MAP 2320 - Métodos numéricos em equações diferenciais II - EP1

Marcelo Monari Baccaro - 898926
Monik Souza Dourado - 9793183

I) Definição do problema

Considerando uma varra uniforme de comprimento l, com temperatura não uniforme, deitada no eixo x, de x=0 até x=l. Assumimos que os lados da barra são isolados e somente os extremos podem ser expostos. Além, assumimos que existe um termo fonte de calor na barra definido pela função f(t, x, k). Assim, obtemos um modelo parabólico não homogêneo com condições de Dirichlet nos extremos e dado inicial:

$$\begin{cases} u_t = u_{xx} + f(t,u,k), & x \in (0,l), t > 0 \\ u(t,0) = u(l,t) = 0, & t > 0 \\ u(x,0) = g(x), & x \in (0,l) \end{cases}$$

$$f(t,u,k) = k_1 * e^{-k_2 t}, \qquad t \in (0, T_{max})$$

$$g(x) = x * (1-x), \qquad x \in (0,l)$$

Onde $k = (k_1, k_2)$ é o vetor de parâmetros do modelo e $l = 1$.

O objetivo deste exercício é verificar como ajustar os parâmetros k do modelo para representar dados colhidos em experimentos.

II) Metodologia

1) Calcular a solução do problema com o parâmetro fixo $k = (0.5, 0.5)$ usando o método de Crank–Nicolson para diferenças finitas, que é dita a solução analítica;

2) Aplicar a esta solução uma perturbação para imitar um ruído;

3) Estimar os valores de k1 e k1 usando o mínimo quadrado como métrica e procurá los através de dois métodos: Hypercube, que discretiza e varre o conjunto de parâmetros, e Monte Carlo, que sorteia aleatoriamente os parâmetros;

4) Comparar a precisão e a eficiência de ambos os métodos descritos acima além de representar os resultados graficamente.

III) Desenvolvimento e testes numéricos

1) Solução analítica da EDP do calor

Para a solução do problema homogêneo f(t,u)=0

$$\begin{cases} u_t = u_{xx}, & x \in (0,l), t > 0 \\ u(t,0) = u(l,t) = 0, & t > 0 \\ u(x,0) = g(x), & x \in (0,l) \end{cases}$$

Temos a solução por separação de variáveis:

$$u(x,t) = X(x) * T(t) = \sum_{n=1}^{\infty} B_n(t) * sin\left(\frac{n\pi x}{l}\right)$$

$$B_n(t) = b_n * e^{-\left(\frac{n\pi}{l}\right)^2 t}$$

$$u(x,0) = \sum_{n=1}^{\infty} b_n * sin\left(\frac{n\pi x}{l}\right) = g(x)$$

$$b_n = \frac{2}{l} * \int_0^l g(x) * \sin\left(\frac{n\pi x}{l}\right) * dx$$

Para a solução não-homogênea, seguimos o mesmo formato de solução acima:

$$\begin{cases} u_t = u_{xx} + f(t,u), & x \in (0,l), t > 0 \\ u(t,0) = u(l,t) = 0, & t > 0 \\ u(x,0) = g(x), & x \in (0,l) \end{cases}$$

$$f(x,t) = \sum_{n=1}^{\infty} f_n(t) * sin\left(\frac{n\pi x}{l}\right)$$

$$f_n(t) = \frac{2}{l} * \int_0^l f(x,t) * sin\left(\frac{n\pi x}{l}\right) * dx$$

$$u(x,t) = \sum_{n=1}^{\infty} T_n(t) * sin\left(\frac{n\pi x}{l}\right)$$

$$u_t(x,t) = \sum_{n=1}^{\infty} T_n'(t) * sin\left(\frac{n\pi x}{l}\right)$$

$$u_{xx}(x,t) = \sum_{n=1}^{\infty} -\left(\frac{n\pi}{l}\right)^2 * T_n(t) * sin\left(\frac{n\pi x}{l}\right)$$

$$\sum_{n=1}^{\infty} T_n'(t) * sin\left(\frac{n\pi x}{l}\right) = \sum_{n=1}^{\infty} -\left(\frac{n\pi}{l}\right)^2 * T_n(t) * sin\left(\frac{n\pi x}{l}\right) + f_n(t) * sin\left(\frac{n\pi x}{l}\right)$$

$$T_n'(t) + \left(\frac{n\pi}{l}\right)^2 * T_n(t) = f_n(t)$$

$$T_n(t) = e^{-\left(\frac{n\pi}{l}\right)^2 t} * \left[\int_0^t e^{\left(\frac{n\pi}{l}\right)^2 \tau} * f_n(\tau) * d\tau + k_n\right]$$

$$u(x,0) = \sum_{n=1}^{\infty} k_n * sin\left(\frac{n\pi x}{l}\right) = g(x)$$

$$k_n = \frac{2}{l} * \int_0^l g(x) * sin\left(\frac{n\pi x}{l}\right) * dx$$

Para o caso de $g(x) = x * (1 - x)$, $f(x,t) = k_1 * e^{-k_2 t}$ e $l = 1$, temos que:

$$f_n(t) = \frac{2k_1}{n\pi} * (1 - \cos(n\pi)) * e^{-k_2 t}$$

$$k_n = \frac{2}{(n\pi)^3} * (2 - 2 * \cos(n\pi) - n\pi * \sin(n\pi))$$

$$T_n(t) = e^{-(n\pi)^2 t}\left[\frac{2k_1(1-\cos(n\pi))}{(n\pi)^3 - n\pi k_2}\left(e^{((n\pi)^2 - k_2)t} - 1\right) + \frac{2}{(n\pi)^3}(2 - 2\cos(n\pi) - n\pi\sin(n\pi))\right]$$

$$u(x,t) = \sum_{n=0}^{\infty} e^{-[(2n+1)\pi]^2} * \left\{\frac{4k_1\left(e^{\left[((2n+1)\pi)^2 - k_2\right]t} - 1\right)}{[(2n+1)\pi]^3 - (2n+1)\pi k_2} + \frac{8}{[(2n+1)\pi]^3}\right\} * \sin[(2n+1)\pi x]$$

2) Solução numérica da EDP do calor

A solução numérica foi implementada no MatLab utilizando o método de Crank-Nicolson para diferenças finitas, que é aplicado para o problema de transferência de calor não-homogêneo abaixo. [Burden - p733]

α = condutividade térmica
h = passo no espaço
L = comprimento da barra
M = número de vãos na discretização do espaço
k = passo no tempo
T = tempo máximo da simulação
N = número de vãos na discretização do tempo
W = matriz (M+1)x(N+1) em que os seus vetores colunas correspondem a solução por diferenças finitas do problema em (M+1) pontos discretizando o espaço num determinado tempo discreto dentro de [0,N]

$$x_i = i * h, para\ i \in [0, M]\ e\ h = \frac{L}{M}$$

$$t_j = j * k, para\ j \in [0, N]\ e\ k = \frac{T}{N}$$

(i) Condições de contorno $\mathrm{u}(\mathrm{t},0)=\mathrm{u}(\mathrm{l},\mathrm{t})=0$

$$w_{0,j}=w_{M,j}=0, para\ j\in[1,N]$$

(ii) Condições iniciais $u(x,0)=g(x)$

$$w_{i,0}=g(x_i), para\ i\in[0,M]$$

(iii) Método da diferença finita progressiva no j-ésimo no tempo:

$$\frac{w_{i,j+1}-w_{i,j}}{k}-\alpha^2*\frac{w_{i+1,j}-2*w_{i,j}+w_{i-1,j}}{h^2}=f(t_j)$$
$$\text{para } i\in[1,M-1]\ e\ j\in[1,N]$$

(iv) Método da diferença finita regressiva no (j+1)-ésimo em t:

$$\frac{w_{i,j+1}-w_{i,j}}{k}-\alpha^2*\frac{w_{i+1,j+1}-2*w_{i,j+1}+w_{i-1,j+1}}{h^2}=f(t_{j+1})$$
$$\text{para } i\in[1,M-1]\ e\ j\in[1,N]$$

(v) Média aritmética dos dois métodos anteriores com $a=k*\left(\frac{\alpha}{h}\right)^2$:

$$-\frac{a}{2}*w_{i+1,j+1}+(1+a)*w_{i,j+1}-\frac{a}{2}*w_{i-1,j+1}$$
$$=\frac{a}{2}*w_{i+1,j}+(1-a)*w_{i,j}+\frac{a}{2}*w_{i-1,j}+\frac{k}{2}*(f(t_j)+f(t_{j+1}))$$
$$\text{para } i\in[1,M-1]\ e\ j\in[1,N-1]$$

Na forma matricial:

$$A*W_{j+1}=B*W_j+C\quad para\ W_j=\{w_{i,j}\}$$

A = matrix tridiagonal (M+1)x(M+1) com as entradas na diagonal principal iguais a (1+a) e enquanto as entradas das diagonais secundárias são (-0.5*a)

B = matrix tridiagonal (M+1)x(M+1) com as entradas na diagonal principal iguais a (1-a) e enquanto as entradas das diagonais secundárias são (+0.5*a)

$$C = \left\{\frac{k}{2} * \left(f(t_j) + f(t_{j+1})\right)\right\}, para\ j \in [1, N]$$

Na função "fdms.m" (Finite Difference Method Solution), encontra se o trecho de código em que esta equação matricial linear é montada e resolvida pelo método direto da Eliminação de Gauss. [Burden, p361]

No programa "EP1.m", a solução considerada analítica (k=[0.5,0.5]) é armazenada na variável "u". Como o objetivo de gerar um ruído nesta solução, aplicamos uma perturbação randômica uniformemente distribuída em [-na,+na] e obtivemos "un". E é nesta váriável que a estimação é feita.

3) Métodos de estimação dos parâmetros k1 e k2

(i) Métrica de comparação: mínimos quadrados (least squares)

$$J = \sum_{i=1}^{M-1} \sum_{j=1}^{N} \left[u_{i,j}^{E} - u_{i,j}\right]^2$$

(ii) 1º método: Hypercube, que discretiza e varre o conjunto de parâmetros de k1 e k2 com um passo "ks", calcula por diferenças finitas a solução da EDP do calor para cada par [k1,k2], além do custo "J" dos mínimos quadrados em relação a solução analítica e retorna o par com o menor valor deste custo. Este método é implementado na função "ls_hcm_fdm.m".

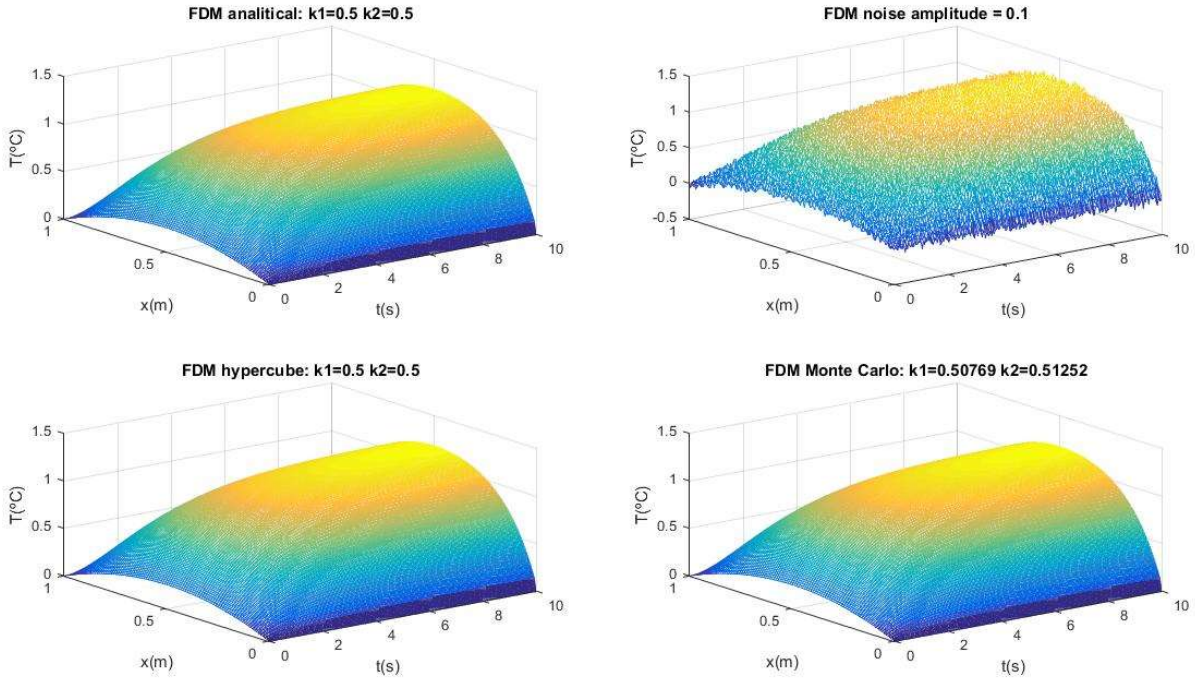
(iii) 2º método: Monte Carlo, que cria valores aleatórios (no caso, uniformemente distribuídos) para o par [k1,k2] em "Ncmc" tentativas, calcula por diferenças finitas a solução da EDP do calor para cada par, além do custo "J" dos mínimos quadrados em relação a solução analítica e retorna o par com o menor valor deste custo. Este método é implementado na função "ls_mcm_fdm.m".

4) Resultados numéricos

A estimação dos parâmetros k1 e k2 em feita em [0,1]. Nos resultados desta seção temos os seguintes valores fixados nas simulações:

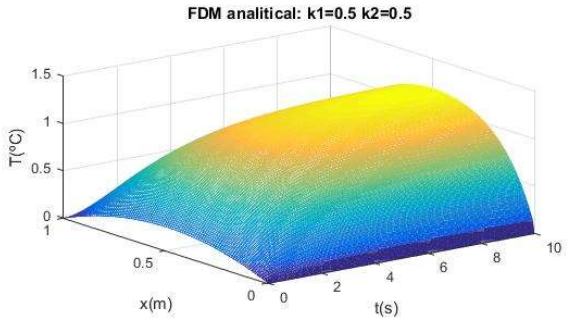
a = 1 (condutividade térmica)
L = 1 (comprimento da barra)
M = 100 (número de vãos na discretização do espaço)
T = 10 (tempo máximo da simulação)
N = 250 (número de vãos na discretização do tempo)
ks =0 .05 => $\left(\frac{1-0}{0.05}+1\right)^2 + 1 = 21^2 + 1 = 442$ estimações por Hypercube
Ncmc = 200 (estimações por Monte Carlo)

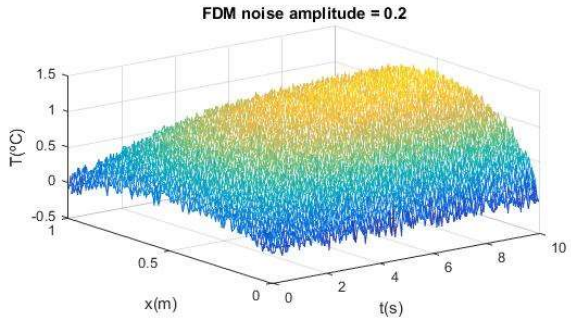
(i) na = 0.1 (amplitude do ruido)

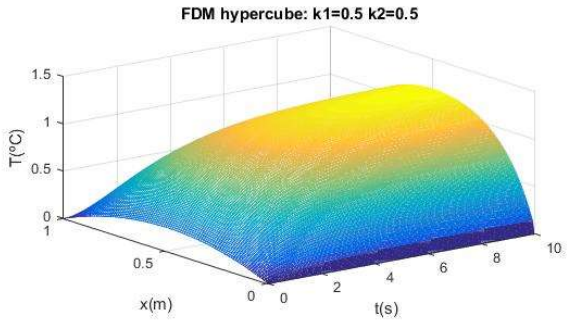


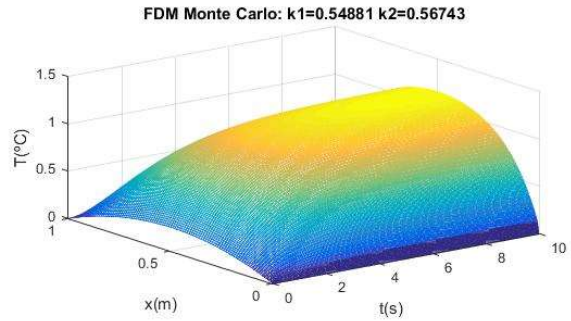
Hypercube: k = [0.5,0.5]
Monte Carlo: k = [0.507688818213242,0.512523050276335]

(ii) na = 0.2 (amplitude do ruido)



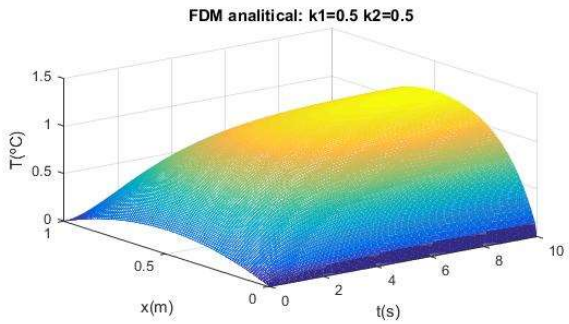


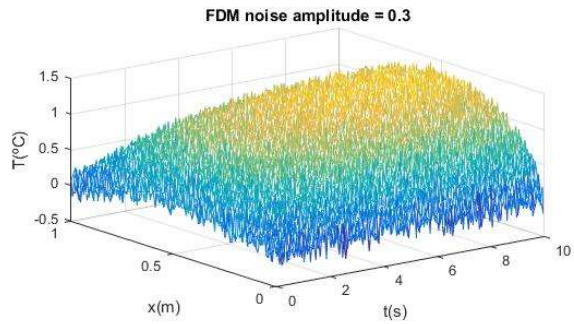


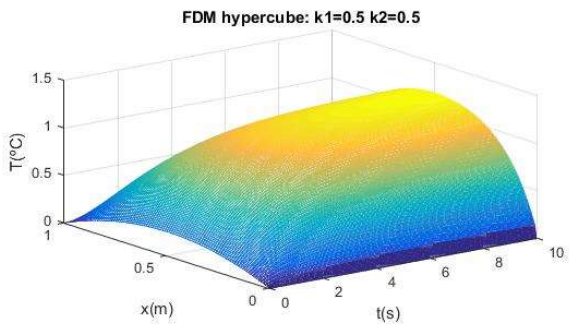


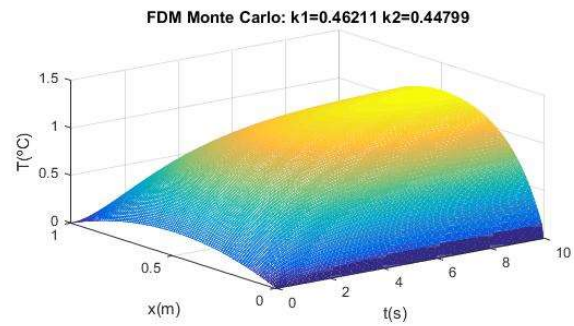
Hypercube: k = [0.5,0.5]
Monte Carlo: k = [0.548813277626217,0.567432400975456]

(iii) na = 0.3 (amplitude do ruido)



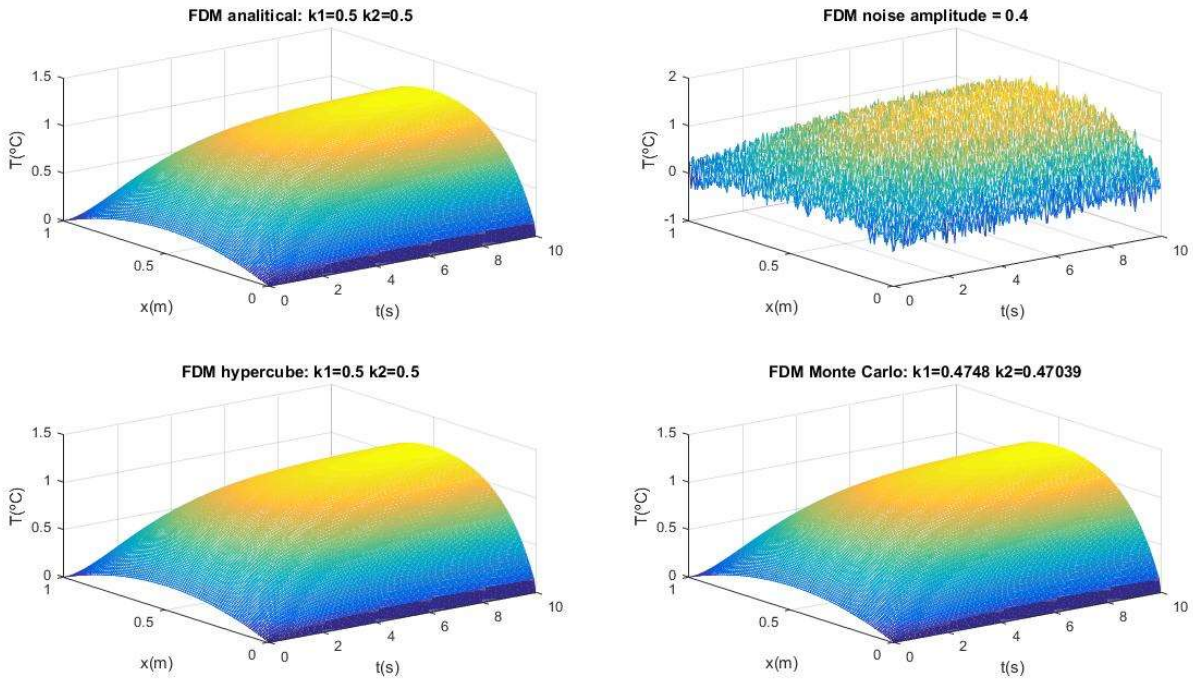






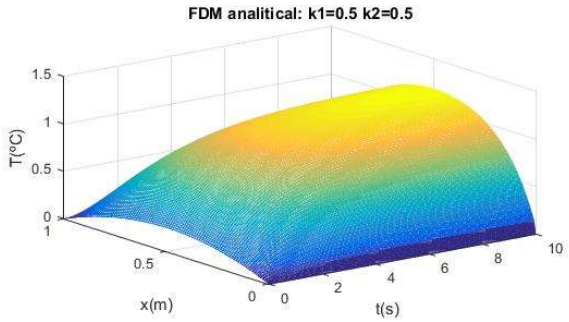
Hypercube: k = [0.5,0.5]
Monte Carlo: k = [0.462106402975797,0.447992760830674]

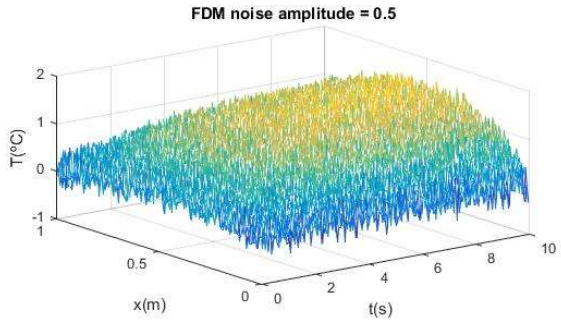
(iv) na = 0.4 (amplitude do ruido)

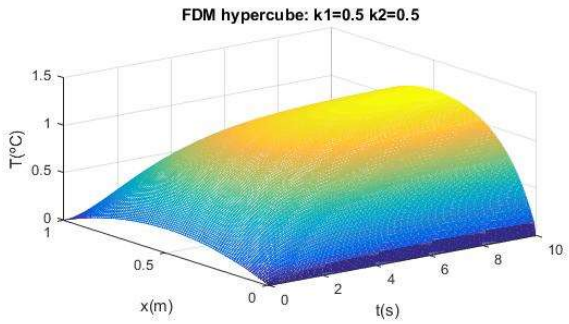


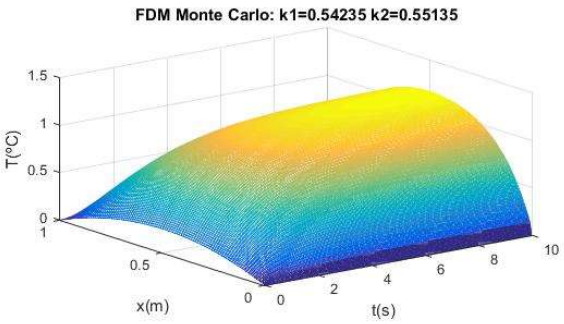
Hypercube: k = [0.5,0.5]
Monte Carlo: k = [0.474796608945116,0.470385719519422]

(v) na = 0.5 (amplitude do ruido)



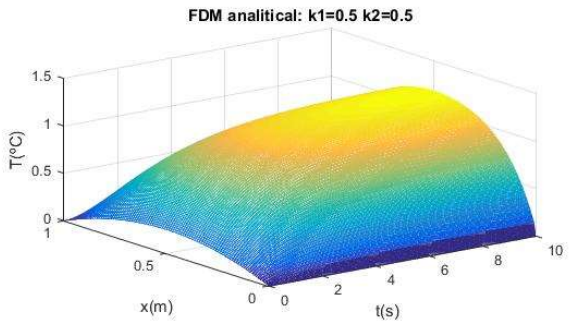


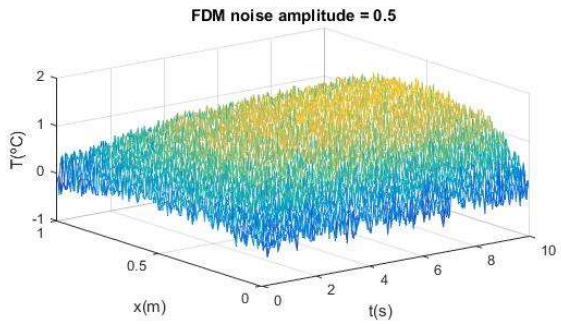


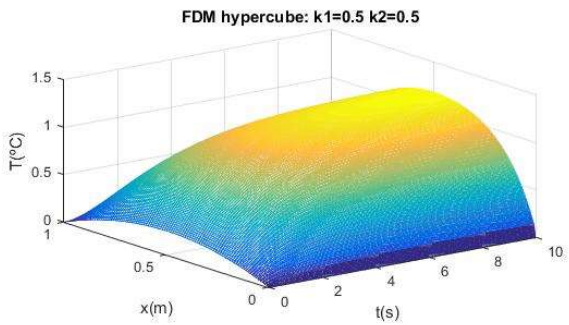


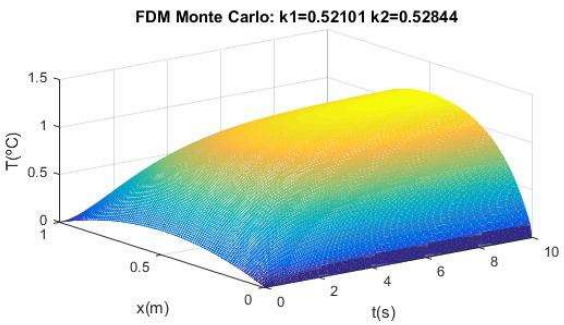
Hypercube: k = [0.5,0.5]
Monte Carlo: k = [0.542354516078700,0.551352222466142]

(vi) na = 0.5 (amplitude do ruido) com <u>Ncmc = 400</u>









Hypercube: k = [0.5,0.5]
Monte Carlo: k = [0.521014820687124,0.528443165847451]

IV) Conclusões

Antes de comentar os resultados numéricos, seria interessante discutir a influência dos parâmetros k1 e k2 na solução da EDP do calor não-homogênia. O sinal do parâmetro k1 determina se energia está entrando (k1>0), ou seja, a temperatura está aumentando com o tempo, ou saindo (k1<0), ou seja, a temperatura está diminuindo com o tempo. Enquanto o parâmetro k2 determina a estabilidade, pois, se k2>0, a distribuição da temperatura se torna estacionária para o tempo tendendo ao infinito (regime permanente), se k2>=0, a temperatura aumentará ou diminuirá infinitamente dependendo do sinal de k1.

Como pode ser observado na seção anterior, para os valores fixados na simulação, o método por Hypercube sempre tem uma estimação melhor que o método de Monte Carlo.

Entretanto a precisão nas casas decimais mais baixas do Hypercube depende do tamando do passo de varredura dos parâmetros, como esta varredura é algoritmo de complexidade $O(n^2)$, o número de iterações e, consequentemente, o tempo de excução aumentam muito para estimações mais precisas.

No método de Monte Carlo (no caso, usando números aleatórios uniformemente distribuídos), o aumento da precisão não precisa de uma aumento quadrático no número de iterações, pois aumentos lineares já mostram grande melhoria. Os resultados deste método foram próximos do Hypercube mesmo com menos da metade de iterações. Certamente, o método de Monte Carlo é mais atraente para aplicações em tempo real, como em embarcados, embora não seja tão simples gerar os números aleatórios.

V) Referências

[Burden] Numerical analysis - 9e - Burden (Cengage)

VI) Anexos

(i) EP1.m => main

```
clear; close; clc;
%%% System constants
L = 1; % bar length
T = 10; % max time
a = 1; % thermal conductivity
K = [0.5,0.5]; % source constants
```

```
na = 0.5; % noise amplitude

%%% Simulation constants
m = 100; % # gaps x
h = L/m; % step x
x = 0:h:L; % spacial vector
n = 250; % # gaps t
k = T/n; % step t
t = 0:k:T; % time vector
ks = 0.05; % k step for Hypercube
Nmcm = 400; % # Monte Carlo estimations

%%% Finite difference method (FDM) solution
u = fdms(a,m,h,x,n,k,t,K);
un = u + na * rand(m+1,n+1) - na * rand(m+1,n+1); % uniformly distributed
noise input signal [-na,+na]

%%% K vector estimation by FDM
Ke_hc = ls_hcm_fdm(un,a,m,h,x,n,k,t,ks); % hypercube
ue_hc = fdms(a,m,h,x,n,k,t,Ke_hc);
Ke_mc = ls_mcm_fdm(un,a,m,h,x,n,k,t,Nmcm); % Monte Carlo
ue_mc = fdms(a,m,h,x,n,k,t,Ke_mc);

%%% Plot output
subplot(2,2,1); mesh(t,x,u); title(['FDM analitical: k1=' num2str(K(1)) '
k2=' num2str(K(2))]);
    xlabel('t(s)'); ylabel('x(m)'); zlabel('T(°C)');
subplot(2,2,2); mesh(t,x,un); title(['FDM noise amplitude = '
num2str(na)]);
    xlabel('t(s)'); ylabel('x(m)'); zlabel('T(°C)');
subplot(2,2,3); mesh(t,x,ue_hc); title(['FDM hypercube: k1='
num2str(Ke_hc(1)) ' k2=' num2str(Ke_hc(2))]);
    xlabel('t(s)'); ylabel('x(m)'); zlabel('T(°C)');
subplot(2,2,4); mesh(t,x,ue_mc); title(['FDM Monte Carlo: k1='
num2str(Ke_mc(1)) ' k2=' num2str(Ke_mc(2))]);
    xlabel('t(s)'); ylabel('x(m)'); zlabel('T(°C)');
```

(ii) fdms.m => Finite Difference Method Solution

```
function u = fdms(a,m,h,x,n,k,t,K) % Finite difference method solution
la = a^2 * k / h^2; % lambda
u = zeros(m+1,n+1); % FDM solution declaration
u(:,1) = ic(x); % initial condition

% A*w = C | A -> D (main diagonal) & R (ratio)
R = zeros(m,1); % ratio for LSS
D = zeros(m+1,1); % main diagonal
R(1) = 0.5 * la / (la+1);
D(1) = la + 1;
for i=2:m
    D(i) = la + 1 - 0.5 * la * R(i-1);
    R(i) = 0.5 * la / D(i);
end
D(m+1) = la + 1 - 0.5 * la * R(m);

% Linear system solver A*w(j+1) = B*w(j) + S = C
for j=2:n+1
    s = 0.5 * k * (source(t(j-1),K) + source(t(j),K));
```

```
    C = zeros(m+1,1);
    C(1) = (1+la)*u(1,j-1) - 0.5*la*u(2,j-1) + s;
    for i=2:m
        C(i) = (1+la)*u(i,j-1) - 0.5*la*(u(i-1,j-1)+u(i+1,j-1)) + s + R(i-
1)*C(i-1);
    end
    C(m+1) = (1+la)*u(m+1,j-1) - 0.5*la*u(m,j-1) + s + R(m)*C(m);
    for i=m:-1:2 % Boundary conditions: w(t,0) = w(t,L) = 0
        u(i,j) = (C(i) + u(i+1,j)*0.5*la) / D(i);
    end
end
```

(iii) source.m => função excitante

```
function f = source(t,K)
f = K(1) * exp(-K(2)*t);
```

(iv) ic.m => initial conditions

```
function f = ic(x) % initial condition
f = x .* (1-x);
```

(v) ls_hcm_fdm.m => estimação por Hypercube e mínimos quadrados

```
function K = ls_hcm_fdm(un,a,m,h,x,n,k,t,ks)
%%% Least squares by hypercube method and finite difference method
% Initial estimation
K1=rand; K2=rand;
lsp = 0; % previous least square value
ue = fdms(a,m,h,x,n,k,t,[K1,K2]);
for i = 1:m+1
    for j = 1:n+1
        lsp = lsp + (un(i,j) - ue(i,j))^2;
    end
end
% Hypercube search estimation
kf = 1; % final k value
ki = 0; % initial k value
for k1 = ki:ks:kf
    for k2 = ki:ks:kf
        ue = fdms(a,m,h,x,n,k,t,[k1,k2]);
        ls = 0; % least square
        for i = 2:m
            for j = 2:n+1
                ls = ls + (un(i,j) - ue(i,j))^2;
            end
        end
        if ls < lsp % found better estimation
            K1 = k1;
            K2 = k2;
            lsp = ls;
        end
    end
end
K = [K1,K2];
```

(vi) ls_mcm_fdm.m => estimação por Monte Carlo e mínimos quadrados

```
function K = ls_mcm_fdm(un,a,m,h,x,n,k,t,Nmcm)
%%% Least squares by Monte Carlo method and finite difference method
% Initial estimation
K1=rand; K2=rand; % uniformly distributed draws
lsp = 0; % previous least square value
ue = fdms(a,m,h,x,n,k,t,[K1,K2]);
for i = 1:m+1
    for j = 1:n+1
        lsp = lsp + (un(i,j) - ue(i,j))^2;
    end
end
% Monte Carlo search estimation
for nn = 1:1:Nmcm-1
    k1=rand; k2=rand; % uniformly distributed draws in [0,1]
    ue = fdms(a,m,h,x,n,k,t,[k1,k2]);
    ls = 0; % least square
    for i = 2:m
        for j = 2:n+1
            ls = ls + (un(i,j) - ue(i,j))^2;
        end
    end
    if ls < lsp % found better estimation
        K1 = k1;
        K2 = k2;
        lsp = ls;
    end
end
K = [K1,K2];
```