UNIVERSITÀ
DI PARMA

Progetto di Ricerca Operativa
Approssimazione di funzioni di densitá discreta in AMPL

Esposito Luca
Matricola n. 353830

Indice

1 Descrizione del problema

Il problema affrontato in questo progetto consiste nell'implementazione, in linguaggio **AMPL**, di un problema di minimo, ovvero nel raggiungimento dell'obiettivo di definire una funzione di densitá discreta g che approssima una funzione f, definite su un insieme 1..N; Una funzione di densità discreta è una funzione che associa a ciascun possibile valore di una variabile casuale discreta la probabilità di ottenere quel valore. In particolare g é vincolata ad assumere al massimo n $<$ N valori distinti. Inoltre, la funzione f é definita dall'uguaglianza f(i) = y(i) e con

$$\sum_{i=1}^{N} f(i) = 1 \tag{1}$$

Una volta poste queste condizioni si vuole scegliere g in modo che l'errore assoluto della somma della differenza tra f e g sia il piú piccolo possibile.

2 Modello matematico

INSIEMI

In questo problema il modello utilizzerá solamente uno specifico insieme:

- $\{1..N\}$: Insieme su cui sono definite le funzioni f e g;

VARIABILI

In questo problema le variabili risultano essere:

- *La funzione* ***g***, che essendo una funzione di densitá discreta puó assumere solo valori tra 0 e 1;
- ***error_abs***, variabile utilizzata per la formulazione dell'obiettivo;
- ***isDistinct***, variabile binaria per indicare se g[i] é distintivo. Permette di capire quando il valore successivo della funzione cambia(posizione del "salto");

PARAMETRI

I parametri di questo problema sono:

- *Il parametro* ***N***, che rappresenta la dimensione sia dell'insieme f che dell' insieme g;

- *Il parametro* ***n***, che rappresenta il numero di valori distinti che g puó assumere;
- *Il parametro* ***y(i)***, che rappresenta i valori che assume la funzione f(i);
- *Il parametro* ***f(i)***, che rappresenta la funzione di partenza che viene approssimata dalla funzione g e che puó assumere valori solo tra 0 e 1;

VINCOLI

- ***Somma delle probabilitá:*** Il primo vincolo é un vincolo di normalizzazione imposto su f, ovvero la sommatoria da 1 ad N degli elementi di f deve essere uguale ad 1;

$$\sum_{i=1}^{N} f(i) = 1 \tag{2}$$

- Anche il secondo vincolo é un vincolo di normalizzazione ma in questo caso imposto sulla funzione che approssima f, ***g***, ovvero la sommatoria da 1 ad N degli elementi di g deve essere uguale ad 1;

$$\sum_{i=1}^{N} g(i) = 1 \tag{3}$$

- ***Non-negativitá***: La funzione di densitá discreta deve essere non negativa per tutti i possibili valori della variabile casuale. In altre parole, $\boldsymbol{f(i) \geq 0}$ per ogni $\boldsymbol{i}$;
- Come per la funzione f, anche la funzione di densitá discreta g deve essere non negativa, $\boldsymbol{g(i) \geq 0}$;
- ***Probabilitá di singoli eventi***: La probabilitá di un singolo evento é la probabilitá associata a un singolo valore specifico della variabile casuale. Quindi, $\boldsymbol{0 \leq f(i) \leq 1}$ per ogni $\boldsymbol{i}$;
- Vale lo stesso ragionamento fatto per f anche per la funzione che approssima f, quindi $\boldsymbol{0 \leq g(i) \leq 1}$;
- $\boldsymbol{n \leq N}$: ovvero, i possibili distinti valori di *g(i)* sono n;

OBIETTIVO

- Si vuole scegliere g in modo che l'errore assoluto della differenza tra f e g sia il piú piccolo possibile. Quindi

$$min|f(i) - g(i)| \tag{4}$$

.

3 Modello AMPL

Viene di seguito mostrato il file `.mod` per il linguaggio `AMPL`.

3.1 Density.mod

`Density.mod`

```
param N; # Dimensione dell'insieme di f
param n; # Numero massimo di valori distinti per g

set T := 1..N; # Insieme su cui sono definite le funzioni f e g

param y_i{T}; # Valore che assume la funzione f
param f{i in T} = y_i[i]; # Assegnazione dei valori di y ad f

var g{T} >= 0; # Definizione della variabile g
var isDistinct{i in T} binary; # Variabili binaria utilizzata per indicare se g[i
    ]  distintivo

subject to sum_f: sum{i in T} f[i] = 1; # Vincolo di normalizzazione per f
subject to sum_g: sum{i in T} g[i] = 1; # Vincolo di normalizzazione per g

# Vincolo che permette di capire se il valore della funzione precedente
# rispetto a quella corrente  pi grande o pi piccolo
subject to valisDistinct : sum{i in T} (isDistinct[i]) = n-1;

# Vincolo che se la variabile isDistinct[i] = 1 allora g[i] <= 1,
# se isDistinct = 0 allora g[i] <= g[i-1]
subject to vincoloisDistinctPrec{i in T : i > 1} : g[i] <= g[i-1] + isDistinct[i
    ];

# Vincolo che se la variabile isDistinct[i] = 1 allora g[i] >= 1,
# se isDistinct = 0 allora g[i] >= g[i-1]
subject to vincoloisDistinctSucc{i in T : i > 1} : g[i] >= g[i-1] - isDistinct[i
    ];

var error_abs{T} >= 0; # Variabile per l'errore assoluto

# Vincoli che permettono di rendere la funzione obiettivo lineare
subject to errorConstraint{i in T} : error_abs[i] >= f[i] - g[i]; # Vincolo sull'
    errore assoluto
subject to errorConstraintNeg{i in T} : error_abs[i] >= g[i] - f[i]; # Vincolo
    sull'errore assoluto negato

minimize total_error: sum{i in T} (error_abs[i]); # Minimizzazione dell'errore
    totale
```

Il file modella in linguaggio AMPL quanto visto nell'analogo modello matematico. Per prima cosa vengono dichiarati i parametri che caratterizzano il problema: `N`, `n`, `y_i(T)` ed `f_i(T)`. Successivamente vengono stabiliti gli insiemi del problema, ovvero il solo T. Seguono poi le dichiarazioni delle variabili `g(T)` ed `isDistinct`. Il primo definisce un array bidimensionale avente indici sull'insieme T. Il secondo modella anche'esso un array bidimensionale avente indici sempre sull'insieme T, in cui ogni valore, vincolato ad essere di tipo binario, viene utilizzato per indicare se g[i] è distintivo. Vengono dichiarati poi i due vincoli che caratterizzano il problema, impostando i vincoli di normalizzazione sia per f che per g, ovvero ***sum_f*** e ***sum_g***. Successivamente sono stati definiti tre vincoli per l'approssimazione di f:

- ***valisDistinct***, vincolo che permette di capire se il valore della funzione precedente rispetto a quella corrente é piú grande o piú piccolo;
- ***vincoloisDistinctPrec***, vincolo che se la variabile isDistinct[i] = 1 allora $g(i) \leq 1$, se isDistinct = 0 allora $g(i) \leq g[i-1]$;;
- ***vincoloisDistinctSucc***, vincolo che se la variabile isDistinct[i] = 1 allora $g(i) \geq 1$, se isDistinct = 0 allora $g(i) \geq g[i-1]$;

Invece per quanto riguarda l'obiettivo si é proceduto impostando due vincoli che rendono la funzione obiettivo lineare:

- ***errorConstraint***, secondo cui il valore di error_abs deve essere maggiore o uguale della differenza tra f e g;
- ***errorConstraintNeg***, secondo cui il valore di error_abs deve essere maggiore o uguale della differenza tra g e f (diffrenza negata tra f e g);

Infine si é calcolato l'errore totale(total_error) come la somma di error_abs.

4 Esempi di esecuzione

Vengono ora analizzati alcuni esempi di file `.dat` per il problema.
Per testare l'esecuzione del modello corredato dai suoi dati, si utilizzano dei file `.run` di questo tipo:

`Density_1.run`

```
reset;
option solver gurobi;
model Density.mod;
data Density_1.dat;
solve;
display isDistinct;
display g;
display total_error;
```

`Density_2.run`

```
reset;
option solver gurobi;
model Density.mod;
data Density_2.dat;
solve;
display isDistinct;
display g;
display total_error;
```

Come si puó notare, é stato scelto di mostrare in output il contenuto delle variabili al termine della risoluzione da parte del solver, in modo da poter trarre alcune conclusioni sui risultati ottenuti.

4.1 Density_1.dat

Dopo aver lanciato `Density_1.run` si ottiene il seguente risultato:

```
Gurobi 4.0.1: optimal solution; objective 0.2
224 simplex iterations
39 branch-and-cut nodes
plus 16 simplex iterations for intbasis
isDistinct[*] :=
1 0
2 1
3 1
4 0
5 0
6 0
7 1
8 1
9 0
;

g [*] :=
1 0.01
2 0.19
3 0.11
4 0.11
5 0.11
6 0.11
7 0.28
8 0.04
9 0.04
;

total_error = 0.2
```

Notiamo che il solver trova una soluzione ottima eseguendo 224 iterazioni dell'algoritmo del simplesso con un valore ottimo di 0.2.

4.2 Density_2.dat

Invece per quanto riguarda Density_2.dat, dopo aver lanciato `Density_2.run` si ottiene quest'altro risultato:

```
Gurobi 4.0.1: optimal solution; objective 0.11
331 simplex iterations
34 branch-and-cut nodes
plus 33 simplex iterations for intbasis
isDistinct[*] :=
1 0
2 1
3 0
4 0
5 1
6 1
7 1
8 1
9 0
10 0
11 1
12 0
13 0
14 0
;

g [*] :=
1 0.01
2 0.1
3 0.1
4 0.1
5 0.02
6 0.12
7 0.28
8 0.06
9 0.06
10 0.06
11 0.0225
12 0.0225
13 0.0225
14 0.0225
;

total_error = 0.11
```

In questo caso il solver trova una soluzione ottima eseguendo 331 iterazioni

dell'algoritmo del simplesso, con un valore ottimo di 0.11.