

1 Esercizio 1

1.1 Descrizione del problema

Lo scopo di questa relazione è studiare un modello spaziale di diffusione del calore. Tale modello è ottenuto dall'equazione del calore classica perturbata da una funzione nonlineare. Consideriamo l'equazione di diffusione nonlineare descritta dalla seguente equazione:

$$\begin{cases} \frac{\delta u}{\delta t} = K(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}) + g(u) \\ u(0, x, y) \equiv u_0(x, y) \end{cases} \quad \begin{cases} g(u) := u(1-u)(u - \frac{1}{2}) \\ K = 10^{-4} \end{cases}$$

su $[0, 1]^2$ e con condizioni Neumann nulle al bordo. Per risolvere questo sistema abbiamo applicato il metodo di Eulero implicito definito dalla seguente espressione:

$$(I - \Delta_t \cdot A_{nn})U_{k+1}(\cdot) = U_k(\cdot) + \Delta_t \cdot G_k(\cdot)$$

dove,

1. I è la matrice identità;
2. U_k è la matrice che si ottiene valutando u ad un tempo fissato t_k ;
3. $A_{nn} = I \otimes A_n + A_n \otimes I$, dove A_n è la matrice della discretizzazione per funzioni di una variabile.
4. G_k è la matrice delle valutazioni di g a tempo fissato k .

Con la notazione $M(\cdot)$ si intende il vettore delle colonne di M .

1.2 Descrizione della sperimentazione

Per eseguire la sperimentazione abbiamo scelto tre diversi valori di K : $K_1 = 10^{-2}$, $K_2 = 10^{-4}$, $K_3 = 10^{-6}$ e abbiamo integrato numericamente l'equazione per un tempo tale da poter avere convergenza ad una soluzione. Per eseguire la sperimentazione abbiamo implementato due function che simulano la diffusione del calore su una membrana integrando l'equazione, il funzionamento di queste function è descritto nella sezione successiva.

1.3 Function

function data a lezione
diffusione

```
function diffusione(n)
%DIFFUSIONE_CALORE

% Discretizzazione del dominio nelle x e nelle y.
x = linspace(0, 1, n);
y = x;

% Passo di discretizzazione spaziale
h = x(2) - x(1);

% Coefficiente di diffusione per il nostro operatore laplaciano
K = 1e-2;

% Step di integrazione in tempo
dT = 0.01;

% Calcoliamo quanti step mi servono ad arrivare a t = 5.
nt = round(5 / dT);
```

```

% La sorgente di calore attiva solo fino a tempo 1/2, in un angolo del
% dominio.
f = @(x,y,t) (t < .5) .* (x < .3) .* (x > .1) .* (y < .9) .* (y > .7);

% Valutiamo il dato iniziale, nel nostro caso una campana esponenziale
% centrata in mezzo al dominio; per questo ci serve una griglia di punti
% che pu essere generata con il comando meshgrid.
[xx, yy] = meshgrid(x, y);

u0 = @(x,y) exp(-100*((x-.5).^2 + (y-.5).^2));

U = u0(xx, yy);

mesh(x, y, U);

% Costruiamo l'operatore Laplaciano come una matrice sparsa, per rendere i
% conti sufficientemente veloci. Questo di vitale importanza!
A = K * spdiags(ones(n-2, 1) * [1 -2 1], -1:1, n-2, n-2) / h^2;
AA = kron(speye(n-2), A) + kron(A, speye(n-2));

% Cominciamo ad integrare il sistema usando Eulero implicito
for j = 1 : nt
    % Risolvo il sistema (I - dT * L) u_{n+1} = u_n + dT * f + dT * BC,
    % dove f il termine sorgente
    rhs = U(2:end-1, 2:end-1) + ...
        dT * f(xx(2:end-1,2:end-1), yy(2:end-1,2:end-1), j*dT);

    Unew = (speye(size(AA)) - dT * AA) \ reshape(rhs, (n-2)^2, 1);

    U(2:end-1,2:end-1) = reshape(Unew, n-2, n-2);

    % Plot della soluzione
    mesh(x, y, U);
    title(sprintf('Time = %f', j * dT));
    axis([0 1 0 1 0 1])
    drawnow;
    pause(0.1);
end

end

```

Riportiamo di seguito le due function utilizzate. Questa function **diffusione2** implementa la diffusione per un problema in due dimensioni con condizioni di Neumann nulle al bordo. Abbiamo realizzato questa function modificando quella fornita a lezione.

```

function diffusione2(n)
    %DIFFUSIONE_CALORE
    % Questa function implementa la diffusione per un problema in due
    % dimensioni con condizioni di Neumann nulle al bordo.
    %In input:
    %n = dimensione della griglia.
    % Discretizzazione del dominio nelle x e nelle y.
    x = linspace(0, 1, n);
    y = x;

    % Passo di discretizzazione spaziale
    h = x(2) - x(1);

```

```

% Coefficiente di diffusione per il nostro operatore
    laplaciano
K = 1e-2;

% Step di integrazione in tempo
dT = 0.01;

% Calcoliamo quanti step mi servono ad arrivare a t = 5.
nt = round(5 / dT);

% La sorgente di calore attiva solo fino a tempo 1/2, in
    un angolo del
% dominio.
f = @(x,y,t) (t < .5) .* (x < .3) .* (x > .1) .* (y < .9)
    .* (y > .7);

% Valutiamo il dato iniziale, nel nostro caso una campana
    esponenziale
% centrata in mezzo al dominio; per questo ci serve una
    griglia di punti
% che pu essere generata con il comando meshgrid.
[xx, yy] = meshgrid(x, y);

u0 = @(x,y) exp(-100*((x-.5).^2 + (y-.5).^2));

U = u0(xx, yy);

mesh(x, y, U);

% Costruiamo l'operatore Laplaciano come una matrice sparsa
    , per rendere i
% conti sufficientemente veloci. Questo di vitale
    importanza!
A0 = spdiags(ones(n, 1) * [1 -2 1], -1:1, n, n);
A0(1, 1) = -1;
A0(end, end) = -1;
A = K * A0 / h^2;
AA = kron(speye(n), A) + kron(A, speye(n));

% Cominciamo ad integrare il sistema usando Eulero
    implicito
for j = 1 : nt
    % Risolvo il sistema  $(I - dT * L) u_{\{n+1\}} =$ 
         $u_n + dT * f + dT * BC,$ 
    % dove f il termine sorgente
    rhs = U + dT * f(xx, yy, j*dT);

    Unew = (speye(size(AA)) - dT * AA) \
        reshape(rhs, (n)^2, 1);

    U = reshape(Unew, n, n);

    % Plot della soluzione
    mesh(x, y, U);
    title(sprintf('Time = %f', j * dT));
    axis([0 1 0 1 0 1])
    drawnow;
    pause(0.1);
end

```

end

Questa function **diffusione3** utilizza la precedente per generare la simulazione della soluzione del problema considerato.

```
function diffusione3(n)
%DIFFUSIONE_CALORE
%In input:
%n=dimensione della griglia
% Discretizzazione del dominio nelle x e nelle y.
x = linspace(0, 1, n);
y = x;

% Passo di discretizzazione spaziale
h = x(2) - x(1);

    % Coefficiente di diffusione per il nostro operatore
    % laplaciano
    K = 1e-2;
    %K = 1;

    % Step di integrazione in tempo
    dT = 0.05;

    % Calcoliamo quanti step mi servono ad arrivare a t = 5.
    nt = round(500 / dT);

    % La sorgente di calore attiva solo fino a tempo 1/2, in
    % un angolo del
    % dominio.
    g = @(x,y,t,u) u .* (1 - u) .* (u - 0.5);

    % Valutiamo il dato iniziale, nel nostro caso una campana
    % esponenziale
    % centrata in mezzo al dominio; per questo ci serve una
    % griglia di punti
    % che pu essere generata con il comando meshgrid.
    [xx, yy] = meshgrid(x, y);

    u0 = @(x,y) 0.5 + randn(n);
    %u0 = @(x,y) x;

    U = u0(xx, yy);

    mesh(x, y, U);

    % Costruiamo l'operatore Laplaciano come una matrice sparsa
    % , per rendere i conti sufficientemente veloci. Questo
    % di vitale importanza!
    A0 = spdiags(ones(n, 1) * [1 -2 1], -1:1, n, n);
    A0(1, 1) = -1;
    A0(end, end) = -1;
    A = K * A0 / h^2;
    AA = kron(speye(n), A) + kron(A, speye(n));

    % Cominciamo ad integrare il sistema usando Eulero
    % implicito
    for j = 1 : nt
        % Risolvo il sistema (I - dT * L) u_{n+1} =
        % u_n + dT * f + dT * BC,
        % dove f il termine sorgente
```

```

        rhs = U + dT * g(xx, yy, j*dT, U);

        Unew = (speye(size(AA)) - dT * AA) \
            reshape(rhs, (n)^2, 1);

        U = reshape(Unew, n, n);

        % Plot della soluzione
        mesh(x, y, U);
        title(sprintf('Time = %f', j * dT));
        axis([0 1 0 1 0 1])
        drawnow;
        pause(0.1);

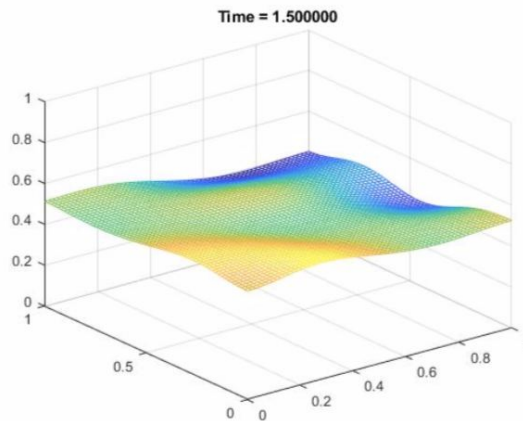
    end

end

```

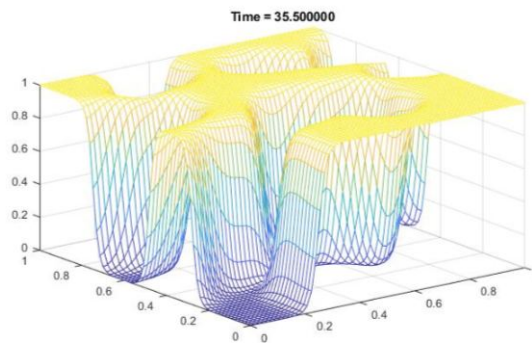
1.4 Commenti

Osservando i grafici si può notare che, più K è grande, migliore e più rapida è la convergenza, mentre se K è molto piccolo, il suo contributo diventa trascurabile. Infatti nell'immagine seguente si può vedere come la soluzione converga già dopo solo 1.5 secondi.



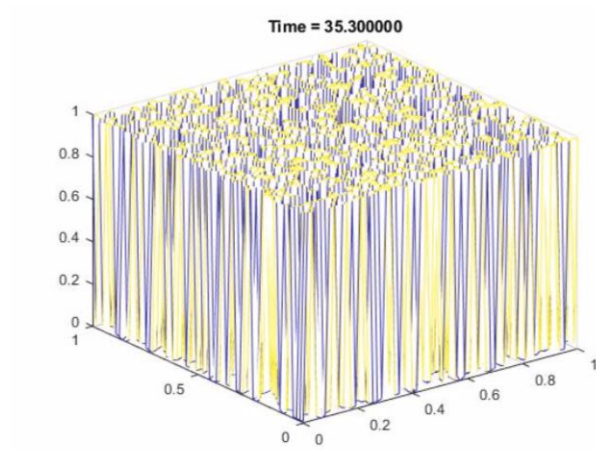
(a) Grafico nel caso $K = 10^{-2}$

Nella figura sottostante si vede che dopo 35 secondi c'è stata convergenza.



(a) Grafico nel caso $K = 10^{-4}$

Nel caso $k = 10^{-6}$ illustrato sotto si vede che non c'è convergenza, quantomeno in tempi brevi.



(a) Grafico nel caso $K = 10^{-6}$