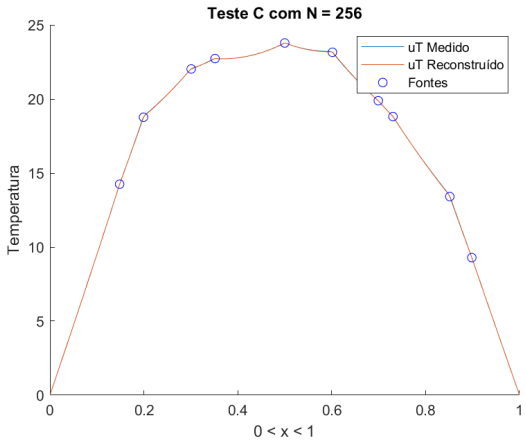
Posição da fonte	$N = 128$	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$
0.15	1.209123	0.904501	0.928688	1.007281	1.0
0.20	4.839259	5.077573	5.053708	4.992443	5.0
0.30	1.887241	2.100854	2.043701	1.985877	2.0
0.35	1.583400	1.414156	1.467671	1.513258	1.5
0.50	2.214504	2.229245	2.196763	2.192693	2.2
0.60	3.121295	3.104614	3.091131	3.095153	3.1
0.70	0.377340	0.509453	0.637588	0.652327	0.6
0.73	1.492348	1.386509	1.271687	1.253790	1.3
0.85	3.975139	3.949879	3.878095	3.879667	3.9
0.90	0.404145	0.414893	0.530557	0.529737	0.5

Tabela 2: Coeficientes obtidos para o teste C

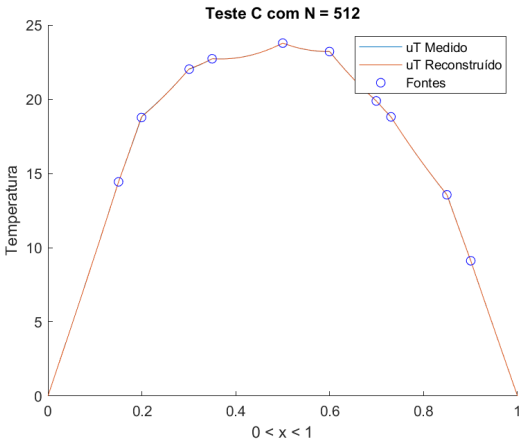
Figura 4: Distribuição de temperatura do teste C



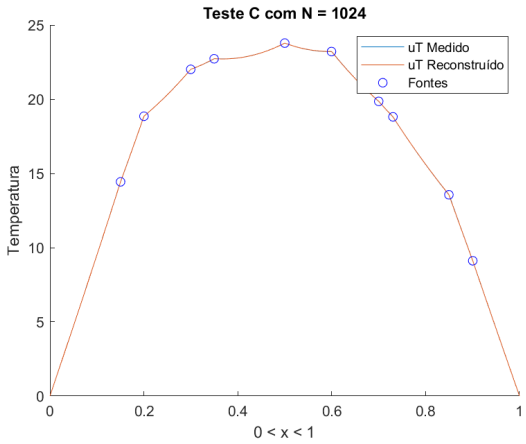
(a) $N = 128$



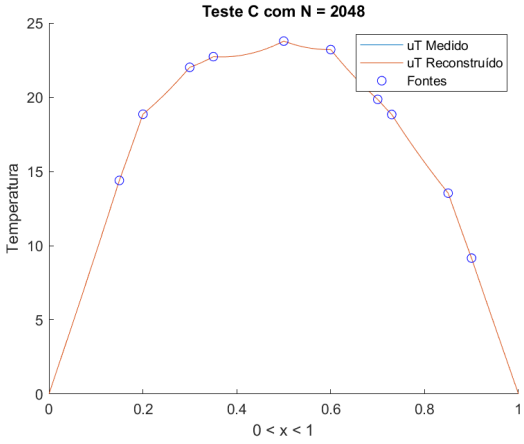
(b) $N = 256$



(c) $N = 512$



(d) $N = 1024$



(e) $N = 2048$

4.4 Teste D

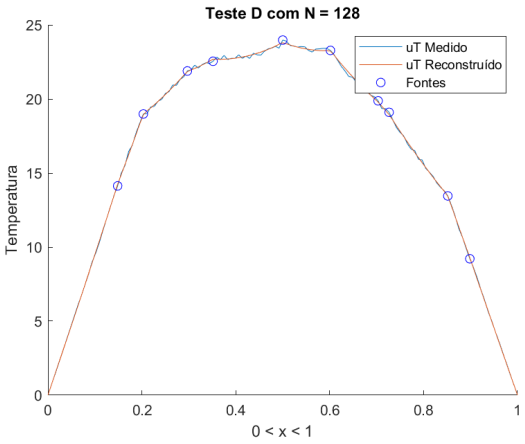
Por último, a partir do mesmo arquivo *.txt* do teste anterior, acrescentou-se ruído à distribuição de temperatura já conhecida. Tal ruído segue a forma $1+ r\epsilon$, sendo proporcional à curva original, com $\epsilon = 0,01$ e r um número aleatório entre -1 e 1, sendo que cada posição do vetor $u_T(x_i)$ é multiplicada por um destes valores gerados aleatoriamente.

No programa, feito em C++14, foi utilizado um PRNG, ou gerador de números pseudo-randômicos, como fonte do valor de r. Nesse caso, foi escolhido um Mersenne Twister, que é disponível livremente na linguagem[1] e é utilizado largamente na computação. Tal implementação requer uma semente, e para isso, utilizou-se um relógio de alta resolução[3], isto é, o tempo é a fonte de entropia. Assim, a partir desses dispositivos, é possível atingir uma distribuição numérica uniforme que, embora não seja realmente aleatória, é inteiramente adequada para os objetivos em questão.

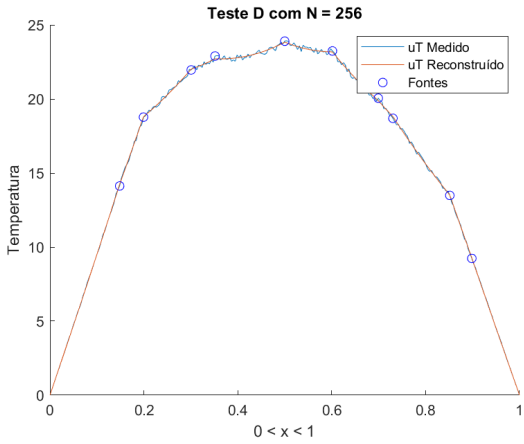
As figuras 5a, 5b, 5c, 5d e 5e mostram as curvas geradas para os casos em que $N = 128,\ 256,\ 512,\ 1024$ e 2048, respectivamente. Pela forma em que os fatores aleatórios são incorporados ao vetor, é possível notar um maior nível de ruído para maiores valores de temperatura.

Os erros E_2 obtidos pela equação (8) permanecem em torno de 0,1 e podem ser vistos individualmente, bem como os coeficientes a_k obtidos, na saída do programa. É possível explicar tal comportamento para o erro devido ao ruído introduzido na distribuição. Sabendo-se que o valor de $1 + r\epsilon$ varia uniformemente, de forma aleatória, entre 99% e 101% de $u_T(x_i)$, pode-se estimar o valor do erro quadrático médio, ou seja, a medida do segundo momento[6] do erro entre a curva de regressão e u_T medido (ruidoso), deve ser em torno de 1%. Portanto, o valor RMS do EQM, ou sua raiz quadrada, como descrito em (8), deve valer 0,1.

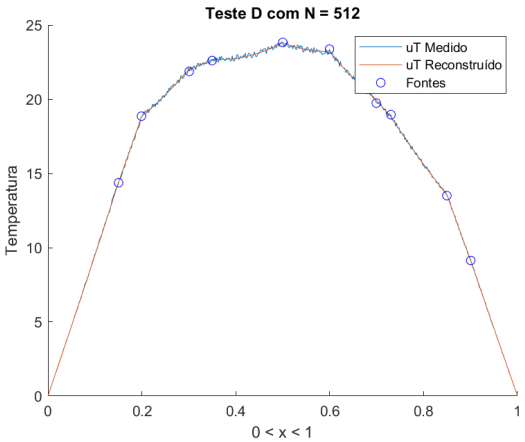
Figura 5: Distribuição de temperatura do teste D



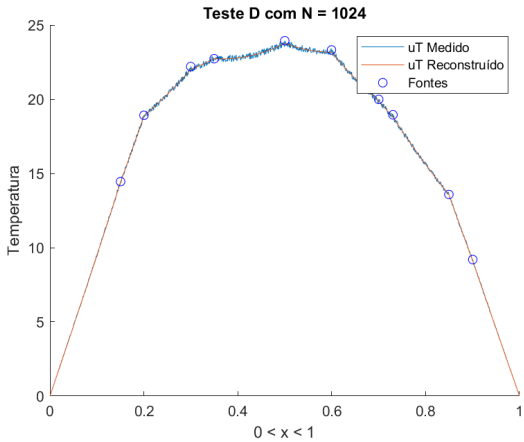
(a) $N = 128$



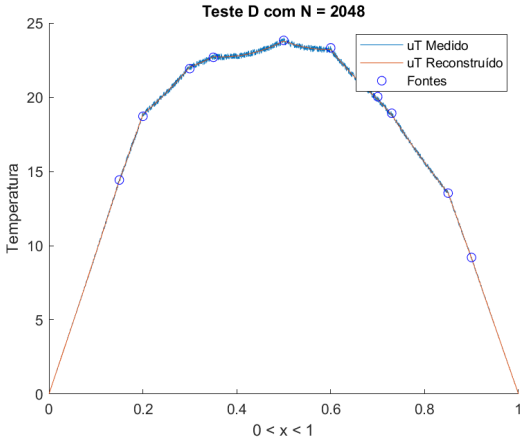
(b) $N = 256$



(c) $N = 512$



(d) $N = 1024$



(e) $N = 2048$

Referências

1 BROWN, Walter E. Random Number Generation in C++11. **Programming Language C++**, mar. 2013. https://isocpp.org/files/papers/n3551.pdf.

2 BURDEN, Richard L.; FAIRES, J. Douglas. **Numerical Analysis**. ninth edition: Cengage Learning, 2010.

3 DATE and time utilities — std::chrono::high_resolution_clock. https://en.cppreference.com/w/cpp/chrono/high_resolution_clock. Acessado em 28/06/2020.

4 NGUYEN, Duc. In: **Cholesky and LDLt Decomposition**. Jul. 2010. cap. 04.11.

5 PSEUDO Random Number Generation — std::mersenne_twister_engine. https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine. Acessado em 28/06/2020.

6 WEISSTEIN, Eric W. **Moment**. From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/Moment.html.

Apêndices

A Código em MATLAB para gerar as curvas

```
%% Teste A

N = 128;
x = linspace(0, 1, N + 1);
U = readmatrix('OutputA128.txt');
p = 0.35;
i = floor(N*p+1/2)+1;

figure;
hold
title("Teste A");
plot(x, U(:,1));
plot(x, U(:,2));
plot(x(i), U(i,1),'bo');
legend('uT Medido','uT Reconstruido', 'Fonte')
xlabel('0 < x < 1')
ylabel('Temperatura')
saveas(gcf,'FigA128.png')

%% Teste B

N = 128;
x = linspace(0, 1, N + 1);
U = readmatrix('OutputB128.txt');
p = [0.15; 0.30; 0.70; 0.80];
i = floor(N*p+1/2)+1;

figure;
hold
title("Teste B");
plot(x, U(:,1));
```

```
plot(x, U(:,2));
plot(x(i), U(i,1),'bo');
legend('uT Medido','uT Reconstruido', 'Fontes')
xlabel('0 < x < 1')
ylabel('Temperatura')
saveas(gcf,'FigB128.png')

%% Para os testes C e D
p = [0.15; 0.20; 0.30; 0.35; 0.50; 0.60; 0.70; 0.73; 0.85; 0.90];

%% Teste C
N = 128;
i = floor(N*p+1/2)+1;
x = linspace(0, 1, N + 1);
U = readmatrix("OutputC" + N + ".txt");

figure;
hold
title("Teste C com N = " + N);
plot(x, U(:,1));
plot(x, U(:,2));
plot(x(i), U(i,1),'bo');
legend('uT Medido','uT Reconstruido', 'Fontes')
xlabel('0 < x < 1')
ylabel('Temperatura')
saveas(gcf,"FigC" + N + ".png")

%% Teste C
N = 256;
i = floor(N*p+1/2)+1;
x = linspace(0, 1, N + 1);
U = readmatrix("OutputC" + N + ".txt");

figure;
hold
title("Teste C com N = " + N);
```

```
68 plot(x, U(:,1));
69 plot(x, U(:,2));
70 plot(x(i), U(i,1),'bo');
71 legend('uT Medido','uT Reconstruido', 'Fontes')
72 xlabel('0 < x < 1')
73 ylabel('Temperatura')
74 saveas(gcf,"FigC" + N + ".png")
75
76 %% Teste C
77 N = 512;
78 i = floor(N*p+1/2)+1;
79 x = linspace(0, 1, N + 1);
80 U = readmatrix("OutputC" + N + ".txt");
81
82 figure;
83 hold
84 title("Teste C com N = " + N);
85 plot(x, U(:,1));
86 plot(x, U(:,2));
87 plot(x(i), U(i,1),'bo');
88 legend('uT Medido','uT Reconstruido', 'Fontes')
89 xlabel('0 < x < 1')
90 ylabel('Temperatura')
91 saveas(gcf,"FigC" + N + ".png")
92
93 %% Teste C
94 N = 1024;
95 i = floor(N*p+1/2)+1;
96 x = linspace(0, 1, N + 1);
97 U = readmatrix("OutputC" + N + ".txt");
98
99 figure;
100 hold
101 title("Teste C com N = " + N);
102 plot(x, U(:,1));
103 plot(x, U(:,2));
```

```
104 plot(x(i), U(i,1),'bo');
105 legend('uT Medido','uT Reconstruido', 'Fontes')
106 xlabel('0 < x < 1')
107 ylabel('Temperatura')
108 saveas(gcf,"FigC" + N + ".png")
109
110 %% Teste C
111 N = 2048;
112 i = floor(N*p+1/2)+1;
113 x = linspace(0, 1, N + 1);
114 U = readmatrix("OutputC" + N + ".txt");
115
116 figure;
117 hold
118 title("Teste C com N = " + N);
119 plot(x, U(:,1));
120 plot(x, U(:,2));
121 plot(x(i), U(i,1),'bo');
122 legend('uT Medido','uT Reconstruido', 'Fontes')
123 xlabel('0 < x < 1')
124 ylabel('Temperatura')
125 saveas(gcf,"FigC" + N + ".png")
126
127 %% Teste D
128 N = 128;
129 i = floor(N*p+1/2)+1;
130 x = linspace(0, 1, N + 1);
131 U = readmatrix("OutputD" + N + ".txt");
132
133 figure;
134 hold
135 title("Teste D com N = " + N);
136 plot(x, U(:,1));
137 plot(x, U(:,2));
138 plot(x(i), U(i,1),'bo');
139 legend('uT Medido','uT Reconstruido', 'Fontes')
```

```
140 xlabel('0 < x < 1')
141 ylabel('Temperatura')
142 saveas(gcf,"FigD" + N + ".png")
143
144 %% Teste D
145 N = 256;
146 i = floor(N*p+1/2)+1;
147 x = linspace(0, 1, N + 1);
148 U = readmatrix("OutputD" + N + ".txt");
149
150 figure;
151 hold
152 title("Teste D com N = " + N);
153 plot(x, U(:,1));
154 plot(x, U(:,2));
155 plot(x(i), U(i,1),'bo');
156 legend('uT Medido','uT Reconstruido', 'Fontes')
157 xlabel('0 < x < 1')
158 ylabel('Temperatura')
159 saveas(gcf,"FigD" + N + ".png")
160
161 %% Teste D
162 N = 512;
163 i = floor(N*p+1/2)+1;
164 x = linspace(0, 1, N + 1);
165 U = readmatrix("OutputD" + N + ".txt");
166
167 figure;
168 hold
169 title("Teste D com N = " + N);
170 plot(x, U(:,1));
171 plot(x, U(:,2));
172 plot(x(i), U(i,1),'bo');
173 legend('uT Medido','uT Reconstruido', 'Fontes')
174 xlabel('0 < x < 1')
175 ylabel('Temperatura')
```

```
176 saveas(gcf,"FigD" + N + ".png")
177
178 %% Teste D
179 N = 1024;
180 i = floor(N*p+1/2)+1;
181 x = linspace(0, 1, N + 1);
182 U = readmatrix("OutputD" + N + ".txt");
183
184 figure;
185 hold
186 title("Teste D com N = " + N);
187 plot(x, U(:,1));
188 plot(x, U(:,2));
189 plot(x(i), U(i,1),'bo');
190 legend('uT Medido','uT Reconstruido', 'Fontes')
191 xlabel('0 < x < 1')
192 ylabel('Temperatura')
193 saveas(gcf,"FigD" + N + ".png")
194
195 %% Teste D
196 N = 2048;
197 i = floor(N*p+1/2)+1;
198 x = linspace(0, 1, N + 1);
199 U = readmatrix("OutputD" + N + ".txt");
200
201 figure;
202 hold
203 title("Teste D com N = " + N);
204 plot(x, U(:,1));
205 plot(x, U(:,2));
206 plot(x(i), U(i,1),'bo');
207 legend('uT Medido','uT Reconstruido', 'Fontes')
208 xlabel('0 < x < 1')
209 ylabel('Temperatura')
210 saveas(gcf,"FigD" + N + ".png")
```

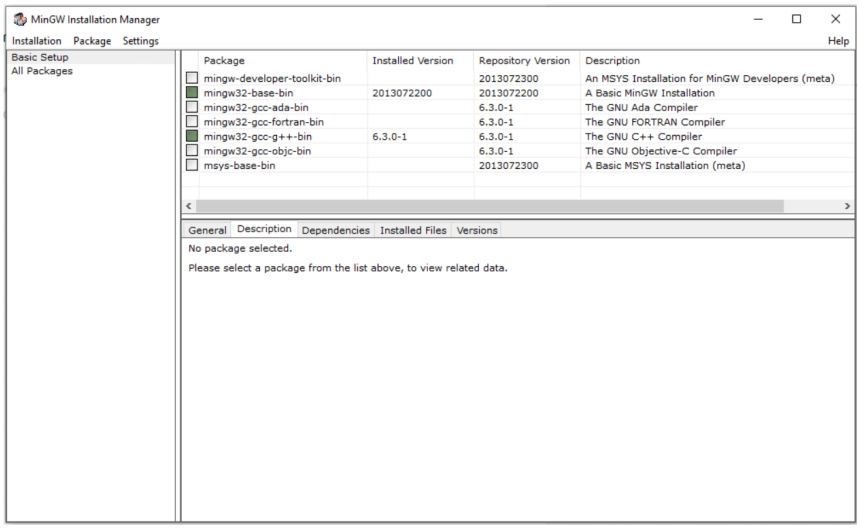
B LEIAME

MAP3121 - Métodos Numéricos e Aplicações (2020)

Exercício Programa 2

Instruções de compilação

1. Instale o compilador de linguagem C/C++ disponível em MINGW.
2. No instalador, selecione os seguintes pacotes:



3. Clique em *Installation* e selecione *Apply Changes*.
4. Para adicionar o compilador no PATH do Windows, siga as instruções do link.
5. Coloque os seguintes arquivos em um mesmo diretório:
   - main.cpp
   - Tarefa.h
   - Tarefa.cpp
6. Abra a janela de comando no caminho do diretório.
7. Execute o seguinte comando:

```
g++ -O2 -std=c++14 main.cpp Tarefa.cpp -o EP2.exe
```

Instruções de uso

O programa recebe a seleção do teste a ser realizado e, nos casos dos testes C e D, pede para que o usuário forneça o valor de N, a discretização do espaço.

Em todos os testes, o programa imprime tanto os coeficiente obtidos, quanto o erro quadrático calculado, no console. Além disso, é gerado um arquivo de saída que mostra o vetor uT de entrada, medido, ao lado do vetor uT recuperado a partir dos coeficientes. Note que, no teste D, o vetor uT medido, já possui ruído.

O nome dos arquivos de saída, no formato *.txt* seguem a seguinte convenção:

```
Output + <letra do teste> + <valor de N>
```

[0] https://github.com/eliasrr1/EP2-Numerico