Capitolo 1

Laboratorio di ottimizzazione

1.1 Introduzione

In questo capitolo vedremo come implementare gli algoritmi sviluppati precedentemente in linguaggio MatLab (`http://www.mathworks.com/`) e utilizzare quelli già disponibili nell'ambiente. Successivamente i programmi svillupati saranno testati su problemi modello facili da scrivere e su problemi forniti dalla libreria CUTEr (`http://cuter.rl.ac.uk/cuter-www`) in ambiente MatLab.

Come requisiti, si richiede una buona conoscenza del linguaggio MatLab quali le istruzioni per il controllo del flusso di un programma (`if`, `else`, `for`, `while`, ...), le istruzioni per valutare una espressione (`eval`, `feval`, ...) e la scrittura di funzioni e la differenza tra funzioni e script. Per una introduzione al linguaggio MatLab si veda l'appendice del libro G. Zilli, Calcolo numerico, DMMMSA, Padova. Mentre per quanto riguarda CUTEr è richiesto l'installazione dell'ambiente di lavoro e del suo utilizzo in linguaggio MatLab.

Nella sezione 1.2 vedremo come lavorare con le matrici sparse in MatLab. Poi, nella sezione **??** vedremo come utilizzare CUTEr in ambiente MatLab attraverso due esempi, il primo su un problema non vincolato mentre il secondo su un problema vincolato. Alcuni algoritmi di ottimizzazione non vincolata e vincolata saranno presentati nelle sezioni **??** e **??**.

Tutti i codici degli esempi qui riportati possono essere scariti dalla pagina web degli autori.

1.2 Matrici sparse

Molti problemi che si incontrano nell'ambito della ottimizzazione coinvolgono matrici sparse e di grandi dimensioni dove la presenza degli elementi nulli viene sfruttata per ottimizzare le operazioni di gestione della memoria e di accesso agli elementi della matrice.

Una matrice viene detta sparsa se il numero di elementi diversi da zero (*nonzeri*), sono dell'ordine della dimensione della matrice, cioè $\mathcal{O}(n)$ con n una delle dimensioni della matrice, mentre la loro disposizione prende il nome di *pattern* di sparsità.

Molti formati per l'immagazzinamento degli elementi nonzeri della matrice sono stati sviluppati, ma quello adottato in MatLab è una variante del *formato per coordinate*. In questo formato gli elementi non nulli di una matrice vengono memorizzati in tre vettori che indicheremo con `I`, `J` ed `A`. Nel vettore `A` (reale o complesso) vengono memorizzati gli elementi nonzero della matrice A, mentre nei vettori `I` e `J` (interi) vengono memorizzati i rispettivi indici di riga e colonna degli elementi. La lunghezza dei tre vettori è pari a nnz con nnz il numero di elementi nonzero presenti nella matrice. Questo tipo di formato non prevede che gli elementi siano ordinati per riga o colonna.

Sapendo che l'occupazione in memoria di un intero è 4 byte mentre quella di un numero reale è 8 bytes, si ha che l'occupazione totale in memoria per il formato a coordinate è $(4 + 4 + 8) * \text{nnz}$ bytes. Nel formato pieno l'occupazione di memoria è di 8 bytes per elemento in quanto non vi è necessità di memorizzare gli indici di riga e colonna.

Ad esempio, data la matrice A

$$A = \begin{pmatrix} 1 & 0 & 5 & 8 & 0 \\ 0 & 2 & 6 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 4 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

una sua possibile rappresentazione sparsa potrà essere

```
A = [1,2,3,4,5,6,7,8,9];

J = [1,2,4,2,3,3,4,4,5];

I = [1,2,4,4,1,2,3,1,5];
```

con occupazione di memoria pari a $(4 + 4 + 8) * 9 = 144$ bytes contro i $8 * 25 = 200$ bytes richiesti per la memorizzazione della matrice nel formato pieno.

Per questo tipo di formato non è previsto nessun ordinamento dei vettori `I` e `J` e quindi la ricerca di un generico elemento richiede un tempo proporzionale a $\mathcal{O}(\text{nnz})$ in quanto devono essere controllati tutti gli elementi dei vettori `I` e `J`. MatLab per ridurre il costo di questa operazione ha implementato un albero di ricerca al fine di ridurre il costo computazionale a $\mathcal{O}(\log_2(\text{nnz}))$.

Un vantaggio di questo formato risiede nel fatto che è facile aggiungere un nuovo elemento alla matrice in quanto basta accodarlo alla fine dei vettori `A`, `I` e `J`, previa un'opportuna gestione dinamica della memoria.

MatLab supporta sia le matrici in formato pieno sia le matrici in formato sparso e tutte le operazioni di base su di esse. Negli esempi proposti qui di seguito verrà mostrato come utilizzare le routine di MatLab sulle matrici sparse. In particolare, si ha :

- Esempio 1 : Creazione di una matrice sparsa;
- Esempio 2 : Operazioni di base sulle matrici sparse;
- Esempio 3 : Algebra lineare sulle matrici sparse (autovalori);
- Esempio 4 : Algebra lineare sulle matrici sparse (risoluzione di sistema lineare mediante metodo diretto ed esempi di fattorizzazioni incomplete);

Informazioni sulle matrici sparse in MatLab possono essere ottenuto digitando `help sparfun` o `doc sparfun`.

Esempio 1. — File: EsempioSparse.m

Nel codice riportato viene mostrato come costruire una matrice sparsa a partire dai vettori `I`, `J` *e* `A` *attraverso l'istruzione MatLab* `sparse`*, come passare alla sua rappresentazione in formato pieno attravero l'istruzione* `full` *e come calcolare l'occupazione in memoria dei due formati attraverso l'istruzione* `whos`*.*

Nota 1: *Si osservi che se troviamo due indici uguali nei vettori* `I` *e* `J` *allora i corrispondenti elementi del vettore* `A` *vengono sommati.*

Nota 2: *Il conteggio dell'occupazione in memoria dei due formati potrebbe differire da quello spiegato precedentemente perchè il formato utilizzato da MatLab non è esattamente quello descritto precedentemente, ma una sua variante.*

Nota 3: *Le istruzioni MatLab utilizzate in questo esempio sono:* `sparse`*,* `full` *e* `whos`*.*

```
% Esempio di creazione di una matrice sparsa
% ──────────────────────────────────────────

% Dimensione della matrice (numero di righe e colonne)
m = 5; n = 5;

% Vettore degli elementi della matrice
Av = [1,2,3,4,5,6,7,8,5,4];

% Vettore degli indici di colonna
Jv = [1,2,4,2,3,3,4,4,5,5];

% Vettore degli indici di riga
Iv = [1,2,4,4,1,2,3,1,5,5];

% Creazione della matrice A
A = sparse(Iv,Jv,Av,m,n)

% Risultato
% A =
%    (1,1)        1
%    (2,2)        2
%    (4,2)        4
%    (1,3)        5
%    (2,3)        6
%    (1,4)        8
%    (3,4)        7
%    (4,4)        3
%    (5,5)        9

% Conversione in formato pieno
B = full(A)

% Risultato
% B =
%      1     0     5     8     0
%      0     2     6     0     0
%      0     0     0     7     0
%      0     4     0     3     0
%      0     0     0     0     9

% Visualizzo occupazione in memoria
whos A B

% Risultato
% Name      Size            Bytes  Class
%   A         4x4               144  double array (sparse)
%   B         4x4               200  double array
```

Esercizio 1. *Scrivire un file MatLab con le istruzione per la creazione della matrice sparsa che sia la trasposta dell'esempio precedente.*

Suggerimento: *Per creare la matrice trasposta dell'esempio precedente è sufficiente invertire gli indici di riga e colonna del codice precedente.*

Esempio 2. — **File: MatriciBase.m**
In questo esempio mostriamo come si possono effettuare le operazioni di accesso alla matrice quali, ad esempio, l'accesso ad un elemento o ad un intera riga della matrice ed alcune operazioni base sulle matrici sparse quali la somma, differenza e moltiplicazione di due matrici e la trasposizione.

Nota 1: *La matrice test è già predefinita in MatLab.*

Nota 2: *Le istruzioni MatLab utilizzate in questo esempio sono:* `gallery`, `disp`, `num2str`, `figure`, `spy`, `title`, `,` `(help punct)`, `:` `(help colon)`, `pi`, `speye`, `+` `(help ops)`, `*`, `-` *e* `.'`.

```
% OPERAZIONI BASE SU UNA MATRICE SPARSA
% ---------------------------------------

% Pulizia dello schermo e delle variabili
clear all
close all
clc

% Matrice test formato sparso di MatLab
A = gallery('poisson',4);
% Visualizzo la matrice A in formato pieno
full(A)

% SEZIONE 1 : Informazioni sulla matrice
% ---------------------------------------

% Dimensione
[m,n] = size(A);
disp(['tot. righe   : ',num2str(m)]);
disp(['tot. colonne : ',num2str(n)]);

% Numero di elementi nonzero della matrice
disp(['nnz di A     : ',num2str(nnz(A))]);

% Visuallizzazione del pattern della matrice
figure(1);
spy(A);
title('Pattern della matrice di Poisson');
% Salvataggio della figura in formato postscript
print -f1 -depsc2 'pattern4.eps';
```

```
31
32 % SEZIONE 2 : Accesso agli elementi della matrice
33 % ------------------------------------------------------
34 % Accesso ad un sinsolo elemento: riga 1, colonna 2
35 A(1,2)
36 % Accesso ad un sinsola riga: riga 1
37 A(1,:)
38 % Accesso ad un sinsola colonna: colonna 2
39 A(:,2)
40 % Accesso ad una sottomatrice:
41 A(1:2,1:2)
42
43 % SEZIONE 3 : Operazioni base sulle matrici
44 % ------------------------------------------------------
45
46 % Creazione di una matrice identita' delle stesse dimensioni
47 % di A e moltiplicata per la costa pi.
48 % La sua visualizzazione avviene in formato sparso.
49 M = pi*speye(size(A))
50
51 % Somma ad A la matrice M
52 A = A + M
53
54 % Moltiplico A per M
55 A = A*M
56
57 % Sotraggo a M se stessa (matrice vuota)
58 M = M-M
59
60 % Matrice trasposta
61 A = A.';
```

Esercizio 2. *In questo esercizio viene chiesto di*

1. *Visualizzare l'help della matrice di Poisson utilizzata nel codice precedente (Suggerimento:* `help private/poisson`*);*

2. *Cambiare il parametro N da 2 (esempio corrente) a 4 e visualizzare le dimensioni, il numero di elementi nonzero e pattern di sparsità della matrice A;*

3. *Salvare la figura in un formato a propria scelta.*

Il pattern di sparsità, per $N = 2$ ed $N = 4$ è mostrato in figura **??**.

Esempio 3. — **File: MatriciAlgebra.m**
In questo esempio mostriamo alcune routine di MatLab che implementano

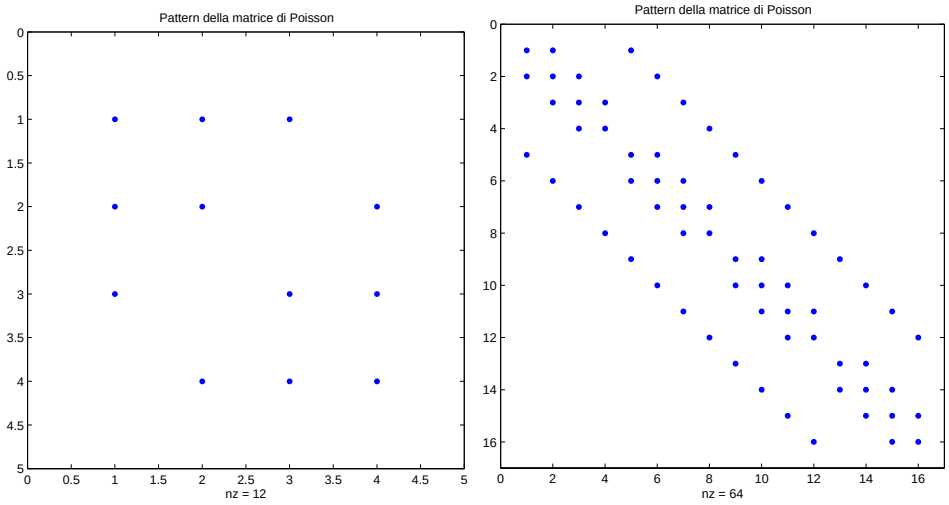

Figura 1.1: Pattern della matrice di Poisson per $N = 2$ (sinistra) ed $N = 4$ (destra).

gli algoritmi di base di algebra lineare. In particolare vedremo come calcolare tutti o alcuni autovalori della matrice.

Nota 1: *La matrice test è già predefinita in MatLab ed è simmetrica e definita positiva.*

Nota 2: *Le istruzioni MatLab utilizzate in questo esempio sono:* `gallery`, `eig`, `flipud`, *e* `eigs`.

```
% ALGEBRA LINEARE NUMERICA
% ------------------------------

% Pulizia dello schermo e delle variabili
clear all
close all
clc

% Matrice test formato sparso di MatLab
A = gallery('poisson',8);

% Calcoliamo tutti gli autovalori
% mediante routine per matrici piene
a = eig(full(A));
% estraiamo i 10 autovalori dominanti
% ed ordiniamo in senso decrescente
ad1 = flipud(a(end-9:end));

```

```
% Calcoliamo i primi 10 autovalori dominanti
% mediante routine per matrici sparse
ad2 = eigs(A,10,'LM');
```

Esercizio 3. *Calcolare i 10 più piccoli autovalori della matrice di Poisson precedente.*

Esempio 4. — **File: MatriciFatt.m**
In questo esempio mostriamo alcune fattorizzazioni che MatLab mette a disposizione. Tali fattorizzazioni saranno poi utilizzate come precondizionatori degli algoritmi sviluppati nelle sezione successive. Inoltre questo esercizio mostra il fenomeno del riempimento di una matrice (fill-in) ovvero se la matrice di partenza si presenta sparsa, non è detto che una sua fattorizzata mantenga la stessa sparsità. In figura **??** *tale fenomeno è messo in evidenza dove sono stati colorati di rosso gli elementi aggiunti.*

Nota 1: *La risoluzione del sistema lineare sparso mediante il comando MatLab* \ *invoca la libreria UMFPACK che implementa un metodo diretto multifrontale.*

Nota 2: *Per i dettagli sulle fattorizzazione mostrate in figura* **??** *si veda* `help cholinc`.

Nota 3: *Le istruzioni MatLab utilizzate in questo esempio sono:* `gallery`, \, `chol` *e* `cholinc`.

```
% FATTORIZZAZIONE DI MATRICI
% ---------------------------

% Pulizia dello schermo e delle variabili
clear all
close all
clc

% Matrice test formato sparso di MatLab
A = gallery('poisson',15);
% Visualizzazione della matrice
figure(1);
spy(A);
title('Matrice A');
print -f1 -depsc2 'matriceA.eps';

% Vettore soluzione di riferimento
xsol = ones(size(A,1),1);

% Vettore dei termini noti
```

```
b = A*xsol;

% Risoluzione del sistema lineare A x = b
x = A\b;

% Confronto della soluzione in nroma 2
norm(x-xsol)

% Fattorizzazione di completa di Cholesky
R1 = chol(A);
% Visualizzazione della fattorizzazione
% Visualizzo in rosso i nuovi elementi prodotti
figure(2);
spy(R1+R1','r');
hold on;
spy(A,'b');
hold off;
title('Fattorizzazione completa di Cholesky');
print -f2 -depsc2 'matriceR1.eps';

% Fattorizzazione di incompleta di Cholesky
R2 = cholinc(A,'0');

% Visualizzazione della fattorizzazione
% Visualizzo in rosso i nuovi elementi prodotti
figure(3);
spy(R2+R2','r');
hold on;
spy(A,'b');
hold off;
title('Fattorizzazione completa di Cholesky - opzione 0');
print -f3 -depsc2 'matriceR2.eps';

% Fattorizzazione di incompleta di Cholesky
R3 = cholinc(A,1e-3);

% Visualizzazione della fattorizzazione
% Visualizzo in rosso i nuovi elementi prodotti
figure(4);
spy(R3+R3','r');
hold on;
spy(A,'b');
hold off;
title('Fattorizzazione completa di Cholesky - droptol 1e-3');
print -f4 -depsc2 'matriceR3.eps';
```

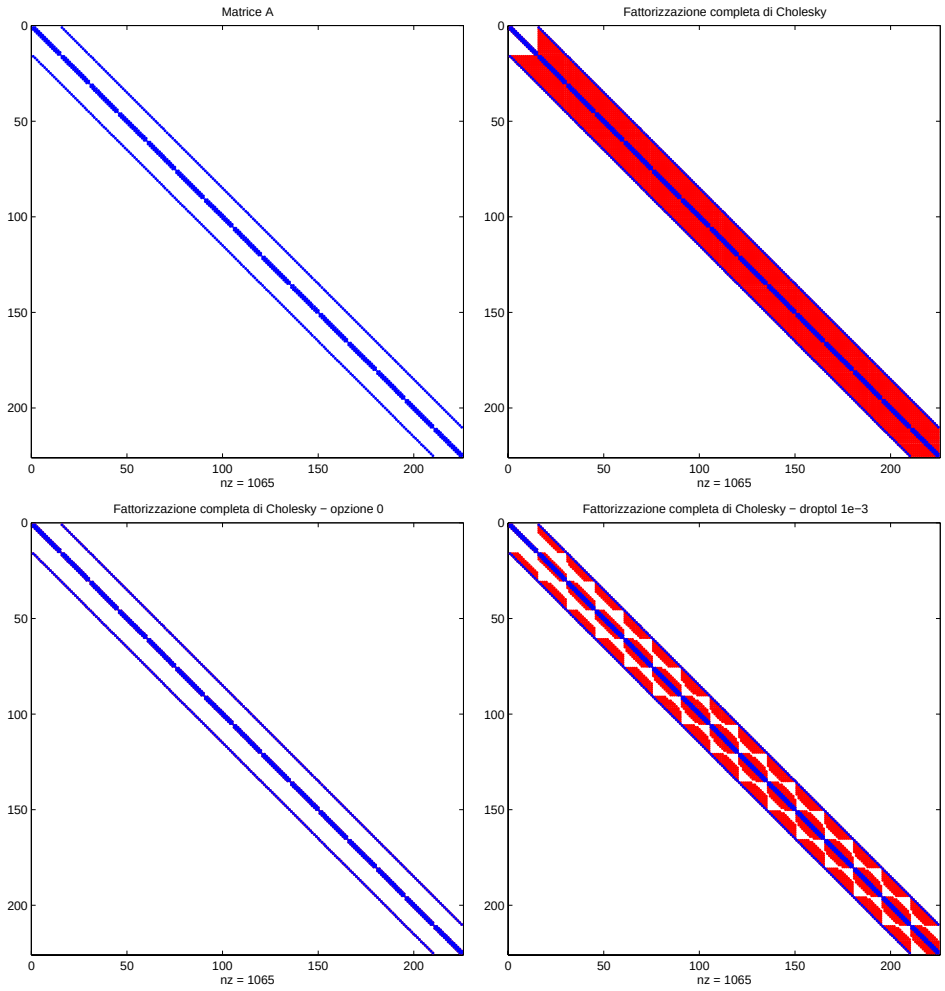

Figura 1.2: Pattern della matrice di Poisson (in alto a sinistra); Pattern fattorizzazione di completa di Cholesky (in alto a destra); Pattern della fattorizzazione incompleta di Cholesky (in basso a sinistra); Pattern della fattorizzazione incompleta di Cholesky con drop (in basso a destra).

1.3 CUTEr

CUTErè un ambiente di testing per l'ottimizzazione e provvede oltre ad una interfaccia C e Fortran anche un interfaccia MatLab per la gestione delle routine del problema. Questa interfaccia avviene attraverso la generazione opportuni MEX-file.

Nella pagina web degli autori viene messo a disposizione i MEX-file dei problemi presentati in questa appendice.

Il primo esempio proposto consiste nel utilizzare questa routine per la creazione di un MEX-FILE di un problema non vincolato e utilizzare le relative routine che CUTEr mette a disposizione per tali tipologie di problemi `usetup`, `unames`, `ufn`, `ugr`, `ush` e `uprod`. Il secondo esempio si riferesce si riferisce ad un problema vincolato e relative routine `DA FARE ELENCO`.

Esempio 5. — File: esunc.m

In questo esempio vediamo alcune istruzioni che CUTEr mette a disposizione per il trattamento di problemi non vincolati. In particolare disegneremo le curve di livello della funzione ed il gradiente della stessa.

Nota 1: *La prima routine che bisogna invocare è* `usetup` *che inizializza il problema.*

Nota 2: *Le istruzioni CUTEr utilizzate in questo esempio sono:* `usetup`.

Nota 3: *Scaricare il file ??.zip.*

```
1  % ESEMPIO DI ROUTINE CUTEr PER L'OTTIMIZZAZIONE NON VINCOLATA
2  format compact
3  clc
4  disp('Unconstraint problem');
5
6  disp('USETUP Set up the data structures for subsequent computations.');
7  [x,bl,bu]=usetup
8
9  disp('UNAMES Get problem name and variable names');
10 [pname,xnames]=unames
11
12 disp('UFN Evaluate objective function at x.');
13 fval = ufn(x)
14
15 disp('UGR Evaluate gradient of objective function at x.');
16 gval = ugr(x)
17
18 disp('USH Evaluate Hessian of objective function at x.');
19 hval = ush(x)
20
```

```
disp('UPROD Evaluate product of vector p with Hessian at x.');
p = x;
uprod(x,p)
% Verifica
hval*p

% Disegna curve di livello
[X,Y] = meshgrid(-2:.1:2, -3:.1:3);
Z = zeros(size(X));
for i=1:size(X,1)
    for j=1:size(X,2)
        Z(i,j) = ufn([X(i,j);Y(i,j)]);
    end
end
figure(1);
contourf(X,Y,Z,30);
colorbar;
print -f1 -depsc2 esunc1.eps;

% Disegna gradiente della funzione
px = zeros(size(X));
py = px;
for i=1:size(X,1)
    for j=1:size(X,2)
        g = ugr([X(i,j);Y(i,j)]);
        px(i,j) = g(1);
        py(i,j) = g(2);
    end
end
figure(2);
contour(X,Y,Z,40);
hold on;
quiver(X,Y,px,py);
hold off;
print -f2 -depsc2 esunc2.eps;

format
```

In figura **??** è stato riportato l'andamento delle curve di livello e del gradiente della funzione.

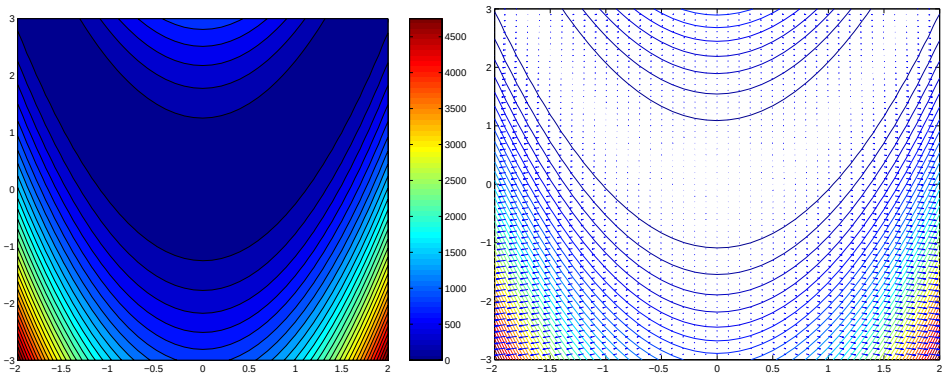

Figura 1.3: Andamento delle curve di livello (sinistra) e del gradiente (destra) per il file `ROSENBR.SIF`.

Esempio 6. — **File: escon.m**
In questo esempio vediamo alcune istruzioni che CUTEr mette a disposizione per il trattamento di problemi vincolati. In particolare vedremo come ...

Nota 1: *La prima routine che bisogna invocare è* `csetup` *che inizializza il problema.*

Nota 2: *Le istruzioni CUTEr utilizzate in questo esempio sono:* `csetup`.

Nota 3: *Scaricare il file ??.zip.*

```
% ESEMPIO DI ROUTINE CUTEr PER L'OTTIMIZZAZIONE VINCOLATA
format compact
clc
disp('Constraint problem');

disp('CSETUP Set up the data structures for subsequent computations.');
options(1) = 1;
options(2) = 1;
options(3) = 0;
[x,bl,bu,v,cl,cu,equatn,linear]=csetup(options)

disp('CNAMES Get problem name, variable names, and constraint names.');
[pname,xnames,gnames]=cnames

disp('CFN Evaluate objective function and general constraint functions at x.');
[f,c]=cfn(x)

disp('CGR Evaluate gradient of the objective or Lagrangian function,and the gra
[g,cjac]=csgr(x)

disp('CSH Evaluate Hessian of the Lagrangian function at x and v.');
h=csh(x,v)

% Disegna curve di livello
[X,Y] = meshgrid(bl(1):.2:bu(1),bl(2):.2:bu(2));
Z = zeros(size(X));
for i=1:size(X,1)
    for j=1:size(X,2)
        [f,c] = cfn([X(i,j);Y(i,j)]);
        Z(i,j) = f;
        if c≤cl(1) || c≥cu(1)
            Z(i,j) = NaN;
        end
    end
end
figure(1);
contourf(X,Y,Z,30);
```

```
38 colorbar;
39
40
41
42 % print -f1 -depsc2 escon.eps;
43
44
45 format
```

Nella figura ?? sono riportate le curve di livello e il disegno del dominio del problema.

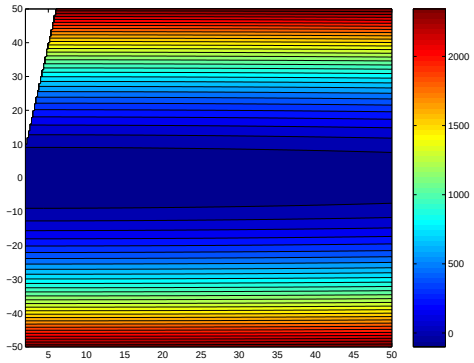

Figura 1.4: Andamento delle curve di livello e dominio del problema vincolato `HS21.SIF`.

1.4 Ottimizzazione non vincolata

1.5 Ottimizzazione vincolata