Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Risoluzione Numerica di Equazioni Differenziali alle Derivate Parziali di Tipo Ellittico e Parabolico in Ambiente Matlab PDE toolbox

Gaetano Tartaglione, Pasqua D'Ambra

RT-ICAR-NA-2012-10 ***Novembre 2012***

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Risoluzione Numerica di Equazioni Differenziali alle Derivate Parziali di Tipo Ellittico e Parabolico in Ambiente Matlab PDE toolbox

Gaetano Tartaglione[1], Pasqua D'Ambra[2]

RT-ICAR-NA-2012-10 ***Data: Novembre 2012***

---

1 Seconda Università di Napoli, Facoltà di Ingegneria, via Roma, 29, 81031 Aversa
2 Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, via P. Castellino, 111, 80131 Napoli.

RISOLUZIONE NUMERICA DI EQUAZIONI DIFFERENZIALI ALLE DERIVATE PARZIALI DI TIPO ELLITTICO E PARABOLICO IN AMBIENTE MATLAB PDE TOOLBOX

GAETANO TARTAGLIONE[¶], PASQUA D'AMBRA[‡]

Sommario. Si riportano i risultati ottenuti dall'analisi e dalla sperimentazione delle funzionalità del PDEtoolbox di Matlab dedicate alla risoluzione di problemi legati ad Equazioni Differenziali alle Derivate Parziali di tipo Ellittico e Parabolico. La sperimentazione di queste funzionalità è avvenuta attraverso la risoluzione di alcuni problemi test, ovvero problemi la cui soluzione analitica è nota. Il testo è diviso in due parti: la prima parte è dedicata ai problemi di tipo ellittico, la seconda parte è dedicata ai problemi di tipo parabolico. Si descrivono le diverse metodologie con cui è possibile definire il problema che si vuole risolvere (dominio, condizioni al contorno, coefficienti dell'equazione) e i modi con cui è possibile rappresentare la soluzione ottenuta. Si riportano, infine, le stime di convergenza dell'*errore globale*.

Introduzione. Il seguente documento nasce al termine del tirocinio formativo svolto presso la sede di Napoli dell'*Istituto di Calcolo e Reti ad Alte Prestazioni* (ICAR). Esso ha lo scopo di sintetizzare tutte le attività svolte, i risultati ottenuti e le conoscenze acquisite. Durante il tirocinio sono state approfondite alcune metodologie per la risoluzione di problemi legati ad Equazioni Differenziali alle Derivate Parziali (da qui semplicemente EDP) di tipo Ellittico e Parabolico. Inoltre, sono state analizzate e sperimentate le funzionalità che Matlab dedica a questo tipo di problemi, ovvero è stato approfondito l'uso del PDEtoolbox.

1. Risoluzione numerica di problemi di tipo ellittico. In questa sezione si descrivono brevemente i metodi numerici per risolvere problemi legati ad EDP di tipo ellittico in ambiente Matlab, PDE toolbox.

In Matlab le EDP di tipo ellittico devono essere riscritte nella forma:

$$-\nabla\cdot(c\nabla u)+au=f \quad \text{definita in} \quad \Omega \ , \tag{1}$$

mentre per le condizioni al contorno bisogna far riferimento ad una delle seguenti espressioni:

$$hu=r \quad \text{in} \quad \partial\Omega \tag{2}$$

$$\boldsymbol{n}\cdot(c\nabla u)+qu=g \quad \text{in} \quad \Omega \tag{3}$$

La (2) indica condizioni alla Dirichlet, poiché si conosce la funzione incognita u sul contorno del dominio. La (3) indica condizioni miste alla Robin, nel caso in cui la funzione q è identicamente nulla si hanno condizioni alla Neumann, poiché si conosce sulla frontiera del dominio solo la derivata di u nella direzione normale alla frontiera.

¶ Seconda Università di Napoli, Facoltà di Ingegneria, via Roma, 29, 81031 Aversa (gaetano.tartaglione@studenti.unina2.it)

‡ Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, via P. Castellino, 111, 80131 Napoli.

Le funzioni $c, a, f,$ sono funzioni definite in Ω .

I metodi di risoluzione sono stati così analizzati: partendo da problemi test, si studiano e si confrontano i possibili metodi di risoluzione numerica del problema ed i risultati ottenuti, usando come confronto i risultati esatti o apposite soluzioni numeriche. Infine sono descritte brevemente le varie funzioni Matlab utilizzate.

1.1. Definizione problemi test. I problemi test di tipo ellittico studiati sono i seguenti:

- Problema di Laplace in 2D:

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0 \quad \text{con} \quad (x,y)\in]0,1[\times]0,1[\tag{4}$$

con condizioni al contorno alla Dirichlet:

$$u(0,y)=u(1,y)=0, \;\; u(x,0)=u(x,1)=\sin(\pi x) \tag{5}$$

La soluzione esatta è rappresentata dalla funzione:

$$u(x,y)=sech(0.5\pi)\cosh(\pi(y-0.5))\sin(\pi x) \tag{6}$$

- Problema di Poisson in 2D:

$$-\nabla\cdot(\boldsymbol{K}\cdot\nabla u)=1 \quad \text{con} \quad (x,y)\in]0,1[\times]0,1[\tag{7}$$

dove $\boldsymbol{K}=\begin{bmatrix} 0.2525 & 0.2475 \\ 0.2475 & 0.2525 \end{bmatrix}$

con condizioni al contorno alla Dirichlet:

$$u(x,y)=0 \quad \text{per} \quad (x,y)\in\partial\Omega \tag{8}$$

- Come caso particolare di problema ellittico è stato considerato il seguente problema test monodimensionale, corrispondente ad un problema ai limiti definito dalla seguente Equazione Differenziale Ordinaria (da qui ODE) del secondo ordine:

$$\frac{d^2 u}{dx^2}=-\frac{1}{x} \quad \text{con} \quad 0<x<1 \tag{9}$$

con condizioni al contorno: $u(0)=u(1)=0$ (10)

la soluzione esatta è rappresentata dalla funzione $u(x)=-xlog(x)$ (11)

1.2. Risoluzione problemi test. Il PDEtoolbox di Matlab mette a disposizione dell'utente diversi strumenti utili per la risoluzione di EDP. Principalmente i modi per risolvere numericamente le EDP sono due: mediante l'interfaccia grafica del PDEtoolbox (Graphical User Interface, da qui GUI), oppure richiamando apposite funzioni direttamente dalla linea di comando (Command Line, da qui CL) e quindi creando appositi script files. Nel seguito si descriveranno entrambi gli approcci.

1.2.1. Soluzione problema di Laplace in 2D.

- Soluzione mediante GUI:

Si accede alla GUI semplicemente digitando il comando **pdetool** tramite CL.

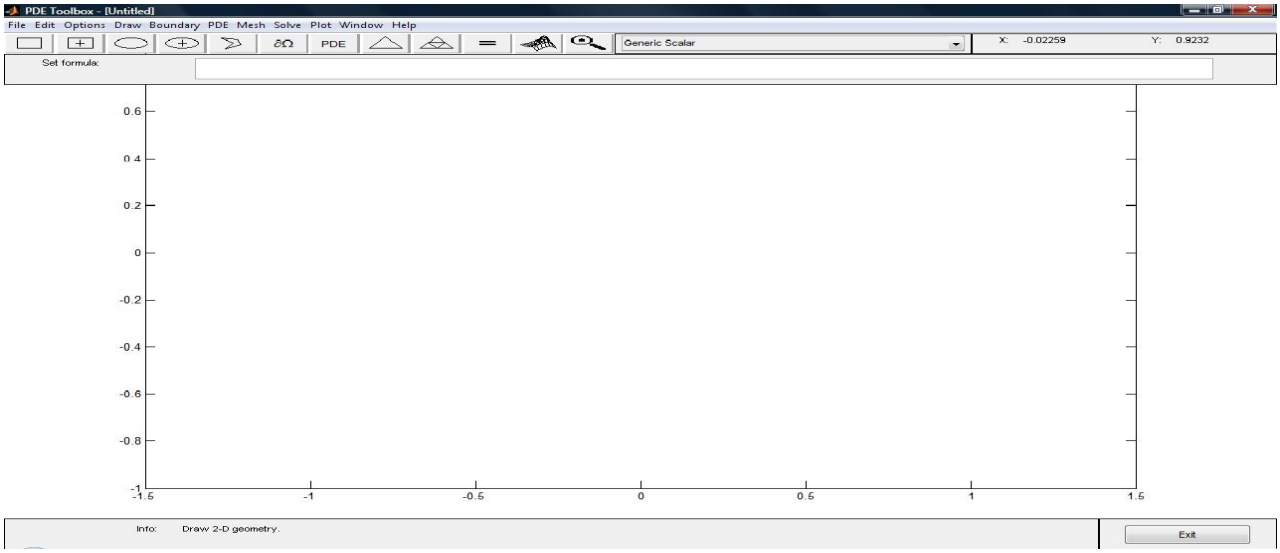

Fig. 1: Panoramica pdetool

Mediante la GUI, la risoluzione del problema e lo studio della soluzione avviene in poche fasi:

1. **Impostare l'area di disegno:** in questo modo sarà più semplice generare il dominio del problema.
   - Dal menù **Options** selezionare:
     - **Grid** in questo modo viene rappresentata una griglia nell'area di disegno.
     - **Axes Limits**: con questa funzione è possibile modificare gli estremi dell'area di disegno. Impostare come limiti [-0.5;1.5] sia per l'asse x, che per l'asse y.
2. **Generare il dominio del problema:** il problema test presenta come dominio un quadrato di lato unitario, con vertici in $(0,0)(1,0)(1,1)(0,1)$.
   - Entrare nel menù **Draw** e selezionare:
     - **Rectangle/Square:** questa opzione permette di disegnare domini rettangolari o quadrati.
   - Spostarsi nell'area di disegno e posizionare il cursore nell'origine (0;0). Tenendo premuto il tasto destro del mouse spostarsi fino al punto (1;1) e rilasciare il tasto. Così facendo si ottiene il dominio del problema (un quadrato di lato 1 con vertici in (0,0) ed in (1,1)), questo di default viene chiamato **SQ1** . Un rapido controllo viene effettuato cliccando due volte sul quadrato generato e quindi controllando i dati del dominio.
   - Ora l'area di disegno appare come in Fig. 2.

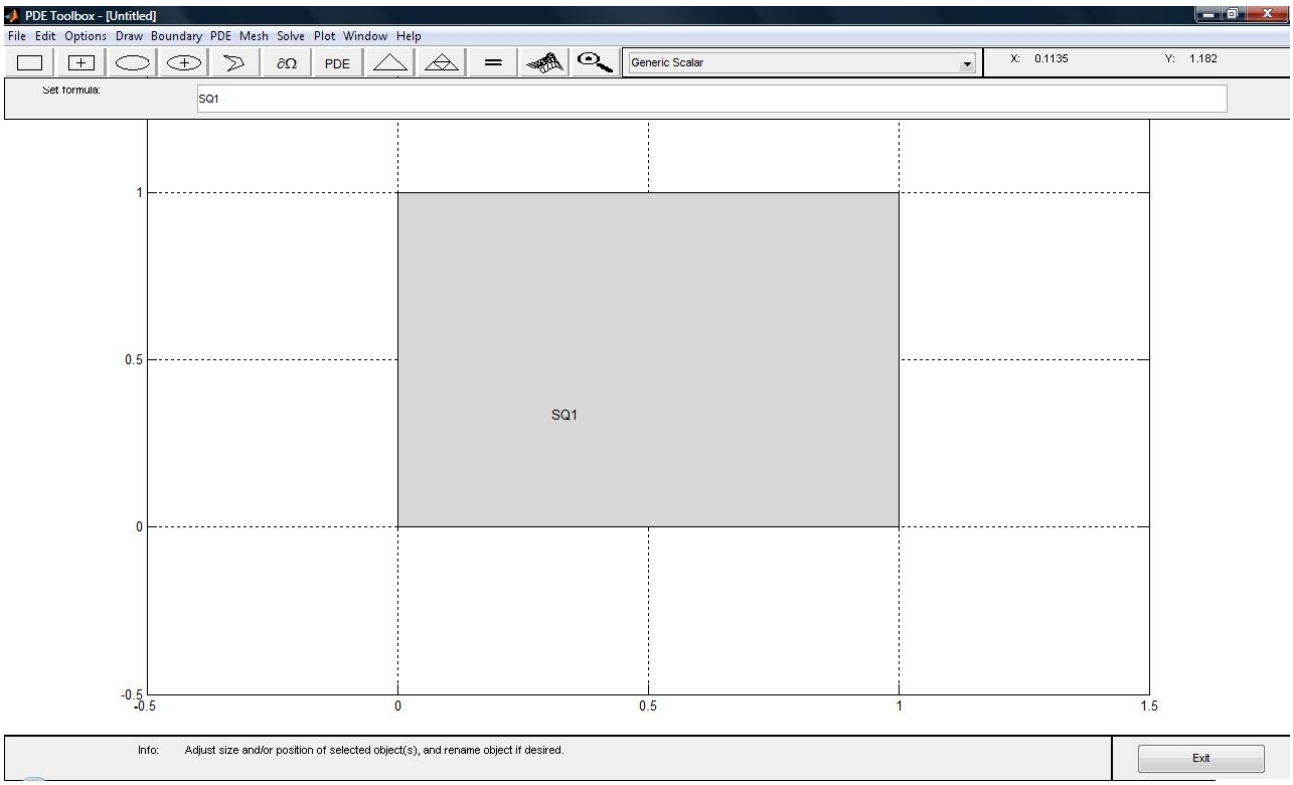

Fig. 2: Dominio SQ1

3. **Impostare le condizioni al contorno del problema:**
   - Entrare nel menù **Boundary** e selezionare:
     - **Boundary Mode,** fatto questo il contorno del dominio viene diviso in quattro parti ed è possibile impostare le condizioni per i quattro lati del quadrato. Le condizioni vanno inserite un lato per volta: posizionare il cursore sul lato e cliccare due volte.
       - Impostare su tutti i lati condizioni di Dirichlet con h=1. Poi impostare: per il lati con y=0 e y=1, r =sin(pi*x); mentre per i lati con x=0 e x=1, r =0.
4. **Creare la griglia all'interno del dominio:**
   - Entrare nel menù **Mesh** e selezionare in sequenza:
     - **Mesh Mode:** per entrare nella modalità di generazione della griglia.
     - **Inizialize Mesh:** in questo modo il software genera nel dominio precedentemente creato una griglia non strutturata di tipo triangolare (per una descrizione dettagliata fare riferimento al paragrafo 1.3, nella sezione relativa alla funzione *initmesh*) .
     - **Refine Mesh:** con questo comando la griglia all'interno del dominio viene raffinata, ovvero aumenta il numero di triangoli.
   - La griglia finale che si ottiene è mostrata in Fig. 3.

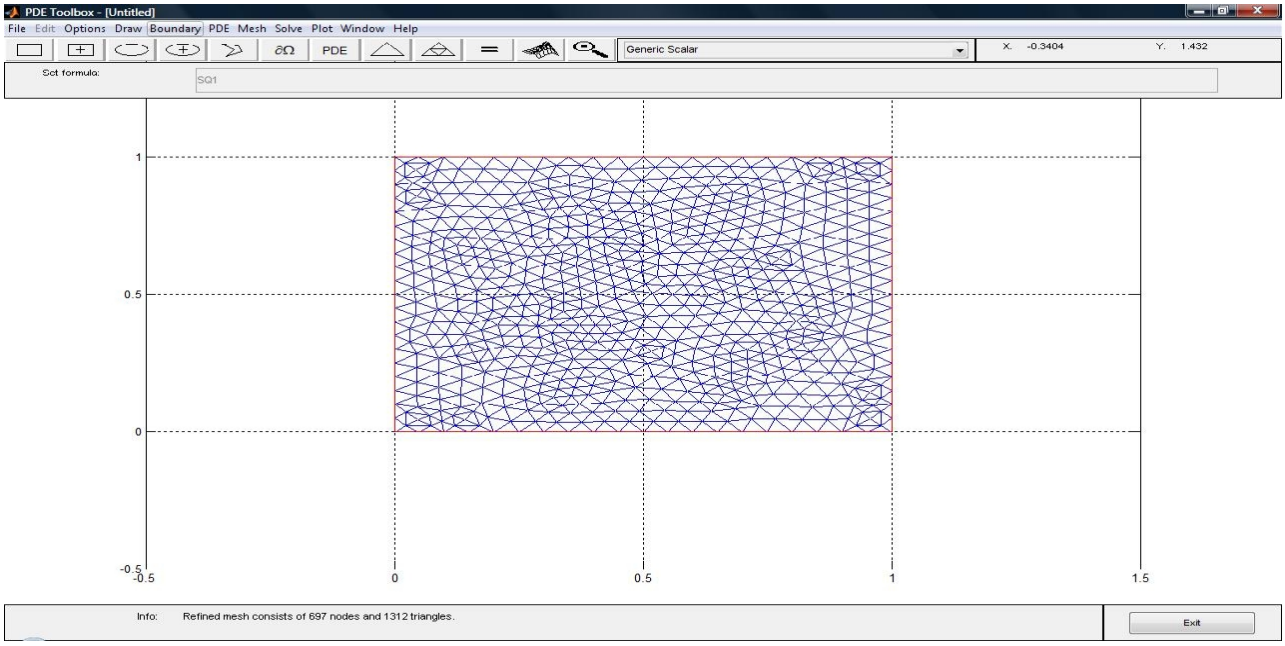

Fig 3: Griglia Generata

5. **Definire l'equazione del problema:**
   - Entrare nel menù **PDE** e selezionare in sequenza
     - **PDE Mode.**
     - **PDE Specification:** selezionare il tipo di problema, in questo caso ellittico. Poi inserire i coefficienti del problema: $c=1\,;\, a=0\,;\, f=0$.
6. **Generare la soluzione del problema:**
   - Entrare nel menù **Solve** e selezionare:
     - **Solve PDE** (per maggiori dettagli fare riferimento al paragrafo 1.3 nella sezione relativa alla funzione *assempde*).
7. **Visualizzare la soluzione del problema:**
   - Andare nel menù **Plot** e selezionare:
     - **Parameters:** qui è possibile modificare il diagramma della soluzione che il software di default restituisce. Ad esempio selezionando
       - **Height** si ottiene il diagramma in 3D della soluzione,
       - **Show Mesh** viene diagrammata anche la griglia di partenza,
       - **Colormap** permette di cambiare i colori del diagramma: scegliere jet.
   - Si ottiene così il diagramma 3D mostrato in Fig. 4.

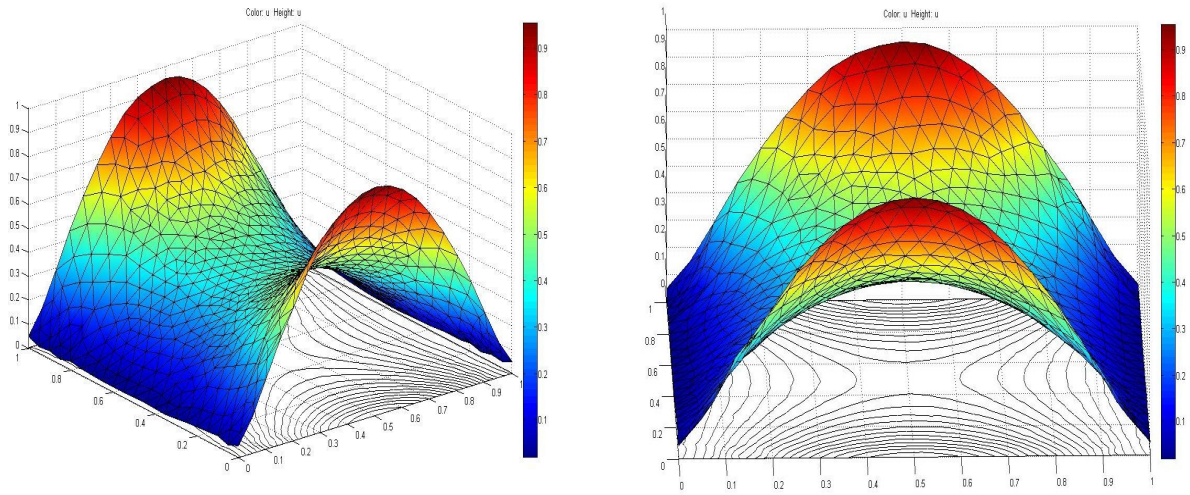

Fig. 4: Due Viste della Soluzione Mediante GUI

La GUI mette a disposizione dell'utente altri utili strumenti:

- Mediante la barra **Set Formula** è possibile combinare tra loro più domini creati in precedenza, per ottenerne uno nuovo.
- In **Options** sono presenti le seguenti funzioni:
  - **Grid Spacing** permette di definire quanto la griglia nell'area di disegno deve essere fitta.
  - **Snap** impone al dominio che si sta disegnando ad aderire alla griglia (ad esempio i vertici di rettangoli e poligoni devono essere posti sul nodo più vicino della griglia);
  - **Application** apre un menù a tendina dove vengono indicati alcuni problemi fisici legati alla risoluzione di EDP, selezionando il tipo di problema da dover risolvere alcune impostazioni vengono fatte in automatico (tipo impostare i coefficienti della EDP).
- Nel menù **Mesh** è possibile intervenire sulla griglia generata all'interno del dominio:
  - **Jiggle Mesh** modifica la griglia creata, ottenendo triangoli il più possibile equilateri (questo per evitare possibili problemi di malcondizionamento).
  - **Parameters** permette di impostare alcuni parametri opzionali per la costruzione della griglia, se non specificati il codice usa valori di default. Impostando:
    - *Maximum Edge Size* si può indicare la lunghezza massima dei lati dei triangoli che compongono la griglia.
    - *Mesh Growth Rate* si indica la differenza di dimensione tra i triangoli più piccoli e quelli più grandi che compongono la griglia, questo è un numero reale compreso tra 1 e 2.
    - *Jiggle Mode* permette di scegliere se l'operazione di jiggling deve essere fatta una volta sola, oppure in modo iterativo fino alla massima precisione (è possibile

specificare il numero massimo di iterazioni).

- *Refinement Method* permette di scegliere il criterio con cui raffinare la griglia generata con la funzione *Refine Mesh: Regular,* tutta la griglia viene raffinata in maniera uniforme, *Longest,* vengono raffinate solo quelle regioni dove sono presenti i triangoli con i lati più lunghi.

- Nel menù **Solve** è possibile selezionare le seguenti opzioni:
  - **Parameters:** qui è possibile scegliere tra due solver alternativi di EDP di tipo ellittico. Questa scelta è opzionale, di default la funzione usata è *assempde* (si veda il paragrafo 1.3, per la descrizione di questa funzione).
    - *Adaptive Mode,* indica il solver adattativo: in automatico il software ad ogni iterazione, raffina la griglia solo dove è necessario. Per questa funzione è possibile fissare:
      - *Maximum Number of Triangles* ovvero il numero massimo di triangoli che vengono aggiunti ad ogni iterazione.
      - *Maximum Number of Refinements* ovvero il numero massimo di iterazioni.
      - Infine il criterio in base al quale scegliere dove andare a raffinare: scegliendo *Worst Triangles* il software stima l'errore commesso su ogni triangolo, paragonando questi valori raffina solo le regioni che presentano degli errori maggiori (bisogna inserire come ulteriore parametro un numero compreso tra 0 e 1, più questo è vicino ad 1 più il software diventa selettivo); scegliendo *Relative Tolerance* vengono raffinati solo quei triangoli dove l'errore relativo stimato è maggiore della tolleranza indicata; scegliendo *User defined function* si indica al software di usare una funzione creata dall'utente.
    - *Use Non Linear Solver*: va selezionata quando l'equazione da risolvere è non lineare. I parametri da definire sono: la tolleranza per l'errore, una soluzione iniziale per il solver non lineare (questo si basa sul metodo di Newton la cui convergenza dipende dalla scelta del punto iniziale).

- Nel menù **Plot** è possibile selezionare:
  - **Parameters**: da qui si possono modificare molti parametri relativi al diagramma della soluzione.
    - *contour* vengono mostrare le curve di livello della soluzione.
    - *arrows* viene mostrato l'andamento del campo mediante delle frecce.
    - *deformed mesh* restituisce una rappresentazione deformata del dominio.
    - *plot in x-y grid* disegna il grafico su una griglia rettangolare.

Prima di procedere con la risoluzione del problema test mediante CL, è importante considerare due fattori.

1. Le due modalità che Matlab mette a disposizione dell'utente per risolvere le EDP sono tra loro collegate e possono interagire. Ad esempio:

   - Mediante le opzioni **Export** nei vari menù della GUI è possibile esportare nell'area di lavoro di Matlab le informazioni riguardanti il dominio, le condizioni al contorno, i coefficienti del problema, le soluzioni create con l'ausilio della GUI.

   - Tramite i comandi **pdecirc, pdeellip, pdepoly, pderect,** è possibile generare all'interno della GUI i domini indicati.

2. Risolvere i problemi mediante la GUI è immediato, ma questo strumento possiede dei limiti: non è possibile considerare il caso di sistemi di EDP, non è possibile lavorare su domini che non possono essere espressi come combinazione di alcune figure geometriche. In questi casi è necessario utilizzare la CL.

➢ Soluzione mediante CL

Per la risoluzione mediante CL del problema test di Laplace si riporta il seguente script file:

```
%fase1: creazione dominio del problema
%Creazione Geometry Description Matrix:
gd=[3;4;0;1;1;0;0;0;1;1];
%Passaggio alla Decomposed Geometry Matrix:
dl=decsg(gd);
%fase2: creazione griglia triangolare all'interno del dominio:
[p,e,t]=initmesh(dl);
[p,e,t]=refinemesh(dl,p,e,t,'regular'); %la griglia viene raffinata
pdemesh(p,e,t) %diagramma del dominio e della griglia
pause
%fase3: condizioni al contorno:
% le condizioni al contorno vengono indicate con la seguente matrice:
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   9 1 9 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   115 48 115 48;
   105 32 105 32;
   110 32 110 32;
   40 32 40 32;
   112 32 112 32;
   105 32 105 32;
   42 32 42 32;
   120 32 120 32;
   41 32 41 32;]
[Q,G,H,R]=assemb(b,p,e);
%fase4: l'equazione viene definita e risolta:
c=1; a=0; f=0;
U=assempde(b,p,e,t,c,a,f);
pause
%fase5: diagramma della soluzione
pdesurf(p,t,U)
pause
% per un diagramma più completo:
pdeplot(p,e,t,'xydata',U,'zdata',U,'mesh','on','flowstyle','arrow','contour','on
```

')

Il grafico della soluzione è riportato in Fig. 5.

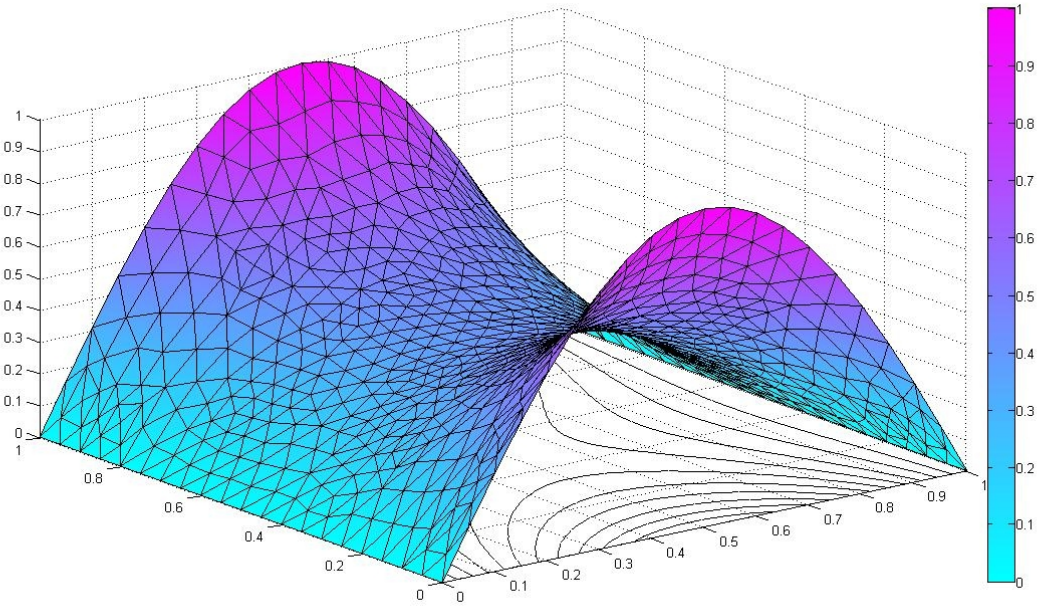

Fig. 5: Soluzione Mediante CL

Usando un algoritmo iterativo, è possibile fare uno studio di convergenza per individuare una prima stima dell'ordine di convergenza del metodo numerico usato in precedenza.

Questa analisi è stata realizzata usando due algoritmi: il primo discretizza il dominio del problema con griglie non strutturate (griglie triangolari), generate mediante la funzione **initmesh** (per maggiori informazioni fare riferimento al paragrafo 1.3), il secondo usa griglie strutturate (griglie quadrate), generate mediante la funzione **poimesh** (per maggiori informazioni fare riferimento al paragrafo 1.3).

Ad ogni iterazione viene dimezzato il passo di discretizzazione del dominio creato:

- Il primo algoritmo ad ogni iterazione dimezza il parametro di input opzionale Hmax della funzione *initmesh*, generando triangoli equilateri, avendo settato il parametro hgrad=1.01, dove *hgrad* è un parametro di input opzionale che indica il rapporto $\frac{Hmax}{Hmin}$ ovvero il rapporto tra il lato più lungo ed il lato più corto dei triangoli di griglia, esso è un numero reale compreso tra 1 e 2 .

- Il secondo algoritmo ad ogni iterazione raddoppia i parametri di input *nx* e *ny* della funzione *poimesh*, questi indicano rispettivamente il numero di punti della griglia nella direzione x e y.

Entrambi gli algoritmi ad ogni iterazione confrontano i risultati ottenuti con i risultati esatti che si ottengono dalla (6), ricavando l'*errore globale* commesso in norma L^2. Si riportano di seguito gli script files creati.

Algoritmo che implementa la funzione *initmesh*:

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   9 1 9 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   115 48 115 48;
   105 32 105 32;
   110 32 110 32;
   40 32 40 32;
   112 32 112 32;
   105 32 105 32;
   42 32 42 32;
   120 32 120 32;
   41 32 41 32];
c=1; a=0; f=0;
fun=inline('sech(0.5.*pi).*sin(pi.*x).*cosh((pi).*(y-0.5))');
i=1; h(i)=0.1;
[p,e,t]=initmesh(dl,'hmax',h(i),'hgrad',1.01);
U=assempde(b,p,e,t,c,a,f);
x=p(1,:); y=p(2,:);
Z=fun(x,y); Z=Z';
Errore(i)=norm(Z-U)/norm(Z);
while i<5
    i=i+1;
    h(i)=h(i-1)/2;
    [p,e,t]=initmesh(dl,'hmax',h(i),'hgrad',1.01);
    U=assempde(b,p,e,t,c,a,f);
    x=p(1,:); y=p(2,:);
    Z=fun(x,y); Z=Z';
    Errore(i)=norm(Z-U)/norm(Z);
end
%diagramma dell'andamento dell'errore
x1=[0.1 0.01 0.001 0.0001 0.00001]; y1=x1; %retta di pendenza 1
x2=x1; y2=x2.^2; %retta di pendenza 2
loglog(h,Errore,'*-',x1,y1,'--',x2,y2,'-.')
grid
```

Algoritmo che implementa la funzione *poimesh*:

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   9 1 9 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   115 48 115 48;
   105 32 105 32;
   110 32 110 32;
```

```
    40 32 40 32;
    112 32 112 32;
    105 32 105 32;
    42 32 42 32;
    120 32 120 32;
    41 32 41 32];
c=1; a=0; f=0;
fun=inline('sech(0.5.*pi).*sin(pi.*x).*cosh((pi).*(y-0.5))');
i=1;N(i)=10;h(i)=1/N(i);
[p,e,t]=poimesh(dl,N(i));
U=assempde(b,p,e,t,c,a,f);
x=p(1,:); y=p(2,:);
Z=fun(x,y); Z=Z';
Errore(i)=norm(Z-U)/norm(Z);
while i<7
    i=i+1;
    N(i)=N(i-1)*2; h(i)=1/N(i);
    [p,e,t]=poimesh(dl,N(i));
    U=assempde(b,p,e,t,c,a,f);
    x=p(1,:); y=p(2,:);
    Z=fun(x,y); Z=Z';
    Errore(i)=norm(Z-U)/norm(Z);
end
%diagramma dell'andamento dell'errore
x1=[0.1 0.01 0.001 0.0001 0.00001]; y1=x1; %retta di pendenza 1
x2=x1; y2=x2.^2; %retta di pendenza 2
loglog(h,Errore,'*-',x1,y1,'--',x2,y2,'-.')
grid
```

In Fig. 6. sono mostrati i risultati ottenuti, ovvero i due andamenti dell'*errore globale*. Da questa stima a posteriori si ricava che sia usando griglie strutturate che griglie non strutturate l'ordine di convergenza del metodo numerico è prossimo a 2.

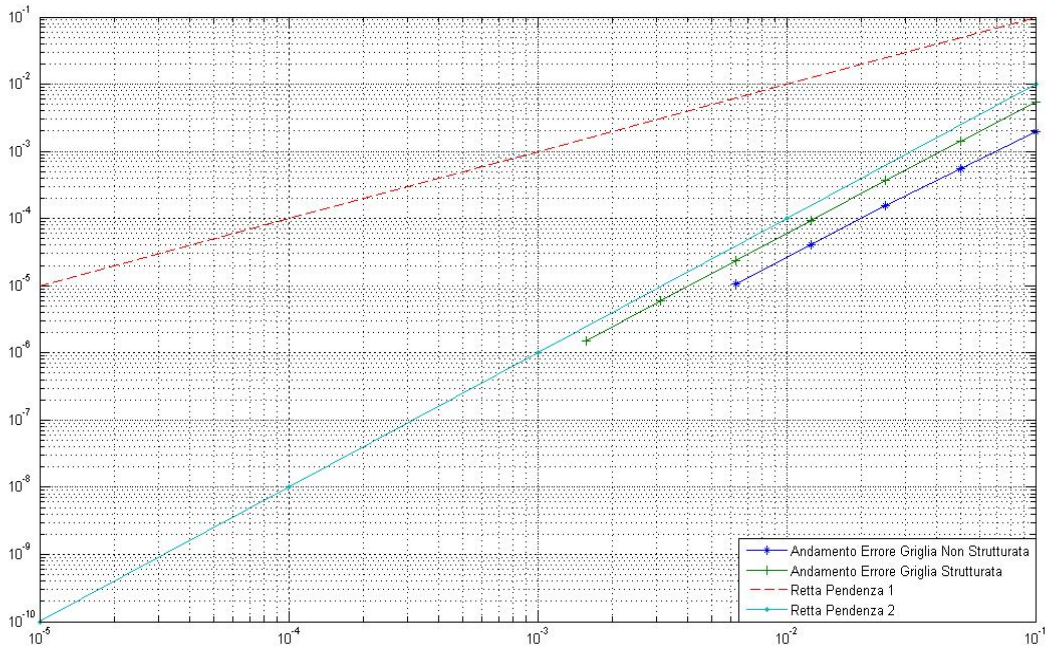

Fig 6: Andamento Errore Globale

Anche lavorando mediante CL è possibile usare la funzione adattativa per risolvere problemi

ellittici: la funzione **adaptmesh** in automatico genera la migliore griglia non strutturata per risolvere il problema dell'utente (per una descrizione dettagliata della funzione fare riferimento al paragrafo 1.3). Tra le varie opzioni che l'utente può impostare, vi è quella di scegliere una misura del massimo *errore relativo locale* consentito. Con un algoritmo iterativo è possibile verificare come varia la dimensione del sistema da risolvere (quindi il numero dei punti di griglia) in funzione del massimo errore relativo locale. L'algoritmo usato per questa analisi si arresta nel momento in cui la funzione adattativa esegue il numero massimo di iterazioni senza raggiungere l'accuratezza richiesta. Si consideri che per default il numero massimo di iterazioni è pari a dieci e che alla decima iterazione vengono creati 28226 triangoli: questo dato, verificato sperimentalmente, dipende dal dominio del problema.

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   9 1 9 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   115 48 115 48;
   105 32 105 32;
   110 32 110 32;
   40 32 40 32;
   112 32 112 32;
   105 32 105 32;
   42 32 42 32;
   120 32 120 32;
   41 32 41 32];
c=1; a=0; f=0;
Err_locale(1)=10^(-1); i=1; k=0;
while k==0
    [U,p,e,t]=adaptmesh(dl,b,c,a,f,'tripick','pdeadgsc','par',Err_locale(i));
    num_tri=size(t,2);
    dim_sist(i)=size(p,2);
    i=i+1;
    Err_locale(i)=Err_locale(i-1)/10;
    if num_tri==28226
        k=1;
    else
        k=0;
    end
end
```

I risultati ottenuti sono i seguenti:

Errore Relativo Locale	1,00E-001	1,00E-002	1,00E-003	1,00E-004
Dimensione Del Sistema	185	205	1511	14295

1.2.2. Soluzione problema di Poisson in 2D.

- Risoluzione mediante CL

Di questo problema si tralascia la risoluzione mediante la GUI, poiché non aggiungerebbe niente di nuovo a quanto già fatto per il problema di Laplace. Si consideri quindi il seguente script file per la risoluzione mediante CL:

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
[p,e,t]=initmesh(dl,'hmax',0.05);
pdeplot(p,e,t)
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   48 48 48 48];
[Q,G,H,R]=assemb(b,p,e);
c=[0.2525;0.2475;0.2475;0.2525]; a=0; f=1;
U=assempde(b,p,e,t,c,a,f);
pdeplot(p,e,t,'xydata',U,'zdata',U,'mesh','on','flowstyle','arrow','contour','on
')
```

La soluzione in formato grafico del problema di Poisson è riportata in Fig. 7.

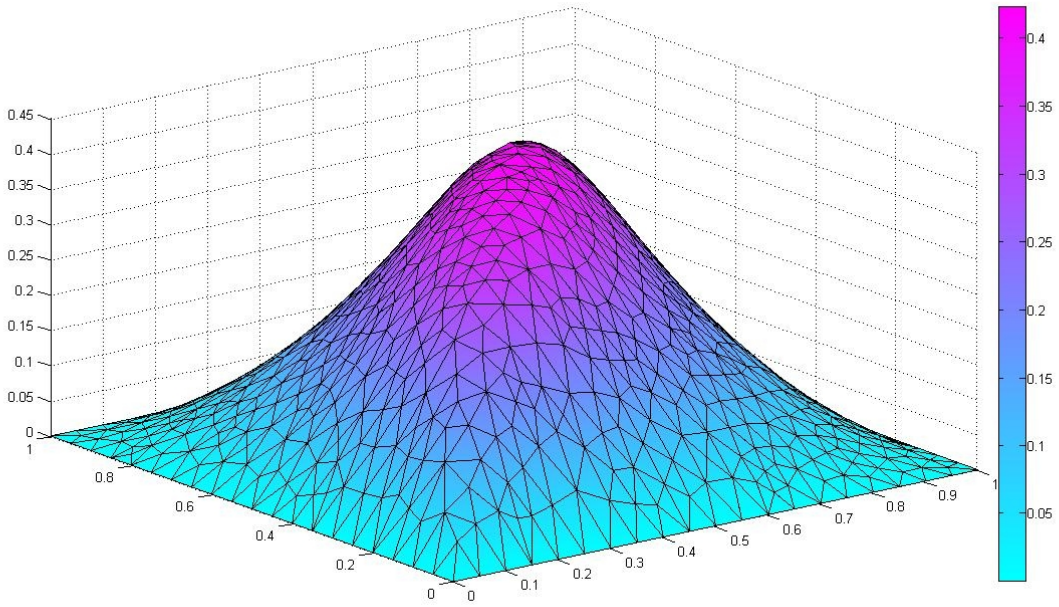

Fig 7: Soluzione Problema di Poisson

Anche per questo problema è possibile fare una prima stima dell'ordine di convergenza del metodo numerico usato. Di questo problema però non si possiede l'espressione analitica della soluzione esatta, quindi per calcolare una prima stima dell'ordine di convergenza, è stata considerata la seguente formula indicata in [MCNI]:

$$\check{R}=\frac{\|y_n^{h/4\to h}-y_n^{h}\|}{\|y_n^{h/4\to h}-y_n^{h/2\to h}\|}=2^p+1 \qquad (12)$$

ovvero: $p \approx \log_2(\check{R} - 1)$ (13)

dove:

- p è una prima stima dell'ordine di convergenza del metodo numerico.
- y_n^h soluzione numerica ottenuta con passo di discretizzazione pari ad h.
- $y_n^{h/2 \to h}$ soluzione numerica ottenuta con passo di discretizzazione pari ad h/2, valutata nei punti della prima griglia.
- $y_n^{h/4 \to h}$ soluzione numerica ottenuta con passo di discretizzazione pari ad h/4, valutata nei punti della prima griglia.

Anche per questa analisi sono stati utilizzati due algoritmi diversi: entrambi permettono di individuare p mediante le relazioni (12) e (13), ma il primo utilizza griglie non strutturate, mentre il secondo utilizza griglie strutturare.

Script file che implementa *initmesh*:

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   48 48 48 48];
c=[0.2525;0.2475;0.2475;0.2525]; a=0; f=1;
i=1; h=0.0063;
[p,e,t]=initmesh(dl,'hmax',h);
x3=p(1,:); y3=p(2,:);
U3=assempde(b,p,e,t,c,a,f);
[p,e,t]=initmesh(dl,'hmax',h*2);
x2=p(1,:); y2=p(2,:);
U2=assempde(b,p,e,t,c,a,f);
[p,e,t]=initmesh(dl,'hmax',h*4);
x1=p(1,:); y1=p(2,:);
U1=assempde(b,p,e,t,c,a,f); U1=U1';
U2=griddata(x2,y2,U2,x1,y1);
U3=griddata(x3,y3,U3,x1,y1);
R=norm(U3-U1)/norm(U3-U2);
p=log(R-1)/log(2)
```

Questo restituisce: $\boldsymbol{p \approx 1.1933}$, si consideri che questo valore è stato ottenuto imponendo $h/4 = 6{,}30\text{E-}003$, ovvero un valore di Hmax per griglie non strutturate, per il quale è possibile ottenere una soluzione numerica in un tempo accettabile con il calcolatore a disposizione.

Script file che implementa *poimesh:*

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[1 1 1 1;
```

```
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   48 48 48 48];
c=[0.2525;0.2475;0.2475;0.2525]; a=0; f=1;
N=100;
[p,e,t]=poimesh(dl,N);
x=p(1,:); y=p(2,:);
U1=assempde(b,p,e,t,c,a,f); U1=U1';
[p,e,t]=poimesh(dl,N*2);
xp=p(1,:); yp=p(2,:);
U2=assempde(b,p,e,t,c,a,f);
U2=griddata(xp,yp,U2,x,y);
[p,e,t]=poimesh(dl,N*4);
xp=p(1,:); yp=p(2,:);
U3=assempde(b,p,e,t,c,a,f);
U3=griddata(xp,yp,U3,x,y);
R=norm(U3-U1)/norm(U3-U2);
p=log(R-1)/log(2)
```

Questo restituisce $\boldsymbol{p \approx 1.7439}$, si consideri che questo valore è stato ottenuto impostando $N*4=400$ ovvero $h/4=2{,}50\text{E-}003$: questo rappresenta un valore di Hmax per griglie strutturate per il quale è possibile ottenere una soluzione numerica in un tempo accettabile con il calcolatore a disposizione.

I due risultati ottenuti differiscono tra loro poiché la stima ottenuta con la relazione (12) tende a quella reale per Hmax → 0. Quindi anche in questo caso la stima a posteriori ci restituisce un ordine di convergenza prossimo a 2.

1.2.3. Soluzione problema 1D. Il problema monodimensionale descritto in precedenza, può essere visto anche come un caso particolare di problema di Poisson:

$$-\nabla \cdot [\boldsymbol{K} \cdot \nabla u] = f \quad con\ (x, y) \in]0,1[\times]0,1[\tag{14}$$

dove $\boldsymbol{K} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$; $f = -\frac{1}{x}$;

con condizioni al contorno alla Dirichlet: (15)

$$u(0, y) = u(1, y) = u(x, 0) = u(x, 1) = 0 \tag{16}$$

Si ottiene la curva incognita considerando l'intersezione tra la superficie della soluzione del problema bidimensionale ed un piano $y = cost$, con $0 < y < 1$. Questa operazione di estrapolazione dei dati viene realizzata senza interpolare i risultati numerici ottenuti, ma risolvendo il problema su una *griglia strutturata* ottenuta mediante la funzione *poimesh*.

Per la risoluzione numerica del problema monodimensionale, si consideri il seguente script file:

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
[p,e,t]=poimesh(dl,20);
b=[1 1 1 1;
```

```
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   48 48 48 48];
c=[1;0;0;0]; a=0; f='1./x';
U=assempde(b,p,e,t,c,a,f);
pdeplot(p,e,t,'xydata',U,'zdata',U,'mesh','on','flowstyle','arrow','contour','on
')
pause
y=p(2,:); k=find(y==0.5);
Up=U(k);xp=p(1,k);
plot(xp,Up)
grid
```

La soluzione in formato grafico è rappresentata in Fig. 8.

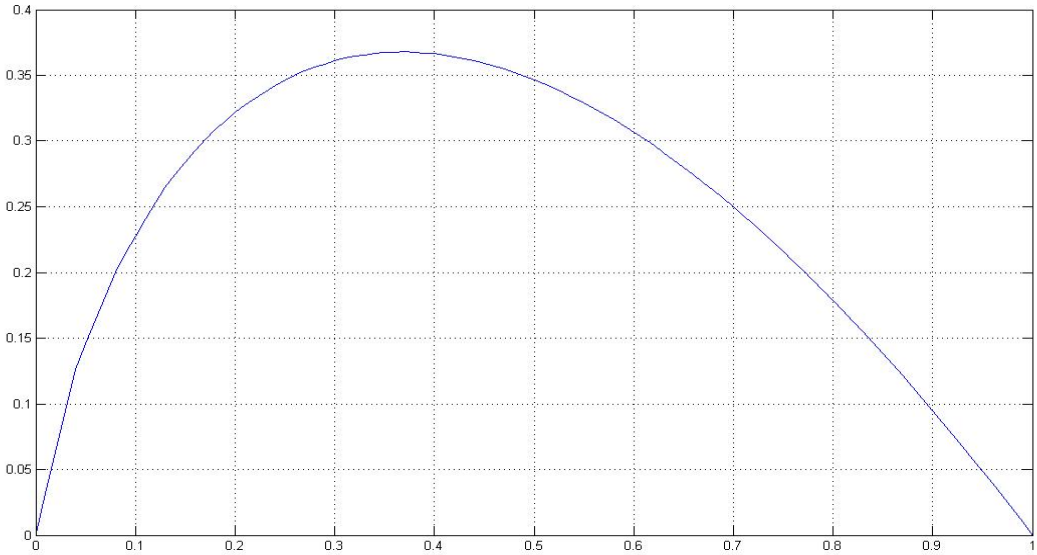

Fig 8: Soluzione Problema 1D

Anche per questo problema, poiché si possiede la soluzione analitica, è possibile determinare una stima dell'ordine di convergenza del metodo numerico usato.

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   1 1 1 1;
   48 48 48 48;
   48 48 48 48;
   49 49 49 49;
   48 48 48 48];
```

```
c=[1;0;0;0]; a=0; f='1./x';
fun=inline('-x.*log(x)');
N(1)=10; h(1)=1/N(1); i=1;
[p,e,t]=poimesh(dl,N);
U=assempde(b,p,e,t,c,a,f);
y=p(2,:); k=find(y==0.5);
U=U(k); U(1)=[]; U=U';
x=p(1,k); x(1)=[]; Z=fun(x);
Errore(i)=norm(Z-U)/norm(Z);
while i<7
    i=i+1;
    N(i)=2*N(i-1);
    h(i)=1/N(i);
    [p,e,t]=poimesh(dl,N(i));
    U=assempde(b,p,e,t,c,a,f);
    y=p(2,:); k=find(y==0.5);
    U=U(k); U(1)=[]; U=U';
    x=p(1,k); x(1)=[]; Z=fun(x);
    Errore(i)=norm(Z-U)/norm(Z);
end
x1=[0.1 0.01 0.001 0.0001]; y1=x1; %retta di pendenza 1
x2=x1; y2=x1.^2; %retta di pendenza 2
loglog(h,Errore,'-*',x1,y1,'--',x2,y2,'-.')
grid
```

L'andamento dell'errore globale per questo problema è mostrato in Fig. 9

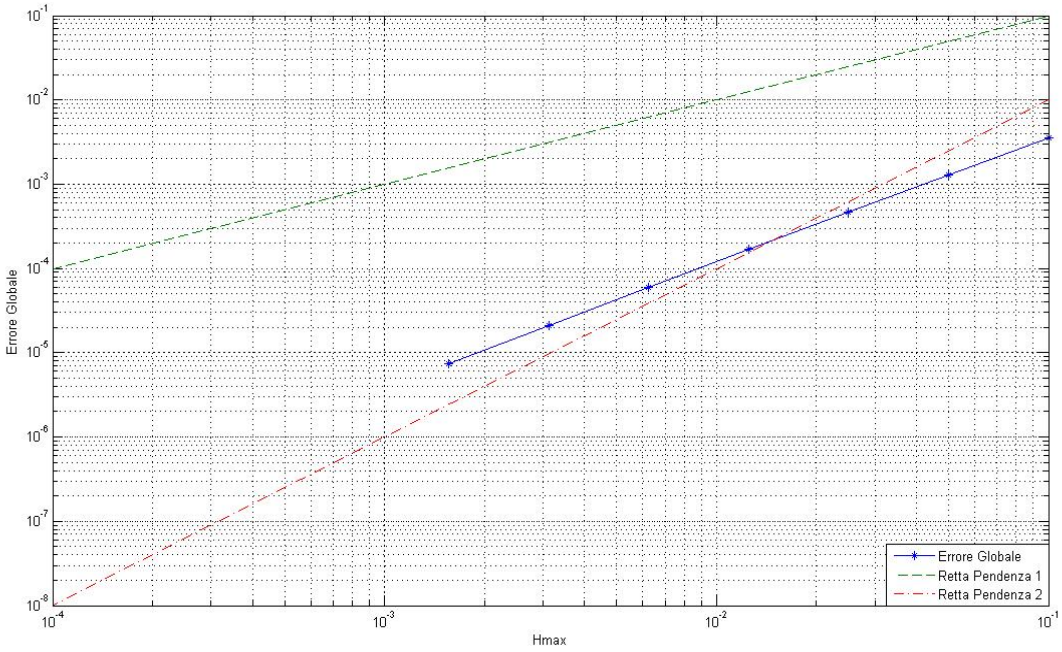

Fig 9: Andamento Errore Globale Problema 1D

Analizzando il diagramma in Fig. 9 si ricava che anche in questo caso l'ordine di convergenza è compreso tra 1 e 2.

1.3. Descrizione delle principali funzioni del PDE toolbox di Matlab.

- **decsg:** funzione che genera il dominio del problema.
  `dl=decsg(gd)`

```
[dl,bt]=decsg(gd)
[dl,bt]=decsg(gd,sf,ns)
```

- Parametri di Input
  - *gd*: matrice di reali, detta *Geometry Description Matrix* viene usata per indicare il dominio del problema.
  - *sf*: stringa di caratteri, è un parametro di input opzionale, da inserire se il dominio è ottenuto come combinazione di più figure geometriche. Inserendo in modo opportuno gli operatori '*', '+', '-' indichiamo in che modo combinare la varie figure: questi indicano rispettivamente le operazioni di intersezione, unione e differenza.
  - *ns*: matrice di reali, è un parametro di input opzionale, da inserire se il dominio è ottenuto come combinazione di più figure geometriche. Il numero di colonne dipende dal numero di figure geometriche usate per la creazione del dominio: ogni colonna indica il nome della figura corrispondente, questo è espresso indicando in sequenza i codici ASCII dei caratteri usati.
- Parametri di Output:
  - *dl*: matrice di reali, detta *Decomposed Geometry Matrix* contiene la descrizione del dominio nella forma parametrica.
  - *bt*: matrice Booleana, è un parametro di output opzionale. Viene così costruita: ogni colonna in bt si riferisce alla colonna in dl con lo stesso indice, ogni riga in bt si riferisce al sotto dominio con lo stesso indice.
- Un approfondimento sulla Geometry Description Matrix e sulla Decomposed Geometry Matrix è presente in [PDE_UG].
  Costruire la Geometry Description Matrix è relativamente semplice: essa presenta una colonna per ogni figura che compone il dominio. Le righe sono così compilate: la prima riga presenta uno dei seguenti numeri: 1 in caso di circonferenza, 2 in caso di polinomio, 3 in caso di rettangolo, 4 in caso di ellisse. Nel caso 1 le successive righe in sequenza indicano: coordinate x e y del centro e lunghezza del raggio; nel caso 2: numero di lati, coordinate x dei vertici e poi coordinate y dei vertici; caso 3: la seconda riga contiene 4, a seguire le coordinate x e y dei vertici del rettangolo; caso 4: coordinate x e y del centro dell'ellisse poi le i semiassi dell'ellisse, l'ultima riga contiene l'angolo di rotazione dell'ellisse. Da questi dati la funzione ricava la Decomposed Geometry Matrix, essa è così costruita: presenta tante colonne quanti sono i segmenti che descrivono il bordo del dominio (minimo 4). La prima riga contiene i valori che assume all'inizio di ogni segmento il parametro che viene utilizzato dal codice per la descrizione parametrica del contorno, la seconda riga contiene i valori che il parametro assume nei punti finali del segmento. Nella terza riga è presente il numero del sotto dominio che si trova a sinistra del segmento, percorrendolo nel verso indotto dalla parametrizzazione. Nella quarta riga il numero del sotto dominio che si trova a destra. Volendo se si possiede già la descrizione parametrica del contorno del dominio si può costruire direttamente la matrice dl, usando lo schema visto prima.

- **initmesh:** funzione che genera la griglia non strutturata che discretizza il dominio del problema.

```
[p,e,t]=initmesh(dl)
[p,e,t]=initmesh(dl,'PropertyName',PropertyValue,...)
```

- Parametri di Input:
  - *dl*: matrice di reali che indica la Decomposed Geometry del dominio.
  - '*PropertyName*': stringa di caratteri, è un parametro opzionale di input da inserire se si vuole modificare una o più opzioni dell'algoritmo.
  - *PropertyValue*: può essere una stringa di caratteri o un numero reale, è un parametro

opzionale di input, in cui si indica come modificare l'opzione indicata in 'PropertyName'.
Le opzioni che si possono modificare sono indicate nella seguente tabella:

PropertyName	PropertyValue	Valore di Default	Descrizione
`Hmax`	Reale positivo	Stimato	Lunghezza massima dei lati
`Hgrad`	Reale positivo compreso tra 1 e 2	`1.3`	Rapporto massimo tra il lato più lungo e quello più corto
`Box`	`on|off`	`off`	Se attivo obbliga a rispettare il bounding box
`Init`	`on|off`	`off`	Se attivo viene mostrata la triangolazione del contorno del dominio
`Jiggle`	`off|mean|min`	`mean`	Attiva la funzione di jiggling
`JiggleIter`	numeric	10	Numero massimo di iterazioni che la funzione Jiggle può realizzare

- Parametri di Output:
  - *p:* matrice di reali, chiamata *Point Matrix.* Essa possiede due righe: nella prima sono memorizzate le coordinate x dei nodi della griglia, nella seconda invece le coordinate y.
  - *e*: matrice di reali, chiamata *Edge Matrix.* Essa presente una numero di colonne pari al numero dei lati dei triangoli. Mentre è composta sempre da sette righe: in queste sono inserite le informazioni relative alla costruzione ed alla memorizzazione dei lati dei triangoli.
  - *t*: matrice di reali, chiamata *Triangle Matrix*. Essa è composta da un numero di colonne pari al numero di triangoli della griglia: mediante gli indici delle colonne i triangoli vengono numerati ed etichettati. Mentre il numero delle righe è sempre quattro: le prime tre righe contengono gli indici dei dei punti che rappresentano i vertici di ogni triangolo (fa riferimento sempre alla matrice p), la quarta riga indica a quale sotto dominio il triangolo appartiene.
- La funzione genera una griglia non strutturata mediante l'algoritmo di triangolazione di *Delaunay* in combinazione con la *tecnica di avanzamento del fronte*. Un approfondimento sull'algoritmo di *Delaunay* è contenuto in [MCNI].
  Una triangolazione di n punti si dice di Delaunay se il cerchio circoscritto di ciascun triangolo non contiene alcun nodo al suo interno. Essa gode di queste proprietà: è unica, tra tutte le triangolazioni questa massimizza il minimo angolo dei triangoli della griglia, l'insieme formato dall'unione dei triangoli è la più piccola figura convessa che racchiude il set di punti dato. Per generare l'insieme dei nodi della griglia viene usata la tecnica di avanzamento del fronte: il fronte è una struttura dati che contiene la lista dei lati che definiscono la frontiera tra la porzione di dominio già triangolata e quella ancora da

triangolare. Il codice parte dal contorno del dominio, il fronte viene inizializzato con i lati del contorno, sempre lungo il contorno vengono poi generati i primi punti di griglia usando il parametro Hmax, che può essere fornito dall'utente o stimato dal codice. Il codice poi si sposta verso l'interno fino a coprire l'intero dominio, aggiornando i dati del fronte in base ai punti ed ai lati creati. Sui triangoli creati il codice effettua un duplice controllo: devono soddisfare la regola di Delaunay e non devono avere lati di lunghezza superiore ad Hmax. Quando questa seconda condizione non è verificata il codice interviene andando ad inserire sul lato troppo lungo un nuovo punto di griglia e generando quindi anche nuovi triangoli. Il codice si ferma quando l'intero dominio è stato triangolato in modo corretto.

- **poimesh:** funzione di Matlab che permette di creare griglie strutturate.
  `[p,e,t]=poimesh(dl,nx,xy)`
  - Parametri di Input:
    - *dl:* matrice di reali, indica la Decomposed Geometry del dominio.
    - *nx:* intero positivo, indica il numero di punti di griglia in direzione x.
    - *ny:* intero positivo, indica il numero di punti di griglia in direzione y.
  - Parametri di Output:
    - *p,e,t:* matrici di reali, memorizzano la griglia creata.

- **refinemesh**: raffina una griglia precedentemente creata con *initmesh.*

```
[p1,e1,t1]=refinemesh(dl,p,e,t)
[p1,e1,t1]=refinemesh(dl,p,e,t,'regular')
[p1,e1,t1]=refinemesh(dl,p,e,t,'longest')
[p1,e1,t1,u1]=refinemesh(dl,p,e,t,u)
```

  - Parametri di Input:
    - *dl,p,e,t:* matrici di reali, sono le stesse matrici usate dal codice *initmesh*.
    - '*regular'*: stringa di caratteri, è un parametro di input opzionale, con questo l'utente indica di voler un raffinare tutto il dominio, dimezzando Hmax.
    - '*longest'*: stringa di caratteri, è un parametro di input opzionale, con questo l'utente indica di voler raffinare solo i triangoli con i lati più lunghi.
    - *u:* vettore di numeri reali, è un parametro di input opzionale, contiene la soluzione del problema nei punti della griglia di partenza.
  - Parametri di Output:
    - *p1,e1,t1,:* matrici di reali, indicano la nuova griglia ottenuta.
    - *u1:* vettore di numeri reali, è un parametro di output opzionale, va richiesto solo se indicato il vettore u in input. Questo vettore contiene la soluzione nei nuovi punti di griglia, questi valori vengono ricavati interpolando linearmente la soluzione di partenza.

- **assemb:** genera le condizioni al contorno del problema

```
[Q,G,H,R]=assemb(b,p,e)
[Q,G,H,R]=assemb(b,p,e,u0,time)
```

  - Parametri di Input:
    - *b,p,e:* matrici di reali indicano le condizioni al contorno e i punti di griglia sul contorno.
    - *u0*: vettore di reali, parametro opzionale di input che va inserito solo in caso di problema iperbolico o parabolico, contiene la soluzione iniziale del problema.
    - time: vettore di reali, parametro opzionale di input, va inserito solo in caso di problema iperbolico o parabolico, contiene gli istanti in cui determinare la soluzione.

- Parametri di Output:
  - *Q*: vettore di reali, contiene il valore della funzione q nei nodi sul contorno del dominio.
  - *G:* vettore di reali, contiene il valore della funzione g nei nodi sul contorno del dominio.
  - *H:* vettore di reali, contiene il valore della funzione h nei nodi sul contorno del dominio.
  - *R:* vettore di reali, contiene il valore della funzione r nei nodi sul contorno del dominio.

- La matrice b va a specificare le condizioni al contorno del problema, la sua costruzione va fatta manualmente dall'utente se si vuole lavorare mediante CL, come indicato in [PDE_UG]. La matrice presenta tante colonne quanti sono i segmenti che compongono il contorno del dominio, il numero di righe invece varia a seconda delle condizioni del problema. Le righe vengono compilate secondo questo schema: la prima riga contiene il numero di equazioni che compongono il sistema di EDP (per il caso scalare indicare 1), la seconda riga indica il tipo di condizioni da imporre: 1 per Dirichlet, 0 per Robin o Neumann. Le quattro righe successive indicano il numero di caratteri usati per indicare rispettivamente q, g, h, r. Da qui in poi le righe contengono i codici ASCII dei caratteri usati per scrivere in linguaggio Matlab le funzioni q, g, h, r.

- **assempde:** funzione che risolve numericamente le EDP, in particolar modo i problemi ellittici (per problemi iperbolici ed parabolici ci sono anche funzioni apposite):

```
u=assempde(b,p,e,t,c,a,f)
u=assempde(b,p,e,t,c,a,f,u0,time)
[K,F]=assempde(b,p,e,t,c,a,f)
[K,F]=assempde(b,p,e,t,c,a,f,u0,time)
[K,F,B,ud]=assempde(b,p,e,t,c,a,f)
[K,F,B,ud]=assempde(b,p,e,t,c,a,f,u0,time)
[K,M,F,Q,G,H,R]=assempde(b,p,e,t,c,a,f)
[K,M,F,Q,G,H,R]=assempde(b,p,e,t,c,a,f,u0,time)
```

  - Parametri di Input:
    - *b:* matrice di reali che specifica le condizioni al contorno.
    - *p,e,t:* matrici di reali, fare riferimento ad *initmesh* per maggiori dettagli.
    - *c,a,f:* coefficienti del problema.
    - *u0:* vettore di reali, parametro di input opzionale da inserire solo se problema è non lineare, contiene la soluzione iniziale del problema.
    - *time:* vettore di numeri reali, parametro opzionale di input da inserire solo se il problema è non lineare.
  - Parametri di Output:
    - *u*: vettore di reali, contiene la soluzione del problema calcolata nei nodi della griglia generata in precedenza.
    - *K:* matrice di reali, detta anche matrice di *stiffness* del problema.
    - *F*: vettore di reali.
    - *B*: matrice di reali, è un parametro di output opzionale che va usato in combinazione con ud.
    - *ud:* vettore di reali, è un parametro di output opzionale che va usato in combinazione con B. Vale la relazione $u = B * u1 + ud$ dove $u1 = K \backslash F$
    - *Q,G,H,R:* vettori di reali, sono parametri di output opzionali, fare riferimento ad *assemb* per maggiori dettagli.

◦ Approfondimento sulla funzione *assempde:*

Assempde implementa un algoritmo basato sul metodo di **Galerkin**, il quale discretizza il problema formulato con il **Metodo degli** E**lementi Finiti** (da qui semplicemente FEM). Questo metodo porta alla risoluzione numerica del problema in pochi passi.

1. Si fissa il tipo di funzione che approssima la soluzione esatta del problema. La soluzione approssimata del problema è indicata con **u**, essa è una funzione polinomiale a tratti lineare. In questo modo il suo gradiente sarà costante all'interno dei triangoli della griglia, ma discontinuo lungo i lati del triangolo.

2. Il problema di partenza viene riscritto nella sua forma *debole* o *variazionale*. Questa la si ottiene andando a moltiplicare l'equazione di partenza per una funzione test arbitraria **U**, che deve essere della stessa classe di funzioni di u. Introdotta la funzione test, l'equazione ottenuta viene integrata su tutto il dominio Ω:

$$\int_\Omega -(\nabla \cdot (c\nabla u))v + auv \; dx = \int_\Omega fv dx$$

Usando la *formula di Green* per il laplaciano, si ottiene:

$$\int_\Omega (c\nabla u) \cdot \nabla v + auv dx - \int_{\partial\Omega} \vec{n} \cdot (c\nabla u) v ds = \int_\Omega fv dx$$

L'integrale curvilineo può essere riscritto usando le condizioni al contorno, ottenendo:

$$\int_\Omega (c\nabla u) \cdot \nabla v + auv dx - \int_{\partial\Omega} (-qu + g)v ds = \int_\Omega fv dx$$

Da questa si ricava la forma debole del problema di partenza:

$$\left(\int_\Omega (c\nabla u) \cdot \nabla v + auv - fv \; dx - \int_{\partial\Omega} (-qu + g)v ds\right) = 0 \qquad \forall v$$

L'aggettivo debole è dovuto al fatto che ogni soluzione del problema differenziale è anche soluzione del problema variazionale, ma non è completamente vero il contrario: la soluzione debole è soluzione del problema differenziale nel senso delle distribuzioni. Infine il *lemma di Lax-Milgram* assicura che il problema variazionale è ben posto: la soluzione esiste ed è unica per le ipotesi fatte in precedenza su u e U, come indicato in [MNPD].

3. Seguendo la trattazione di Galerkin per la discretizzazione del problema in forma debole, si proietta il problema dallo spazio di funzioni V allo spazio di funzioni a dimensione finita V_{Np}. Valgono le seguenti proprietà:

$$V_{np} \subset V, \; dim V_{Np} = Np \; .$$

Dove Np è il numero di punti di griglia, quindi Np dipende dal parametro di discretizzazione Hmax. Questo vuol dire che le funzioni $u, U \in V_{Np}$. La convergenza del metodo è assicurata imponendo che V_{Np} tende a V se $Np \to \infty$. Consideriamo che questa operazione di proiezione nello spazio a dimensione finita è equivalente all'operazione di interpolazione, la soluzione all'interno dei triangoli viene ricavata interpolando la soluzione numerica nei vertici dei triangoli.

4. Il problema nella forma debole viene quindi risolto numericamente. Imponendo che U sia la base canonica dello spazio V_{Np} ovvero $U = \phi_i \; \forall \; i \leq Np$, l'equazione in forma debole restituisce il seguente sistema di equazioni algebriche:

$$\int_\Omega (c\nabla u) \cdot \nabla \phi_i + au\phi_i - f\phi_i dx - \int_{\partial\Omega} (-qu + g)\phi_i ds = 0, \qquad i = 1, \ldots, N_p$$

In modo analogo è possibile esprimere anche la soluzione u mediante la base canonica dello spazio discreto:

$$u(x_i) = \sum_{j=1}^{N_p} U_j \phi_j(x_i) = U_i$$

Ottenendo infine il seguente sistema di equazioni:

$$\sum_{j=1}^{N_p} \left(\int_\Omega (c \nabla \phi_j) \cdot \nabla \phi_i + a \phi_j \phi_i dx + \int_{\partial\Omega} q \phi_j \phi_i ds \right) U_j$$

$$= \int_\Omega f \phi_i dx + \int_{\partial\Omega} g \phi_i \mathrm{d}s \qquad i = 1, \ldots, N_p$$

Considerando la seguente notazione:

$$K_{i,j} = \int_\Omega (c \nabla \phi_j) \cdot \nabla \phi_i dx$$

$$M_{i,j} = \int_\Omega a \phi_j \phi_i dx$$

$$Q_{i,j} = \int_{\partial\Omega} q \phi_j \phi_i ds$$

$$F_i = \int_\Omega f \phi_i dx$$

$$G_i = \int_{\partial\Omega} g \phi_i ds$$

Il sistema viene quindi riscritto nella forma compatta $(K+M+Q)*U=F+G$. Dove K, M e Q sono matrici quadrate di dimensione *Np* la cui somma restituisce la matrice di *rigidezza* o di *stiffness*; F e G sono vettori colonna di lunghezza *Np*. La linearità a tratti imposta assicura che gli integrali che compongono la matrice di rigidezza esistono. La matrice di rigidezza si presenta come una matrice sparsa, questo perché il termine ϕ_i è diverso da zero solo all'interno dei triangoli che contengono il nodo i-esimo. Quindi in generale gli integrali che determinano i termini $K_{i,j}$, $M_{i,j}$, $Q_{i,j}$, F_i, G_i devono essere calcolati solo all'interno dei triangoli che contengono il nodo i-esimo, inoltre i termini $K_{i,j}$, $M_{i,j}$ sono nulli tranne quando i nodi i e j sono vertici dello stesso triangolo.

5. Si procede a questo punto con la costruzione delle matrici del sistema: questo processo viene detto di *assembling* e viene eseguito dalla function **assema**. Il codice analizza l'intera griglia e per ogni triangolo calcola i coefficienti del problema. Si consideri il generico triangolo locale *ΔP1P2P3* come in Fig. 10.

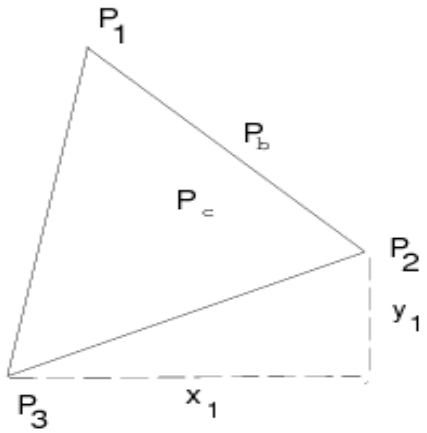

Fig. 10: Triangolo Locale

Di questo triangolo prima si determina la geometria calcolando l'area e la posizione del baricentro, poi si determina il gradiente di ϕ_i:

$$P_c = \frac{P_1 + P_2 + P_3}{3}$$

$$\nabla\phi_1 = \frac{1}{2\,\mathrm{area}(\Delta P_1 P_2 P_3)}\begin{bmatrix} y_1 \\ -x_1 \end{bmatrix}$$

Noti questi si calcolano i coefficienti del problema:

$$k_{i,j} = \frac{1}{4\,\mathrm{area}(\Delta P_1 P_2 P_3)}[y_j, -x_j]c(P_c)\begin{bmatrix} y_1 \\ -x_1 \end{bmatrix}$$

$$m_{i,j} = \int_{\nabla P_1 P_2 P_3} a(P_c)\phi_i(x)\phi_j(x)dx = a(P_c)\frac{\mathrm{area}(\Delta P_1 P_2 P_3)}{12}(1 + \delta_{i,j})$$

$$f_i = f(P_c)\frac{\mathrm{area}(\Delta P_1 P_2 P_3)}{3}$$

Se 2 vertici del triangolo sono posti lungo il contorno $\partial\Omega$, saranno presenti anche elementi delle matrici Q e G . Ad esempio se i punti $P1, P2 \in \partial\Omega$, si ricava:

$$Q_{i,j} = q(P_b)\frac{|P_1 - P_2|}{6}(1 + \delta_{i,j}), \qquad i, j = 1, 2$$

$$G_i = g(P_b)\frac{|P_1 - P_2|}{2}, \qquad i = 1, 2$$

dove P_b è il punto medio del lato $P_1 P_2$.

Dalle formule ricavate è possibile notare che gli integrali vengono calcolati usando la formula del punto medio. Questa approssimazione è ottimale poiché ha lo stesso ordine di accuratezza dell'interpolazione lineare a tratti.

6. A questo punto non resta che ricavare il vettore della soluzione, andando a risolvere il sistema sparso ottenuto in precedenza. Nel caso in cui in output è stato richiesto solo il vettore u, il codice risolve automaticamente il sistema. Nel caso in cui l'utente ha richiesto in output K, F è possibile scegliere uno dei metodi di risoluzione iterativi in Matlab.

- **Adaptmesh:** funzione che genera per la EDP ellittica assegnata (anche non lineare) una griglia adattativa, sulla quale il codice risolve automaticamente il problema.

  ```
  [u,p,e,t]=adaptmesh(dl,b,c,a,f)
  [u,p,e,t]=adaptmesh(dl,b,c,a,f,'PropertyName',PropertyVal
  ue,)
  ```

  - Parametri di Input:
    - *dl:* Decomposed Geometry, Matrix matrice di reali, indica il dominio del problema.
    - *b:* matrice di reali, indica le condizioni al contorno del problema.
    - *c,a,f:* coefficienti del problema.
    - *'PropertyName':* stringa di caratteri, è un parametro di input opzionale. Contiene il nome dell'opzione del codice che si vuole modificare.
    - *PropertyValue:* stringa di caratteri o numero reale, è un parametro di input opzionale. Contiene il valore dell'opzione che si vuole modificare.
      Nella seguente tabella è presente la lista delle opzioni che l'utente può modificare, con i relativi *PropertyName e PropertyValue.*

Property Name	PropertyValue	Default	Descrizione
`Maxt`	Intero positivo	`inf`	Massimo numero di triangoli che possono essere introdotti nella griglia ad ogni iterazione
`Ngen`	Intero positivo	`10`	Numero massimo di iterazioni
`Mesh`	`p1, e1, t1`	`initmesh`	Griglia di partenza
`Tripick`	MATLAB function	`pdeadworst`	Funzione che indica come scegliere i triangoli da raffinare
`Par`	Reale positivo	`0.5`	Parametri delle funzioni tripick
`Rmethod`	`longest|regular`	`longest`	Metodo con cui raffinare la griglia
`Nonlin`	`on|off`	`off`	Indica equazione non lineare
`Toln`	Reale positivo	`1e-4`	Tolleranza per la soluzione del problema non lineare
`Init`	`u0`	`0`	Soluzione iniziale del problema non lineare
`Jac`	`fixed|lumped|full`	`fixed`	Metodo da usare per calcolare la matrice Jacobiana in caso di EDP non lineare
`norm`	numeric`|inf|energy`	`inf`	Tipo di norma da usare per calcolare i residui in caso di EDP non lineare

    - Parametri di Output:
        - *u*: vettore di reali, contiene la soluzione calcolata nei punti di griglia.
        - *p,e,t:* matrici di reali, contengono le informazioni sulla griglia generata (per maggiori informazioni fare riferimento alla funzione *initmesh*).
- **griddata:** funzione che interpola i dati ottenuti su una griglia di partenza, per ottenere dati su una nuova griglia.

```
ZI = griddata(x,y,z,XI,YI)
XI,YI,ZI] = griddata(x,y,z,xI,yI,method)
```

    - Parametri di Input:
        - *x,y:* vettori di reali, indicano la griglia di partenza.
        - *z*: vettore di reali, indica i dati di partenza.

        - *xI,yI:* vettori di reali, indicano la griglia in cui trovare i nuovi dati.
        - *method*: stringa di caratteri, parametro opzionale di input, indica il metodo di interpolazione che si vuole usare.
    - Parametri di Output:
- *ZI:* vettore di reali, contiene i dati calcolati.
- *XI,YI:* matrici di reali, parametri opzionali di output, definiscono la nuova griglia, ovvero [XI,YI]=meshgrid(XI,YI).

2. Risoluzione numerica di problemi di tipo parabolico. In questo capitolo si descrivono brevemente i metodi numerici per la risoluzione di EDP di tipo parabolico in ambiente MATLAB, PDEtoolbox.

In Matlab le EDP di tipo parabolico devono essere riportate nella seguente forma, per poter essere risolte numericamente:

$$d\frac{\partial u}{\partial t}-\nabla\cdot(c\nabla u)+a\,u=f \quad \text{in} \quad \Omega \tag{17}$$

con condizioni iniziali:

$$u(x,y,0)=u0(x,y) \quad \text{con} \quad (x,y)\in\Omega \quad ; \tag{18}$$

e condizioni al contorno in una delle seguenti forme:

$$hu=r \quad \text{in} \quad \partial\Omega \tag{19}$$

$$\boldsymbol{n}\cdot(c\nabla u)+q\,u=g \quad \text{in} \quad \partial\Omega \quad \text{per} \quad t>0 \tag{20}$$

Le funzioni c, a, f, d sono definite in Ω e sono funzioni del tempo.

La (19) indica condizioni alla Dirichlet, poiché si conosce la funzione u sul contorno del dominio, al variare del tempo. La (20) indica condizioni miste alla Robin, nel caso in cui q è la funzione identicamente nulla si hanno condizioni alla Neumann, poiché si conosce sul contorno del dominio solo la derivata di u nella direzione normale al contorno, al variare del tempo.

Il modo di procedere in questo studio è del tutto analogo a quello usato per i problemi di tipo ellittico: partendo da problemi test, si studiano e si confrontano i possibili metodi di risoluzione numerica del problema ed i risultati ottenuti, usando come confronto i risultati esatti. Infine sono descritte brevemente le varie funzioni Matlab utilizzate.

2.1. Definizione problemi test. I problemi test di tipo parabolico studiati sono i seguenti:

- 1D:

  $$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2} \quad \text{per} \quad t\in]0,1[\quad x\in]0,1[\tag{21}$$

  condizioni iniziali:

  $$u(x,0)=\sin(\pi x) \quad \text{per}\, x\in]0,1[\tag{22}$$

  e condizione al contorno alla Neumann:

$$\frac{\partial u}{\partial x}=\pi e^{-2\pi^2 t} \quad \text{con} \quad x=0 \quad t\in[0,1]; \tag{23}$$

$$\frac{\partial u}{\partial x}=-\pi e^{-2\pi^2 t} \quad \text{con} \quad x=1 \quad t\in[0,1]; \tag{24}$$

Questo problema presenta la seguente soluzione esatta:

$$u(x,t)=\sin(\pi x)e^{-\pi^2 t} \tag{25}$$

- 2D:

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2} \quad \text{con} \quad t\in]0,1[\quad (x,y)\in]0,1[\times]0,1[\tag{26}$$

Condizioni iniziali:

$$u(x,y,0)=\sin(\pi x)\sin(\pi y) \quad \text{con} \quad (x,y)\in]0,1[\times]0,1[\tag{27}$$

e condizioni al contorno alla Robin:

$$\frac{\partial u}{\partial x}=+2u=\pi e^{-2\pi^2 t}\sin(\pi x) \quad \text{con} \quad y=0 \quad x\in[0\,1] \quad t\in[0;1] \quad ; \tag{28}$$

$$\frac{\partial u}{\partial x}=+2u=-\pi e^{-2\pi^2 t}\sin(\pi x) \quad \text{con} \quad y=1 \quad x\in[0\,1] \quad t\in[0;1] \quad ; \tag{29}$$

$$\frac{\partial u}{\partial y}=+2u=\pi e^{-2\pi^2 t}\sin(\pi y) \quad \text{con} \quad x=0 \quad y\in[0\,1] \quad t\in[0;1] \quad ; \tag{30}$$

$$\frac{\partial u}{\partial y}=+2u=-\pi e^{-2\pi^2 t}\sin(\pi y) \quad \text{con} \quad x=1 \quad y\in[0\,1] \quad t\in[0;1] \quad ; \tag{31}$$

Questo problema presenta la seguente soluzione esatta:

$$u(x,y,t)=e^{-2\pi^2 t}\sin(\pi x)\sin(\pi y) \quad ; \tag{32}$$

2.2. Risoluzione problemi test. Anche i problemi parabolici esattamente come i problemi ellittici, possono essere risolti mediante la GUI, oppure mediante le apposite funzioni richiamate dalla CL.

2.2.1. Soluzione problema test 1D.

- Soluzione mediante CL

La risoluzione numerica dei problemi monodimensionali di tipo parabolico viene effettuata mediante CL usando la funzione **pdepe** (per maggiori informazioni fare riferimento al paragrafo 2.3)**,** questa non è una funzione del PDEtoolbox, ma fa parte delle funzioni di base di Matlab. Questa funzione può essere usata solo in pochi casi: problemi monodimensionali legati ad EDP di tipo parabolico oppure sistemi di EDP con equazioni paraboliche ed ellittiche, il codice non risolve i problemi monodimensionali legati ad una sola EDP ellittica o ad un sistema di sole EDP ellittiche.

Questo perché la funzione semidiscretizza il problema, ovvero lo trasforma in un problema legato alla risoluzione di un sistema di ODE nella sola variabile temporale, questo infine viene risolto con la funzione **ode15s** (per maggiori informazioni fare riferimento al paragrafo 2.3). La funzione *ode15s* permette all'utente di definire delle tolleranze per l'errore sulla soluzione.

La funzione *pdepe* richiede che l'equazione da risolvere venga riportata in una forma diversa da quella richiesta dalle funzioni del PDEtoolbox:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t}=x^{-m}\frac{\partial}{x}\left(x^{m}f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right)+s\left(x,t,u,\frac{\partial u}{\partial x}\right) \tag{33}$$

Con dati iniziali: $u(t0,x)=u0(x)$; mentre nei punti $x=x0$ e $x=xf$ le condizioni al contorno devono essere riportate nella seguente forma:

$$p(x,t,u)+q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right)=0 \tag{34}$$

Per la risoluzione del problema si consideri il seguente script file ed i seguenti function files:

```
m=0;
x=[0:0.1:1];
t=[0:0.1:1];
sol=pdepe(m,@pdefun,@icfun,@bcfun,x,t);
U=sol(:,:,1);
surf(x,t,U)
xlabel('distanza x');
ylabel('tempo');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [c,f,s]=pdefun(x,t,u,DuDx)
c=1;
f=DuDx;
s=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u0=icfun(x)
u0=sin(pi.*x);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [pl,ql,pr,qr]=bcfun(xl,ul,xr,ur,t)
pl=-pi*exp(-(pi^2)*t);
ql=1;
pr=pi*exp(-(pi^2)*t);
qr=1;
```

La soluzione ottenuta è riportata in Fig. 11.

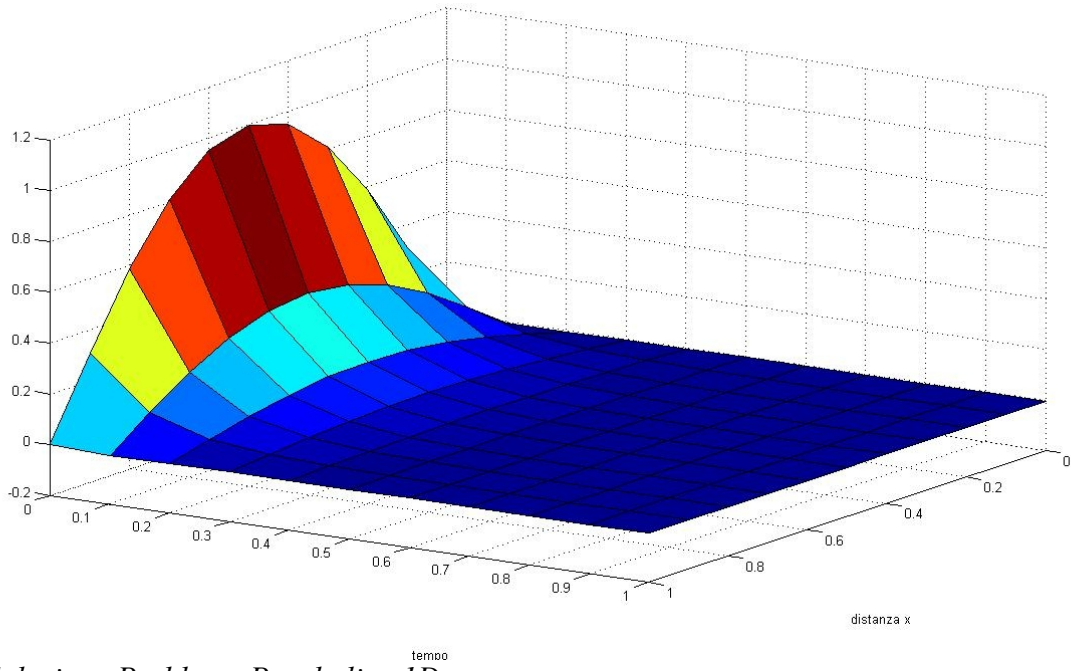

Fig 11: Soluzione Problema Parabolico 1D

2.2.2. Soluzione problema test 2D:

- Soluzione mediante GUI:

Risolvere problemi parabolici mediante GUI è relativamente semplice: esattamente come per i problemi ellittici, definire il problema che si vuole risolvere è immediato. Le operazioni che portano alla creazione del dominio, delle condizioni al contorno e della griglia sono le stesse che vengono compiute per EDP ellittiche. Queste operazioni sono quindi omesse, tenendo conto che sono descritte nel paragrafo 1.2.1.

Il dominio del problema e la griglia creata sono mostrati in Fig. 12.

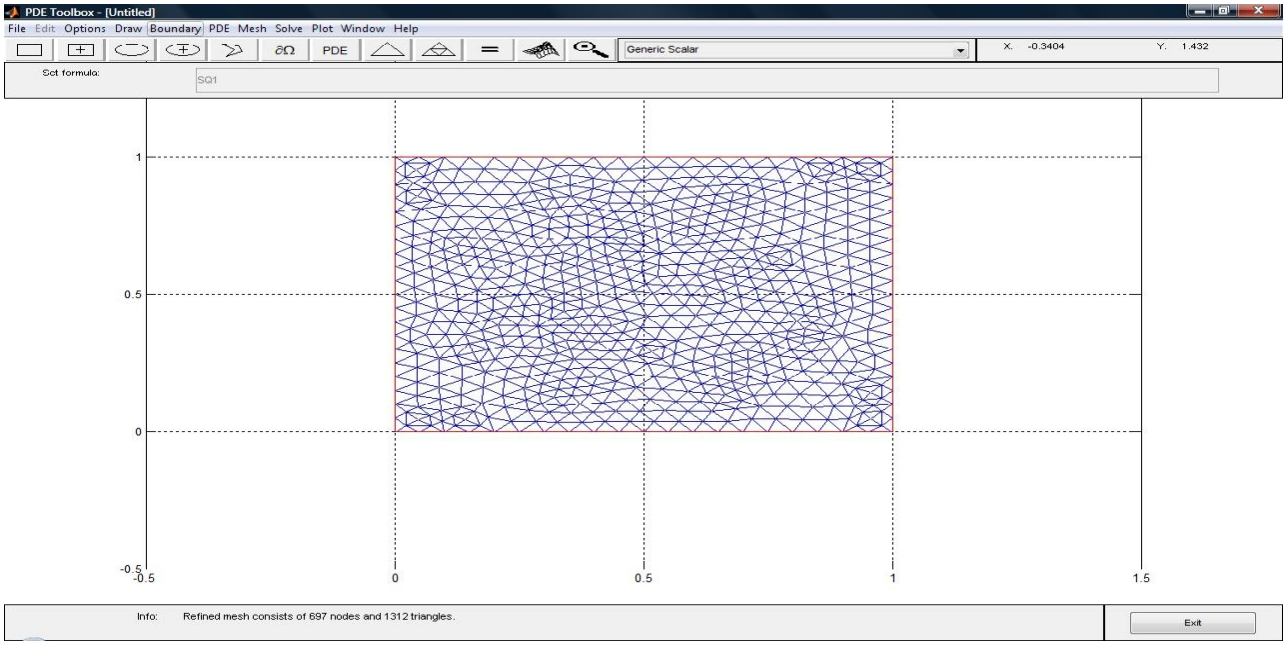

Fig 12: Dominio e Griglia Problema Parabolico 2D

Le condizioni al contorno alla Neumann inserite sono le seguenti:

per $y=0$: $q=2$, $g=\pi e^{-2\pi^2 t}\sin(\pi x)$;
per $y=1$: $q=2$, $g=-\pi e^{-2\pi^2 t}\sin(\pi x)$;
per $x=0$: $q=2$, $g=\pi e^{-2\pi^2 t}\sin(\pi x)$;
per $x=1$: $q=2$, $g=-\pi e^{-2\pi^2 t}\sin(\pi x)$;

A questo punto non resta che definire l'equazione del problema, generare la soluzione e visualizzarla.

- **Definire l'equazione del problema:**
  - Entrare nel menù **PDE** e selezionare in sequenza
    - **PDE Mode.**
    - **PDE Specification:** qui specificare il tipo di problema, in questo caso parabolico. Poi inserire i coefficienti del problema: $c=1, a=0, f=0, d=1$.
- **Generare la soluzione del problema:**
  - Entrare nel menù **Solve** e selezionare:
    - **Parameters:** qui si devono definire altri parametri del problema:
      - **Time:** bisogna indicare gli istanti di tempo in cui si vuole conoscere la soluzione del problema. In questo caso: linspace(0,10)
      - **u(t0):** bisogna indicare il dati iniziali del problema da risolvere. Per questo problema: *sin(pi.*x).*sin(pi.*y)*.
      - **Relative Tollerance:** impostare la tolleranza sull'errore relativo con cui si vuole che *ode15s* risolva il sistema di ODE che si ottiene semidiscretizzando il problema. Inserire: 0.001.
      - **Absolute Tollerance:** inserire la tolleranza sull'errore assoluto per il codice *ode15s*. Inserire: 0.0001.
    - **Solve PDE:** la soluzione del problema viene calcolata (per maggiori dettagli fare riferimento al paragrafo 2.3 nella sezione relativa alla funzione **parabolic**).
- **Visualizzare la soluzione del problema:**

- Spostarsi nel menù **Plot** e selezionare:
  - **Parameters:** qui è possibile modificare il diagramma della soluzione che il software di default restituisce. Ad esempio:
    - **Height** permette di ottenere il diagramma 3D della soluzione.
    - **Show Mesh** permette di diagrammare anche la griglia di partenza.
    - **Colormap** permette di cambiare i colori del diagramma: scegliere jet.
    - **Time for plot** istante di tempo in cui diagrammare la soluzione.
    - **Animation** permette di creare un filmato della soluzione. In *options* l'utente può scegliere il numero di fotogrammi al secondo del filmato(*fps*) ed il numero di volte che il filmato viene ripetuto (*Number of repeats*).

A titolo d'esempio in Fig. 13 sono presenti le soluzioni ottenute al tempo $t0=0$ ed al tempo $tf=10$.

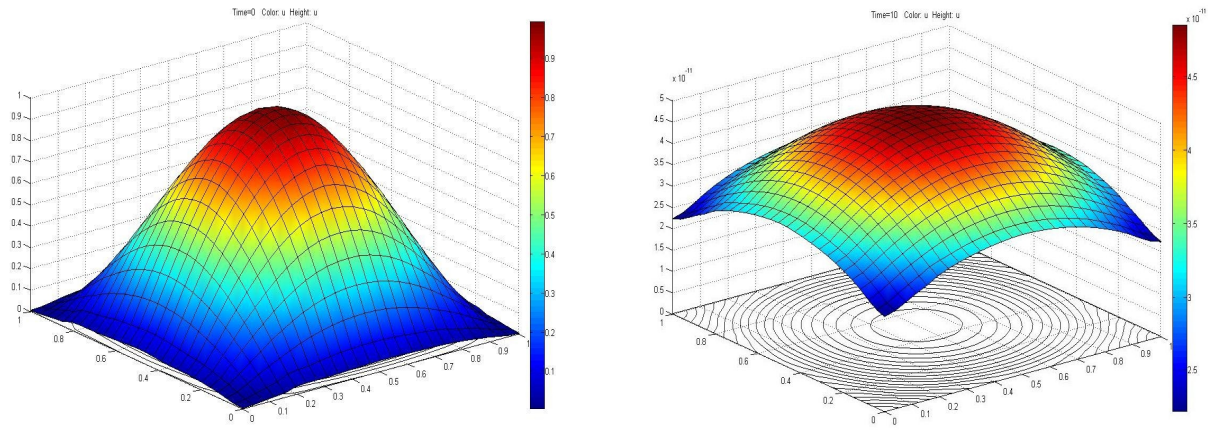

Fig. 13: Soluzione Prolema Parabolico GUI t0 e tf

- Soluzione mediante CL:

Per la soluzione mediante CL del problema test 2D si consideri il seguente script file:

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
[p,e,t]=initmesh(dl);
[p,e,t]=refinemesh(dl,p,e,t);
b=[ 1 1 1 1;
    0 0 0 0;
    1 1 1 1;
    28 28 28 28;
    50 50 50 50;
    43 45 43 45;
    112 112 112 112;
    105 105 105 105;
    42 42 42 42;
    101 101 101 101;
    120 120 120 120;
    112 112 112 112;
    40 40 40 40;
    45 45 45 45;
    50 50 50 50;
```

```
    42 42 42 42;
    112 112 112 112;
    105 105 105 105;
    94 94 94 94;
    50 50 50 50;
    42 42 42 42;
    116 116 116 116;
    41 41 41 41;
    42 42 42 42;
    115 115 115 115;
    105 105 105 105;
    110 110 110 110;
    40 40 40 40;
    112 112 112 112;
    105 105 105 105;
    42 42 42 42;
    120 121 120 121;
    41 41 41 41;];
u0='sin(pi.*x).*sin(pi.*y)'; c=1; a=0; f=0; d=1;
tlist=[linspace(0,1),linspace(1.1,10,20)];
U=parabolic(u0,tlist,b,p,e,t,c,a,f,d,10^(-3),10^(-4));
k=size(tlist,2);
pdeplot(p,e,t,'xydata',U(:,1),'zdata',U(:,1),'mesh','on','flowstyle','arrow','co
ntour','on')
pause
pdeplot(p,e,t,'xydata',U(:,k),'zdata',U(:,k),'mesh','on','flowstyle','arrow','co
ntour','on');
pause
%per ottenere il filmato della soluzione:
umax=max(max(U));
umin=min(min(U));
[m,n]=size(U);
for i=1:n
pdeplot(p,e,t,'xydata',U(:,i),'zdata',U(:,i),'mesh','on','flowstyle','arrow','co
ntour','on');
axis([0 1 0 1 umin umax]); caxis([umin umax]);
Mipe(:,i)=getframe;
end
movie(Mipe,2,1)
```

Visto che non è possibile riportare il filmato ottenuto, come esempio in Fig. 14 sono riportate le soluzioni al tempo $t0=0$ ed al tempo $tf=10$.

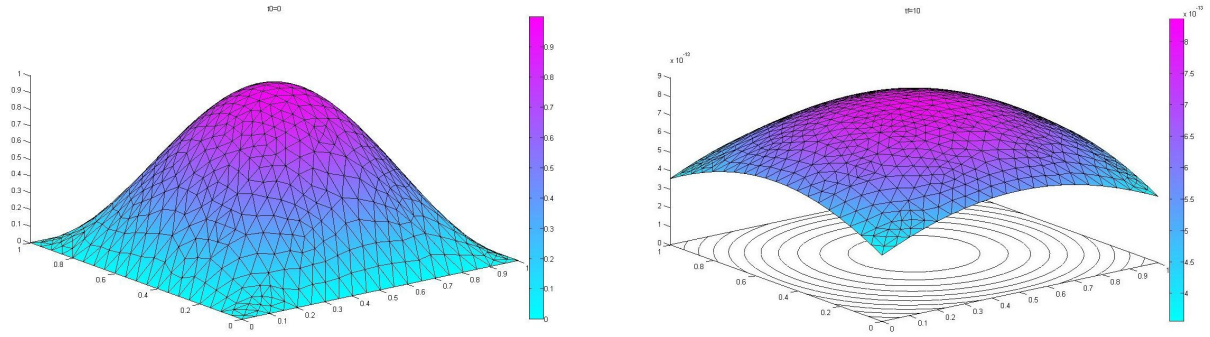

Fig. 14: Soluzione Parabolico CL t0 e tf

Analizzando lo script file si ricava che **parabolic** è la funzione che risolve le EDP di tipo

parabolico. Questa funzione trasforma la EDP in un sistema di ODE che viene risolto con **ode15s** (per maggiori informazioni fare riferimento al paragrafo 2.3). Come per il caso monodimensionale è possibile intervenire sulla precisione della soluzione numerica indicando come parametro di input di *parabolic* la tolleranza per l'errore relativo e per l'errore assoluto, parametri che vengono utilizzati direttamente dalla funzione *ode15s*. Il codice inoltre restituisce in output informazioni sul costo computazionale della risoluzione del sistema di ODE.

Il seguente script file studia come varia il costo computazionale della risoluzione del problema test 2D al variare del passo di discretizzazione spaziale. Fissati gli istanti temporali in cui si vuole conoscere la soluzione del problema e le tolleranze per gli errori, l'algoritmo dimezza ad ogni iterazione il parametro Hmax, generando triangoli il più possibile equilateri, avendo settato il parametro *hgrad=1.01*, dove hgrad è un parametro di input opzionale di *initmesh* che indica il rapporto $\frac{Hmax}{Hmin}$ ovvero il rapporto tra il lato più lungo ed il lato più corto dei triangoli di griglia, esso è un numero reale compreso tra 1 e 2 .

```
gd=[3;4;0;1;1;0;0;0;1;1];
dl=decsg(gd);
b=[ 1 1 1 1;
    0 0 0 0;
    1 1 1 1;
    28 28 28 28;
    50 50 50 50;
    43 45 43 45;
    112 112 112 112;
    105 105 105 105;
    42 42 42 42;
    101 101 101 101;
    120 120 120 120;
    112 112 112 112;
    40 40 40 40;
    45 45 45 45;
    50 50 50 50;
    42 42 42 42;
    112 112 112 112;
    105 105 105 105;
    94 94 94 94;
    50 50 50 50;
    42 42 42 42;
    116 116 116 116;
    41 41 41 41;
    42 42 42 42;
    115 115 115 115;
    105 105 105 105;
    110 110 110 110;
    40 40 40 40;
    112 112 112 112;
    105 105 105 105;
    42 42 42 42;
    120 121 120 121;
    41 41 41 41;];
u0='sin(pi.*x).*sin(pi.*y)'; c=1; a=0; f=0; d=1;
tlist=[linspace(0,1),linspace(1.1,10,20)]; h(1)=0.1; i=1;
while i<=5
    [p,e,t]=initmesh(dl,'hmax',h(i),'hgrad',1.01);
    U=parabolic(u0,tlist,b,p,e,t,c,a,f,d,10^(-5),10^(-6));
    i=i+1;
    h(i)=h(i-1)/2;
end
```

```
h(i)=[];
```

Per questa analisi sono stati settati i seguenti parametri: 1,00E-005 come tolleranza per l'errore assoluto, 1,00E-006 come tolleranza per l'errore relativo e 120 istanti di tempo in cui conoscere la soluzione, il costo computazionale del problema in funzione di Hmax varia nel seguente modo:

Hmax	1,00E-001	5,00E-002	2,50E-002	1,25E-002	6,30E-003
Step Temporali	196	215	225	243	268
Valutazioni di Funzioni	382	418	430	465	501
Fattorizzazioni LU	34	37	39	42	51
Jacobiani Valutati	1	1	1	1	1
Sistemi Lineari Risolti	381	417	429	464	500

2.3. Descrizione delle principali funzioni del PDE toolbox di Matlab per problemi parabolici. Le funzioni usate in precedenza per descrivere il dominio del problema, per generare la griglia all'interno del dominio, per specificare le condizioni al contorno, sono le stesse funzioni usate nel caso di EDP ellittiche. Per maggiori informazioni su queste funzioni quindi fare riferimento al paragrafo 1.3. In questo paragrafo invece sono descritte solo le nuove funzioni introdotte per risolvere mediante CL problemi di tipo parabolico.

- ***Pdepe*:** funzione di base di Matlab, risolve EDP paraboliche monodimensionali oppure sistemi di EDP paraboliche ed ellittiche monodimensionali.

  ```
  sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan)
  sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,options)
  ```

  - Parametri di input:
    - *m:* intero positivo, indica la simmetria del problema: m=0 nessuna simmetria, m=1 simmetria cilindrica, m=2 simmetria sferica.
    - *Pdefun:* stringa di caratteri, contiene il nome della function che definisce l'equazione del problema nella forma indicata nella (33).
    - *Icfun*: stringa di caratteri, contiene il nome della function che definisce i dati iniziali del problema.
    - *Bcfun*: stringa di caratteri, contiene il nome della function che definisce le condizioni al contorno del problema nella forma indicata nella (34).
    - *Xmesh:* vettore di reali, indica i punti di griglia in cui andare a calcolare la soluzione.
    - *Tspan:* vettore di reali, indica gli istanti di tempo in cui calcolare la soluzione.
    - *Options:* struttura dati, è un parametro di input opzionale che viene costruito con la funzione *odeset*. Indica le opzioni della funzione *ode15s* che l'utente vuole modificare.
  - Parametri di output:
    - *Sol:* matrice di reali, contiene la soluzione del problema nei punti indicati con xmesh, agli istanti di tempo indicati con tspan. La colonna i-esima della matrice *sol* contiene la soluzione all'istante di tempo tspan(i) per tutti i punti di griglia. La riga j-

esima della matrice *sol* indica la soluzione al variare del tempo calcolata nel punto di griglia xmesh(j).

- **Parabolic:** funzione che risolve le EDP paraboliche una volta definiti: il dominio, la griglia, i dati iniziali, le condizioni al contorno, i coefficienti del problema.

```
u1=parabolic(u0,tlist,b,p,e,t,c,a,f,d)
u1=parabolic(u0,tlist,b,p,e,t,c,a,f,d,rtol,atol)
u1=parabolic(u0,tlist,K,F,B,ud,M)
u1=parabolic(u0,tlist,K,F,B,ud,M,rtol,atol)
```

  - Parametri di input:
    - *u0:* stringa di caratteri oppure matrice di reali, u0 è una stringa di caratteri quando si conosce la funzione analitica che restituisce i dati iniziali del problema, u0 è una matrice quando non si conosce direttamente la funzione analitica che restituisce i dati iniziali, ma solo i valori che essa assume nei punti di griglia.
    - *tlist:* vettore di reali contiene gli instanti di tempo in cui si vuole conoscere la soluzione del problema.
    - *b:* matrice di reali, indica le condizioni al contorno del problema (per maggiori informazioni fare riferimento al paragrafo 1.3).
    - *p,e,t:* matrici che indicano la griglia generata all'interno del dominio (per maggiori informazioni fare riferimento al paragrafo 1.3).
    - *c,a,f,d:* coefficienti del problema.
    - *rtol,atol:* numeri reali, parametri di input opzionali. Indicano al codice *ode15s* la tolleranza sull'errore relativo ed assoluto per la risoluzione del sistema di ODE che si ricava dalla semidiscretizzazione della EDP.
    - *K,F,B,ud,M:* matrici di reali, sono parametri di input opzionali. Tramite queste matrici si indica al codice direttamente il problema semidiscretizzato e non la EDP di partenza. In questo modo il codice risolve direttamente mediante *ode15s* il seguente problema:

      $$B'MB\frac{du_i}{dt} + K \cdot u_i = F$$

      Ricavando la soluzione come segue: $u1=Bu_i+u_d$
  - Parametri di output:
    - *u1:* matrice di reali, contiene la soluzione del problema nei punti della griglia generata, agli istanti di tempo indicati con il vettore *tlist*. La colonna i-esima della matrice contiene la soluzione all'istante di tempo *tlist*(i) per tutti i punti di griglia. La riga j-esima della matrice indica la soluzione al variare del tempo calcolata nel punto di griglia di coordinate x=p(1,j) y=p(2,j).
  - Approfondimento sulla funzione *parabolic*:

[PDE_UG] Il codice *parabolic* usa per risolvere numericamente le EDP paraboliche il *metodo di semidiscretizzazione delle linee.*

L'equazione di partenza viene prima semidiscretizzata: ovvero viene discretizzata solo la parte spaziale del problema. L'equazione viene trattata come se fosse una EDP ellittica: quindi ancora viene usato il *metodo di Galerkin* per discretizzare il problema riscritto in forma debole (per maggiori informazioni sul *metodo di Galerkin* fare riferimento al paragrafo 1.3, alla sezione di approfondimento della funzione *assempde*). In questo caso però dopo la discretizzazione spaziale del problema si ottiene il seguente sistema di ODE e non un semplice sistema algebrico:

$$\sum_i [(\int_\Omega d\,\phi_i\phi_j\,dx)\frac{dU_i(t)}{dt}] + \sum_i [(\int_\Omega [\nabla\phi_j\cdot(c\,\nabla\phi_i) + a\,\phi_j\phi_i]\,dx + \int_{\partial\Omega} q\,\phi_j\phi_i\,ds)\,U_i(t)] = \int_\Omega f\,\phi_j\,dx + \int_{\partial\Omega} g\,\phi_j\,ds$$

dove questa volta

$$u(x,t) = \sum_i U_i(t)\phi_i(x)$$

Si consideri la seguente notazione:

$$M(j,i) = \int_\Omega d\,\phi_i\phi_j\,dx$$

$$K(j,i) = \int_\Omega [\nabla\phi_j\cdot(c\nabla\phi_i) + a\,\phi_j\phi_i]\,dx + \int_{\partial\Omega} q\,\phi_j\phi_i\,ds$$

$$F(j) = \int_\Omega f\,\phi_j\,dx + \int_{\partial\Omega} g\,\phi_j\,ds$$

Questa porta a scrivere il sistema nella forma compatta:

$$M\frac{dU}{dt} + KU = F$$

dove ancora K è la matrice di *stiffness* o di *rigidezza* ed F è il vettore dei termini noti del sistema, mentre M è chiamata matrice di *massa* del sistema.
Poiché K e M sono matrici sparse di grosse dimensioni, si ottiene un sistema di ODE sparso. Il problema legato al sistema di ODE in molti casi è anche stiff e malcondizionato, questo dipende dai coefficienti della EDP di partenza. Il sistema di ODE quindi viene risolto numericamente con la maggiore precisione possibile ed il minor costo computazionale possibile dalla funzione **ode15s**, questa non è una funzione del PDEtoolbox ma rientra nelle funzioni del *Matalb ODE Suite.* La funzione *ode15s* viene usata per i seguenti motivi: essa implementa un algoritmo *implicito ed adattativo*, così da ottenere un metodo numerico assolutamente stabile per qualsiasi passo temporale scelto, il quale è fissato dinamicamente dal codice in base alla tolleranza sugli errori. La scelta adattativa del passo di discretizzazione temporale comporta un notevole risparmio computazionale: l'uso di un metodo implicito porta a risolvere ad ogni step temporale un sistema lineare sparso. Inoltre il codice calcola la matrice Jacobiana e fattorizza la matrice dei coefficienti solo quando è necessario. Si consideri infine che al termine della routine, se gli istanti di tempo indicati con *tlist* non rientrano negli istanti di tempo analizzati dal codice, a causa della scelta adattativa del passo, i dati mancanti vengono ricavati mediante interpolazione.

- Ode15s: funzione che risolve problemi legati a sistemi di ODE usando uno schema multistep implicito ed adattativo di ordine di accuratezza variabile da 1 a 5.
  `[t,y]=ode15s(odefun,tspan,y0)`
  `[t,y]=ode15s(odefun,tspan,y0,options)`
  - Parametri di input:
    - *odefun:* stringa di caratteri, indica il nome della funzione che definisce il sistema di equazioni differenziali.
    - *tspan:* vettore di reali, contiene l'istante iniziale e l'istante finale dell'intervallo di integrazione.
    - *y0:* vettore di reali, indica i dati iniziali del problema differenziale.
    - *options:* parametro di input opzionale, struttura dati costruita mediante la funzione *odeset,* contiene le opzioni della funzione ode15s che l'utente vuole modificare, ad esempio la tolleranza sull'errore relativo e sull'errore assoluto.
  - Parametri di output
    - *t:* vettore di reali, contiene gli istanti temporali in cui la soluzione è stata calcolata.
    - *y:* matrice di reali, contiene la soluzione calcolata.
  - Approfondimento sulla funzione *ode15s*:
    [SODEII] Ode15s è basata sull'uso di formule multistep note come *Backward Differentiation Formula (BDF)* con ordine di accuratezza che varia da 1 a 5. Queste rientrano in una classe di formule implicite del tipo:

$\sum_{i=1}^{p} \alpha_i y^{n-1} + \gamma f(t^n, y^n) = 0$ dove:

- p ordine di accuratezza della formula usata.
- $\alpha_i, \forall\, 1 \le i \le p$ coefficienti che caratterizzano la formula usata.
- γ coefficiente che caratterizza la formula usata.

BIBLIOGRAFIA

1. [MNPD] Alfio Quarteroni, Modellistica Numerica per Problemi Differenziali, Springer, 2000.

2. [PDE_UG] MathWorks, Partial Differential Equation Toolbox User's Guide, 2010.

3. *[MCNI] Filippo Maria Denaro, Metodi di Calcolo Numerico per l'Ingegneria, Liguori Editore, 2004.*

4. [SODEII] E. Hairer, S.P. Norse, G. Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, Springer, 1991.

5. [NSPDE] K.W. Morton, D. Mayers, Numerical Solution of Partial Differential Equations, Cambridge, 2005.

6. [LCNM] Alessandra D'Alessio, Lezioni di Calcolo Numerico e MATLAB, Liguori Editore, 2006.

7. [CFD] John D. Anderson, JR., Computational Fluid Dynamics, McGraw-Hill, 1995.

8. [IMFD] Richard E. Meyer, Introduction to Mathematical Fluid Dynamics, *Dover Publications, 1982.*