UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Seminario Calcolo Numerico

Anno Accademico 2019/2020

Prof. D'Amore Luisa

Student:

Coppola Vincenzo Matr. M63/1000

Della Torca Salvatore Matr. M63/1011

Indice

Capitolo 1

Introduzione

I metodi ***Multigrid*** nascono come i metodi numerici più veloci per risolvere i problemi al contorno per le equazioni differenziali di tipo ellittico, più comunemente noti come **problemi ai limiti ellittici**.
Un problema al contorno è un'equazione differenziale insieme a dei vincoli, detti **condizioni al contorno**, e una soluzione a tale problema è una soluzione dell'equazione differenziale che rispetta le condizioni al contorno.
I problemi ai limiti ellittici trattano quindi le equazioni differenziali ellittiche, ovvero equazioni del tipo $f = \Delta u$, con Δu somma delle derivate parziali seconde; essi possono essere anche visti come **stato stabile** di un sistema in evoluzione, in quanto sono problemi che non hanno dipendenza dal tempo.
In realtà l'idea che sta alla base dei metodi Multigrid è stata generalizzata e al giorno d'oggi essi vengono applicati in problemi di diverse discipline.
Gli algoritmi basati sui metodi Multigrid sono algoritmi a convergenza rapida, ovvero algoritmi la cui complessità computazionale è sempre lineare anche all'aumentare della complessità del problema, e la loro efficienza dipende da come si adattano i componenti che utilizzano ai vari problemi che risolvono.
Consideriamo ad esempio un semplice problema in cui si vogliono distribuire ***N+1*** campioni lungo una distanza ***L*** in modo tale che siano tutti equidistanti gli uni dagli altri.
Per prima cosa si possono iniziare a numerare i campioni da *1* a N e si possono posizionare i campioni 1 e N alle estremità di L.
A questo punto si potrebbe pensare di inserire ciascun campione i, per $i = 2, \ldots, N$, ad una distanza $\frac{iL}{N}$ dal campione 1.
Questo processo così definito è un processo **globale** di risoluzione del problema, nel senso che risolverebbe direttamente il problema e richiederebbe un alto grado di raffinatezza per l'esecuzione dei calcoli matematici delle distanze.

Una tecnica più efficiente potrebbe essere quella di inserire un terzo campione esattamente a metà della distanza dei primi due e proseguire poi in questo modo distribuendo ogni campione esattamente nel punto centrale tra i suoi due vicini.
Questo processo è un processo di risoluzione **locale** del problema ed è molto più efficiente rispetto ad un metodo globale perchè riesce a risolvere il problema in un numero di passi $O(logN)$.
Consideriamo un sistema lineare $Ax = b$, dove la matrice **A** è *simmetrica*, *tridiagonale a blocchi* e tale che i blocchi su ciascuna diagonale sono uguali.

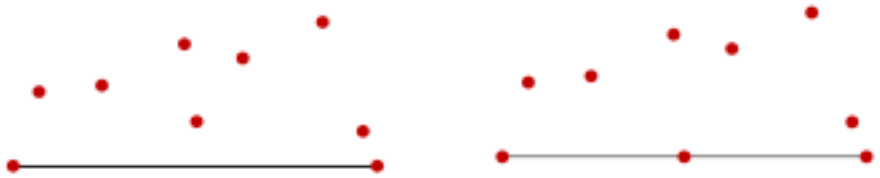
$$A = \begin{pmatrix} D & F & 0 & \dots & 0 \\ F & D & F & \dots & 0 \\ 0 & F & D & \dots & 0 \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 0 & 0 & 0 & F & D \end{pmatrix}$$

Queste matrici così definite sono matrici *sparse* (con molti elementi nulli), che derivano dalla discretizzazione di equazioni differenziali e pertanto la costruzione di queste matrici dipende dal passo di discretizzazione: più il passo di discretizzazione tende a 0, più lo schema di discretizzazione è accurato, cioè l'errore di discretizzazione tende a 0.
Se consideriamo allora un *metodo iterativo* per la risoluzione di questo sistema, si può osservare che la velocità di convergenza peggiora al decrescere di h, dove h è il *passo di discretizzazione*, cioè la distanza fra i punti di una griglia.
Ad esempio, supponendo di essere in 2D *($n = N^2$)*:

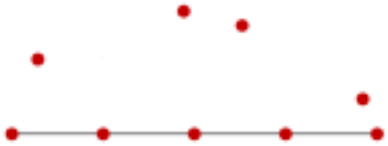
- la fattorizzazione **LU** applicata ad una matrice *densa* ha complessità $O(n^3)$;
- la fattorizzazione **LU** applicata ad una matrice a *banda* (matrice sparsa i cui elementi diversi da zero sono tutti posti in una banda diagonale che comprende la diagonale principale e, opzionalmente, una o più diagonali alla sua destra o alla sua sinistra) ha complessità $O(n^2)$;
- l'algoritmo FFT ha complessità $O(n \cdot logn)$.

Questo fenomeno è dovuto al fatto che l'***indice di condizionamento*** di queste matrici aumenta al decrescere di h, cioè per $h \to 0$ la matrice A diventa *mal condizionata* perchè $\mu(A) = \frac{1}{O(h^2)}$.

Infatti il condizionamento influenza la velocità di convergenza perchè, indicato con δ il *fattore di convergenza*, tale che $0 \leq \delta \leq 1$, allora $e^{k+1} \leq \delta e^k$.
Nella maggior parte dei metodi iterativi il fattore δ dipende dal passo di discretizzazione h ed è per questo che la velocità peggiora al decrescere di h, mentre negli algoritmi Multigrid δ è indipendente da h e quindi si riesce ad avere un bilanciamento tra il passo di convergenza e l'errore di troncamento introdotto dalla discretizzazione.
Quindi il Multigrid riesce a raggiungere la *complessità asintotica lineare* che gli altri metodi non sono in grado di raggiungere.
L'idea principale del Multigrid è quella di accelerare la convergenza di un metodo iterativo di base, noto come rilassamento, che in genere riduce l'errore mediante una correzione globale dell'approssimazione della soluzione a *griglia fine* ottenuta mediante approssimazioni successive a partire da un problema *coarse*. Questo processo ricorsivo si ripete fino a quando non si raggiunge una griglia dove il costo della soluzione diretta è trascurabile rispetto al costo di una soluzione rilassata sulla griglia fine.
Questo ciclo Multigrid riduce tipicamente tutti i componenti di errore di una quantità fissa indipendentemente dalla dimensione delle maglie della griglia fine.
L'applicazione tipica per il multigrid è nella soluzione numerica delle equazioni differenziali parziali ellittiche in due o più dimensioni.
I metodi Multigrid possono essere applicati in combinazione con qualsiasi tecnica di discretizzazione comune e sono tra le tecniche di soluzione più veloci oggi conosciute; a differenza di altri metodi, sono generali in quanto possono trattare regioni e condizioni al contorno arbitrarie e non dipendono dalla separabilità delle equazioni o da altre proprietà speciali dell'equazione.
Sono stati ampiamente utilizzati anche per sistemi di equazioni non simmetriche e non lineari più complessi, come le *equazioni di elasticità di Lamé* o le *equazioni di Navier-Stokes*, che sono un sistema di tre equazioni di bilancio *(equazioni alle derivate parziali)* che descrivono un fluido viscoso lineare. Si tratta di tre equazioni di bilancio della meccanica dei continui, in cui sono introdotte come leggi costitutive del materiale la legge di Stokes sull'attrito viscoso e la legge di Fourier per il calcolo della quantità di calore che viene trasferita per conduzione in corpo.
L'idea principale alla base dei metodi multigrid è quella di utilizzare un processo locale per tutte le scale del problema.
Tornando all'esempio della distribuzione che abbiamo visto in precedenza, si supponga che N sia una potenza di due. Nel nostro esempio si parte con una visione su "larga scala" in cui il problema è costituito solamente dal campione 1, dal campione N e dal campione $\frac{N}{2}$, che sarà posizionato esattamente a metà della distanza **L** tra 1 e N.



Successivamente, con una visione su "scala intermedia", i campioni $\frac{N}{4}$ e $\frac{3N}{4}$ vengono spostati esattamente a metà strada tra i loro *"vicini di media scala"*.



Si procede in questo modo fino a risolvere il problema su "piccola scala" così da ottenere la soluzioni in soli $logN$ passi.

Questo approccio è definito approccio *multiscala* e a primo impatto sembrerebbe essere molto semplice ma in realtà nasconde molte difficoltà. Lo sviluppo di un risolutore multiscala per un determinato problema richiede di:

- scegliere un processo locale appropriato;
- scegliere variabili *coarse* (su larga scala) appropriate;
- sviluppare equazioni o processi appropriati per le variabili *coarse*;
- scegliere metodi appropriati per il trasferimento delle informazioni da una scala ad un'altra.

A seconda dell'applicazione, ognuno di questi compiti può essere semplice o complesso e questo è in parte il motivo per cui il multigrid continua ad essere un campo di ricerca molto attivo.

Capitolo 2

Metodi MULTIGRID

Le idee che sono alla base degli algoritmi Multigrid sono molteplici:

- raffinamento iterativo della soluzione di un sistema lineare;
- griglie innestate;
- procedimento di economizzazione di Lanczos;
- algoritmo *divide et impera*;

2.1 Raffinamento iterativo della soluzione

Consideriamo il sistema $Ax = b$ e indichiamo con x_{calc} la soluzione calcolata.
Sappiamo che il *residuo* può essere calcolato come $r = Ax_{calc} - b$. Se andiamo allora a moltiplicare la matrice A per l'errore $e = x_{calc} - x$ quello che si ottiene è

$$A(x_{calc} - x) = Ax_{calc} - Ax = Ax_{calc} - b = r$$

Questa equazione $A \cdot e = r$ è detta *equazione del residuo* e risolvendola si ricava l'errore e necessario per raffinare la soluzione calcolata $x_{agg} = e + x_{calc}$.
Un metodo di raffinamento iterativo è il *metodo di Jacobi*.

2.1.1 Metodo di Jacobi

Il metodo di Jacobi è un metodo iterativo per la risoluzione di sistemi lineari; metodo iterativo perchè calcola la soluzione di un sistema di equazioni lineari dopo un numero *teoricamente* infinito di passi.
Per calcolare tale soluzione il metodo utilizza una successione $x^{(k)}$ che converge verso la

soluzione esatta del sistema lineare e ne calcola progressivamente i valori arrestandosi quando la soluzione ottenuta è sufficientemente vicina a quella esatta.
L'idea su cui si basa il metodo di Jacobi è che, dato il sistema $Ax = b$, scrivendo la matrice A come differenza di due matrici $A = M - N$, dove M è una matrice non singolare con determinante non nullo, e quindi *invertibile*, allora la soluzione x di $Ax = b$ risolve anche l' equazione

$$Mx = Nx + b$$

Il metodo di Jacobi è basato su un algoritmo che, a partire da un qualunque vettore x_0, costruisce una successione di vettori x_k tale che

$$x^{k+1} = M^{-1}(Nx^k + b)$$

e se questa successione converge ad un vettore x, allora $Ax = b$.
Per misurare la distanza dei termini della successione x_k dalla soluzione e controllare se la successione converge, si può considerare un vettore

$$e^k = x^k - x$$

tale che la successione dei vettori differenza è data dalla ricorsione

$$e^{k+1} = x^{k+1} - x = M^{-1}(Nx^k + b) - M^{-1}(Nx + b) = M^{-1}N(x^k - x) = (M^{-1}N)e^k$$

ovvero

$$e^k = (M^{-1}N)^k e^0$$

L'algoritmo converge se e solo se la successione delle differenze e^k tende al vettore nullo. La convergenza è garantita, indipendentemente dalla scelta iniziale di x_0, se e solo se tutti gli autovalori di $B = M^{-1}N = M^{-1}A - I$ hanno norma inferiore a 1, ovvero se il valore massimo tra i moduli degli autovalori è minore di 1.
Si può dimostrare che se la matrice A è una matrice a *diagonale dominante* allora l'algoritmo converge.
Il metodo di Jacobi consiste quindi nell'applicare l'algoritmo utilizzando come matrice M una matrice *diagonale* D con la stessa diagonale della matrice A. Affinchè ciò sia possibile è però necessario che la matrice A non presenti elementi nulli sulla diagonale, perchè D deve essere invertibile. Nel caso in cui la matrice A dovesse avere elementi nulli sulla diagonale si eseguirebbe prima una *permutazione* mediante una matrice P e quindi si otterrebbe il sistema

$$PAx = Pb$$

Quindi ponendo $M = D$ la formula ricorsiva diventa

$$x^{k+1} = -D^{-1}((A - D)x^k - b)$$

che associata ad un sistema di equazioni lineari corrisponde ad isolare una variabile per ogni riga del sistema.

$$\begin{cases} a_{1,1}x + a_{1,2}y + a_{1,3}z = b_1 \\ a_{2,1}x + a_{2,2}y + a_{2,3}z = b_2 \\ a_{3,1}x + a_{3,2}y + a_{3,3}z = b_3 \end{cases} \Leftrightarrow \begin{cases} a_{1,1}x = -a_{1,2}y - a_{1,3}z + b_1 \\ a_{2,2}y = -a_{2,1}x - a_{2,3}z + b_2 \\ a_{3,3}z = -a_{3,1}x - a_{3,2}y + b_3 \end{cases} \Leftrightarrow \begin{cases} x = -\frac{a_{1,2}}{a_{1,1}}y - \frac{a_{1,3}}{a_{1,1}}z + \frac{b_1}{a_{1,1}} \\ y = -\frac{a_{2,1}}{a_{2,2}}x - \frac{a_{2,3}}{a_{2,2}}z + \frac{b_2}{a_{2,2}} \\ z = -\frac{a_{3,1}}{a_{3,3}}x - \frac{a_{3,2}}{a_{3,3}}y + \frac{b_3}{a_{3,3}} \end{cases}$$

Per cui la ricorsione sui vettori è definita come

$$\begin{cases} x_{k+1} = -\frac{a_{1,2}}{a_{1,1}}y_k - \frac{a_{1,3}}{a_{1,1}}z_k + \frac{b_1}{a_{1,1}} \\ y_{k+1} = -\frac{a_{2,1}}{a_{2,2}}x_k - \frac{a_{2,3}}{a_{2,2}}z_k + \frac{b_2}{a_{2,2}} \\ z_{k+1} = -\frac{a_{3,1}}{a_{3,3}}x_k - \frac{a_{3,2}}{a_{3,3}}y_k + \frac{b_3}{a_{3,3}} \end{cases}$$

In generale la ricorsione si può esprimere come

$$x_i^{k+1} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}$$

Il metodo di Jacobi richiede di tenere in memoria almeno due vettori, ma i calcoli possono essere svolti in parallelo sulle componenti. Una volta calcolata la matrice B, per ogni iterazione il calcolo di ognuna delle n componenti richiede $n - 1$ moltiplicazioni e $n - 1$ somme.

2.2 Griglie Innestate

Consideriamo un intervallo [0, T] e un numero di campioni N equidistanti tra loro: si indica con Ω^h la **griglia fine**, cioè quella ottenuta discretizzando l'intervallo con un passo di campionamento $h = \frac{1}{N}$, mentre si indica con Ω^{2h} la **griglia coarse**, ossia quella ottenuta con un passo di discretizzazione $h = \frac{2}{N}$; questo vuol dire che i nodi della griglia coarse corrispondono ai nodi della griglia fine che sono a distanza $2h$.

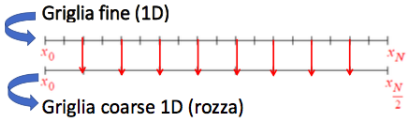


Fig. 2.1: Griglia fine e griglia coarse in una dimensione

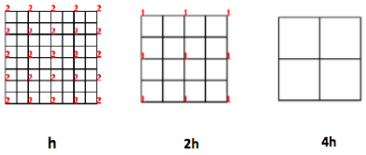


Fig. 2.2: Griglia fine e griglie coarse in due dimensioni

Combinando il *raffinamento iterativo* con le *griglie innestate* si ottiene un primo schema Multigrid detto **coarse grid correction**.
Si parte da una griglia fine, quindi da un sistema $A_h x_h = b_h$, e si esegue un rilassamento n_1 *volte* sulla griglia Ω^h con una data ipotesi iniziale così da ottenere un'approssimazione $\tilde{x}_h$. Si esegue poi una *restrizione*: si calcola il *residuo* $r_h = b_h - A\tilde{x}_h$ e lo si proietta sulla griglia coarse Ω^{2h} ottenendo $r_{2h} = I_h^{2h} r_h$, dove I_h^{2h} è l'operatore di restrizione.
Una volta passati allo *spazio coarse*, si esegue un rilassamento sull'equazione $A_{2h} e_{2h} = r_{2h}$ sulla griglia Ω^{2h} per ottenere un'approssimazione dell'errore $\tilde{e}_{2h}$ e tramite un operatore di interpolazione I_{2h}^h si ritorna allo spazio fine calcolando l'errore $\tilde{e}_h = I_{2h}^h \tilde{e}_{2h}$.
Quest'errore ci consente di correggere l'approssimazione ottenuta sulla griglia Ω^h:
$\tilde{x}_h \leftarrow \tilde{x}_h + \tilde{e}_h$. A questo punto si esegue un rilassamento n_2 volte su $A_h x_h = b_h$ sulla griglia Ω^h per ottenere il risultato corretto.

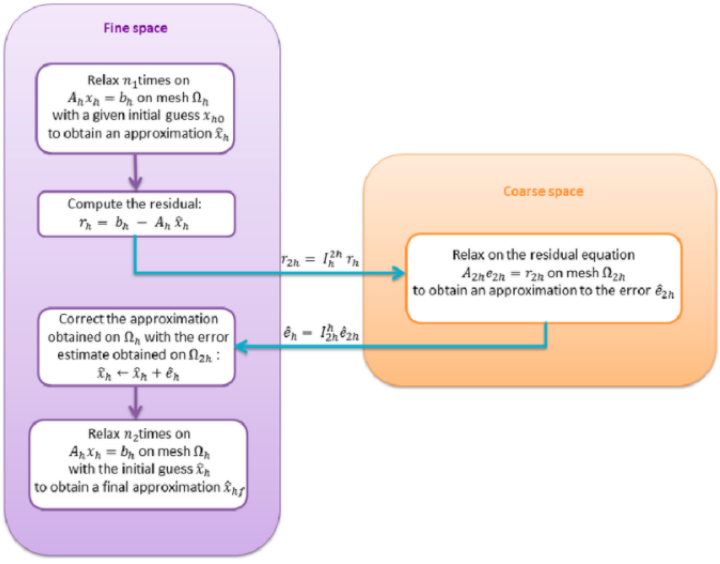


Fig. 2.3: Coarse Grid Correction

2.3 Procedimento di economizzazione di Lanczos

Dato un problema P l'idea è quella di risolvere non il problema P ma un problema P_1 la cui soluzione S_1 è una buona approssimazione della soluzione S del problema P. Iterativamente, invece di risolvere il problema P_1, si risolve il problema P_2 la cui soluzione S_2 è una buona approssimazione della soluzione S_1 del problema P_1, e cosi via.
Si ottiene quindi una sequenza di sottoproblemi e la stima dell'errore sulla soluzione S di P si ottiene facendo una stima degli errori dei sottoproblemi. In generale, se si hanno n sottoproblemi, e quindi n sottosoluzioni S_i, allora

$$S - S_n = S - S_1 + S_1 - S_2 + S_2 + \cdots - S_n$$

da cui, applicando la disuguaglianza triangolare, si ottiene

$$||S - S_n|| < ||S - S_1|| + ||S_1 - S_2|| + \cdots + ||S_{n-1} - S_n||$$

Il procedimento di economizzazione di Lanczos porta ad un secondo schema Multigrid detto **nested iterations**.
Sia Ω un dominio poligonale, ovvero un dominio la cui frontiera è fatta da segmenti. Tipicamente la frontiera di un dominio è una curva arbitraria, ma supponiamo di poterla approssimare con una spezzata fatta di segmenti sufficientemente piccoli che non fanno ridurre troppo l'accuratezza.
Supponiamo ora di voler risolvere un sistema lineare del tipo $A_k z = g$, dove A_k è l'approssimazione discreta di un sistema differenziale, che caratterizza il problema da risolvere tramite una griglia T_k su Ω. Utilizzeremo K sistemi ausiliari del tipo $A_i \cdot x_i = b_i$, con i=1,...,k-1, che sono discretizzazioni su una gerarchia di griglie dello stesso problema continuo *(un esempio è il dominio* $[0,1]^2$ *e le griglie quadrate che si ottengono dividendo i lati in 2, 4, 8, 16 parti)*.
A tale problema verrà applicato l'algoritmo di *Nested Iteration (NI)* di cui mostriamo uno pseudo-codice trovato in rete.

```
begin
  Choose   m_k ∈ ℕ, k = 1, ..., K;
  x := A_1^{-1} b_1;
  for k := 1, K do
    x := I_{k-1}^k x; { x is prolonged }
    for i := 1, m_k do
      x := S(A_k, x, b);
    endfor
  endfor
  return   x;
end
```

Fig. 2.4: Pseudo codice algoritmo NI

L'operatore S(A_k, x, b) rappresenta un solutore di sistemi lineari; assumeremo in particolare che

$$x_k^{(i+1)} = S(A_k, x_k^{(i)}, b_k)$$

mentre l'errore al passo successivo

$$e_k^{(i+1)} = |x - x_k^{(i+1)}| \simeq \mu_k |x - x_k^{(i)}| = \mu_k e_k^{(i)}$$

ovvero l'errore al passo successivo è direttamente proporzionale all'errore mediante una costante μ_k; si parla di *convergenza lineare*.
L'operatore I_{k-1}^k è un operatore di prolungamento, o di interpolazione, che permette di interpolare una funzione da una griglia più grossolana (*coarse*) ad una più fine. Tale algoritmo può essere usato per risolvere il sistema $A_k z = g$, visto precedentemente, quando i k sistemi ausiliari hanno p_i incognite $p_1 < ... < p_k$.
Se ogni metodo risolutivo S ha costante μ_k, ed esiste un valore μ, compreso tra μ_k e 1 per $k = 1, \ldots, K$, tali che per $\mu_k = \mu$ l'algoritmo NI converge, allora l'algoritmo ha complessità ottimale ed è chiamato *metodo multigrid*. Quindi il *Nested Iterations* esegue alcuni passi del metodo iterativo sulla griglia più coarse possibile e poi usa l'approssimazione ottenuta come valore iniziale per applicare alcuni passi sulla griglia più fine. *La soluzione del problema P_{i-1} rappresenta l'approssimazione iniziale per il calcolo della soluzione P_i.*

2.4 Divide et Impera

Mettendo insieme il concetto di *coarse grid correction* e *nested iterations* si ottiene un terzo schema Multigrid che è lo schema **ricorsivo**, noto anche come *Divide et Impera* in quanto consiste nell'eseguire prima una restrizione da *griglia fine* a *griglia coarse*, fino ad ottenere la griglia più coarse possibile, e poi un'interpolazione che permette di ottenere la soluzione corretta.
Lo schema che descrive questo procedimento è detto ***V-Cycle***.
Per trasferire informazioni tra le griglie, abbiamo bisogno di definire gli operatori adatti. Consideriamo solo due griglie: la griglia fine, con dimensione h e con $N - 1$ punti, e la griglia coarse, con dimensione $2h$ e $\frac{N}{2} - 1$ punti.
Si definisce:

- un operatore di restrizione, indicato con I_h^{2h}, per il trasferimento dei dati da griglia fine a coarse;
- un operatore di prolungamento *(interpolazione)*, indicato con I_{2h}^h per il trasferimento di dati da griglia coarse a fine *(tipicamente si sceglie l'interpolazione lineare)*;

- un operatore sulla griglia coarse, indicato con L^{2h}, che approssima l'L^h della griglia fine *(una scelta comune è quella di discretizzare semplicemente l'equazione differenziale sulla griglia coarse usando uno schema simile a quello applicato sulla griglia fine).*

Una volta definiti questi operatori possiamo definire il *V-Cycle.*

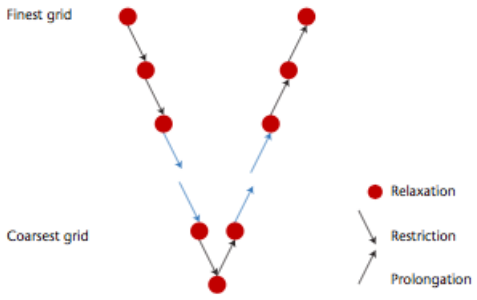


Fig. 2.5: Si procede da sinistra a destra

Consideriamo due parametri, v_1 e v_2 che rappresentano il numero di rilassamenti di Jacobi fatti prima e dopo la correzione della griglia coarse: l'algoritmo V-Cycle rilassa v_1 volte ogni griglia (eccetto la più coarse possibile) per ottenere una griglia più coarse e poi interpola v_2 volte per aggiungere la correzione e determinare il risultato corretto; si opera quindi in maniera *ricorsiva.*
Generalmente però, il V-Cycle non risolve il problema esattamente e necessita di essere applicato iterativamente; tuttavia ad ogni iterazione tipicamente l'errore si riduce drasticamente.

Algorithm V-Cycle: $u^h = \text{VCycle}(u^h, h, N, f^h, v_1, v_2)$
1. If $N == 2$, solve $L^h u^h = f^h$ and return u^h;
2. $u^h = \text{Relax}(u^h, h, N, f^h, v_1)$;
   relax v_1 times.
3. $r^h = f^h - L^h u^h$;
   compute residual vector.
4. $f^{2h} = I_h^{2h} r^h$;
   restrict residual.
5. $u^{2h} = 0$;
   set u^{2h} to zero, including boundary conditions.
6. $u^{2h} = \text{VCycle}(u^{2h}, 2h, N/2, f^{2h}, v_1, v_2)$;
   treat coarse-grid problem recursively.
7. $u^h = u^h + I_{2h}^h u^{2h}$;
   interpolate and add correction.
8. $u^h = \text{Relax}(u^h, h, N, f^h, v_2)$;
   relax v_2 times;
9. Return u^h.

Fig. 2.6: Algoritmo V-Cycle

N.B: La prima istruzione verifica che $N = 2$ perchè in tal caso si ha già la griglia più coarse possibile e quindi bisogna risolvere il problema.

Il numero di operazioni eseguite nel V-Cycle su ogni griglia è proporzionale al numero di punti della griglia. Poiché il numero di variabili in ogni griglia è approssimativamente una frazione costante del numero di variabili della griglia successiva più fine (una metà in 1D e un quarto in 2D), il numero totale di operazioni per il V-Cycle è maggiore di un fattore costante del numero di operazioni eseguite solo sulla griglia più fine.
Infine, si può osservare che su ogni griglia della gerarchia le operazioni di rilassamento, prolungamento e restrizione possono essere eseguite in parallelo cosi che il numero di passi sequenziali richiesti dal V-Cycle sia proporzionale al numero di griglie, che è semplicemente $O(logN)$.
Vogliamo ora capire come determinare la migliore ipotesi iniziale possibile. Un approccio molto efficace è quello di risolvere approssimativamente il problema su una griglia più coarse e di interpolare questa soluzione ottenuta alla griglia fine da usare come prima approssimazione. Questo approccio viene applicato in modo ricorsivo, ottenendo l'algoritmo *Full Multigrid(FMG)*.
Abbiamo bisogno di un parametro positivo μ, che indichi il numero di V-Cycle applicati ad ogni livello dell'algoritmo.

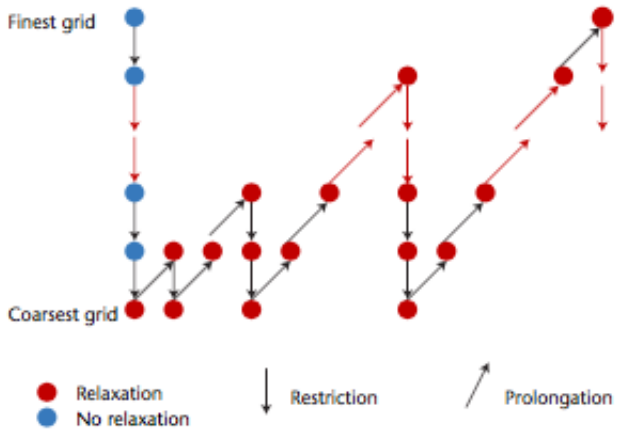


Fig. 2.7: Full Multigrid per $\mu = 1$

L'algoritmo inizialmente trasferisce il problema dalla griglia più fine alla griglia più coarse, dopo di che risolve il problema sulla griglia più coarse e interpola questa soluzione alla seconda griglia più coarse ed esegue un V-Cycle *(tipicamente si usa l'interpolazione cubica)*. Questi due step sono ripetuti ricorsivamente a griglie sempre più fini, terminando con la griglia più fine.

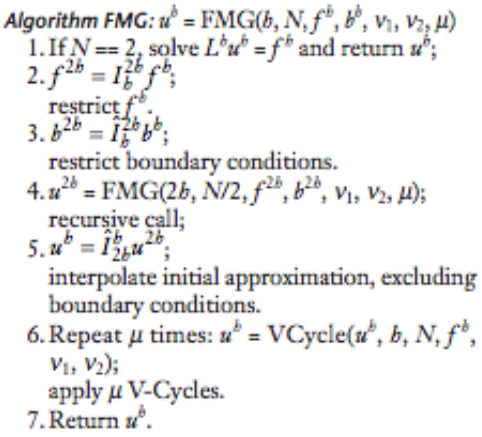


Fig. 2.8: Algoritmo FMG

Come sappiamo esistono diversi tipi di errori tra i quali abbiamo:

- errore algebrico, che è la differenza fra la soluzione discreta e la sua approssimazione ottenuta;
- errore di discretizzazione, che è la differenza tra la soluzione dell'equazione differenziale (campionata sulla griglia) e quella del sistema discreto.

L'errore totale è la somma di questi due errori, cioè la differenza tra la soluzione dell'equazione differenziale (campionata sulla griglia fine) e l'attuale approssimazione discreta.
Una volta che l'errore algebrico diventa più piccolo dell'errore di discretizzazione, quest'ultimo comincia a dominare l'errore totale, per cui sarebbe utile fare ulteriori sforzi per ridurlo.
Per molti problemi, i ricercatori hanno dimostrato che l'algoritmo FMG può produrre piccoli errori algebrici rispetto all'errore di discretizzazione anche con solo $\mu = 1$ e la complessità di calcolo dell'algoritmo FMG è lineare (un algoritmo FMG con $\mu = 1$ costa circa il doppio di un V-Cycle in 1D e circa uno e un terzo in 2D).
Se consideriamo la ***work unit (WU)***, ossia la quantità di calcolo necessaria per un singolo rilassamento sulla griglia più fine o per un calcolo del residuo, l'algoritmo FMG fornisce una soluzione accurata al problema al costo di poche WU, che è in genere una decina di operazioni per ogni variabile a griglia fine.

Capitolo 3

Griglie triangolari

Una *triangolazione* del dominio Ω è una suddivisione fatta di triangoli, in cui nessun vertice di un triangolo cade su un lato di un altro triangolo. I lati dei triangoli vengono detti anche *lati della triangolazione*.
Supponiamo ora di avere un insieme di K triangolazioni T_i, con $i = 1, \ldots, K$: chiameremo T_i *mesh a livello* i.
Assumiamo che le mesh siano raffinamenti regolari della mesh iniziale T_1, ovvero ogni triangolo di T_{i+1} si ottiene unendo i punti medi dei lati di T_i; se quest'ultima ha diametro $diam_i$, p_i nodi e t_i elementi, si ottiene una mesh T_{i+1} che ha $t_{i+1} = 4t_i$ elementi e diametro $diam_{i+1} = diam_i/2$.
I triangoli di ogni mesh raffinata sono simili a quelli della mesh di partenza.

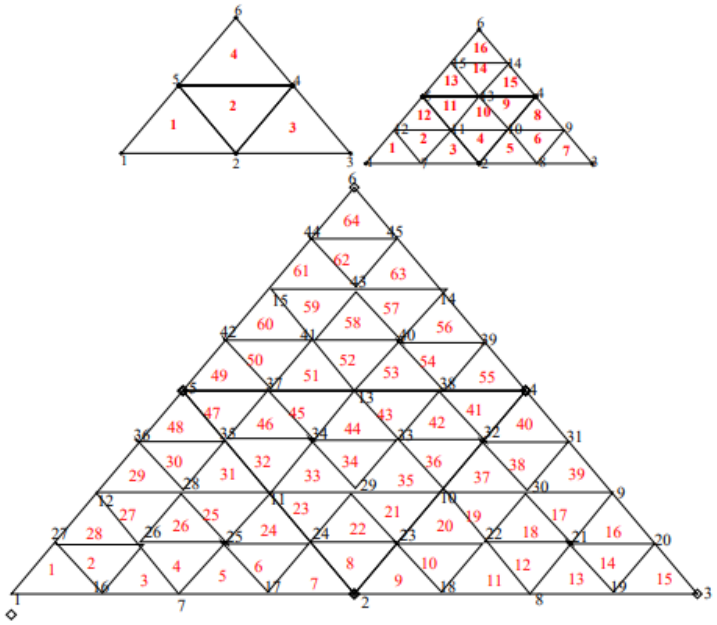


Fig. 3.1: Esempio di mesh triangolare con due raffinamenti regolari

Ora possiamo dire che una famiglia di mesh triangolari T_k sul dominio Ω, ognuna con diametro d_i, è quasi uniforme se esiste un $\rho > 0$ tale che:

$$max\{diam(B_T) : T \in T_K\} \geq \rho \cdot diam_k diam(\Omega) \quad \forall diam_k \in (0, 1]$$

con B_T la più grande sfera contenuta in T.
Sia V_i lo spazio delle funzioni $\phi \in C^0$ lineari a tratti su T_i e nulle su $\partial\Omega$ e sia $n(T_k)$ l'insieme dei nodi della k-esima mesh tali che

$$n(T_i) \supset n(T_j) \implies V_i \supset V_j, \quad 1 \leq i \leq j \leq K$$

In ogni caso il fatto che ogni $v \in V_i$ sia nulla in $\partial\Omega$ non è limitante solo se il problema da risolvere è di Dirichlet, ovvero un problema che richiede di trovare una funzione che soddisfi una determinata equazione alle derivate parziali all'interno di una regione sulla cui frontiera la funzione assume determinati valori al contorno.
Sia inoltre N_k il numero di nodi $P_{\psi_k(i)}$ interni a T_k, con i=1,...,N_k e con $\psi_k : \{1, \cdots, N_K\} \to \{1, \cdots, p_K\}$; definiamo il prodotto interno su V_k dipendente dalla mesh (*mesh–dependent inner product*) come

$$(v, w)_k = diam_k^2 \sum_{i=1}^{N_k} v(P_{\psi_k(i)}) w(P_{\psi_k(i)})$$

Definiamo infine gli operatori di interpolazione e restrizione che useremo in seguito:

- $I_{k-1}^k : V_{k-1} \to V_k$ è l'operatore di interpolazione che ci permette di passare da una griglia coarse ad una fine, e

$$I_{k-1}^k v = v, \quad \forall v \in V_{k-1}$$

- $I_k^{k-1} : V_k \to V_{k-1}$ è l'operatore di restrizione che ci permette di passare da una griglia fine a una coarse, ed è definito come il trasporto di I_k^{k-1} rispetto ai prodotti interni $(\cdot, \cdot)_{k-1}$, $(\cdot, \cdot)_k$, ovvero:

$$(I_k^{k-1} w, v)_{k-1} = (w, I_{k-1}^k v)_k = (w, v)_k \quad \forall v \in V_{k-1}, w \in V_k$$

Se la restrizione è il trasporto del prolungamento, è più facile dimostrare che il metodo multigrid converge, ma non sempre accade: in tali casi non vengono date dimostrazioni di convergenza, ma ci si limita ad una validazione dell'algoritmo su problemi di test.

Capitolo 4

Applicazioni del metodo multigrid

In questo capitolo mostriamo due applicazioni del metodo multigrid: innanzitutto mostriamo un algoritmo "generale" che usa tale metodo, e in seguito mostriamo degli algoritmi multilivello che servono a generare griglie coarse per i metodi multigrid.

4.1 Algoritmo generale

Lo schema multigrid usa due strategie generali:

- smussamento su T_i: vengono smussate le componenti ad alta frequenza dell'errore;
- correzione dell'errore su T_{i-1} usando uno schema iterativo sviluppato proprio su quest'ultimo spazio: vengono smussate le componenti a bassa frequenza del rumore.

Descriviamo nel dettaglio un algoritmo multigrid generale, su K griglie triangolari $T_1 \subset \cdots \subset T_K$, implementato dalla funzione ricorsiva *MG(k,z,g)*; le considerazioni che faremo su tale funzione valgono, con opportune modifiche, anche su griglie quadrate e rettangolari.
Di seguito mostriamo uno pseudo-codice della funzione *MG* trovato in Internet.

```
MG(k, z, g)
begin
  if k = 1
    then
      z := A_1^{-1} g;
    else
      z := I_{k-1}^k z; { z is prolonged }
      for s := 1, s̄ do
        for l := 1, m_1 do
          z := S(A_k, z, g);
        endfor
        g := I_k^{k-1} r(A_k, z, g); { r is restricted }
        q := 0;
        for i := 1, p do
          q := MG(k − 1, q, g);
        endfor
        z := z + I_{k-1}^k q; { q is prolonged }
        for l := m_1 + 2, m_1 + m_2 + 1 do
          z := S(A_k, z, g);
        endfor
      endfor
  endif
end
```

Fig. 4.1: Pseudo-codice funzione MG

In questo capitolo faremo quindi riferimento sia alle **Nested Iteration** spiegate nel capitolo 2, sia alle griglie triangolari definite nel capitolo 3.
Innanzitutto $MG(1, z_0, g) = A_1^{-1}g$, ossia il livello più grossolano, *coarse*, si risolve con un sistema lineare.
Occorre poi fissare due valori $m_1, m_2 \in \mathbf{N}$ e un parametro $p \in \mathbf{N}$ che assume quasi sempre valori 1 o 2: se poniamo $p = 1$ otteniamo il *V-cycle*, per $p = 2$ otteniamo il *W-cycle*.
Definiamo poi l'operatore di rilassamento o smussamento

$$S(A, z, g) = z - \frac{1}{\Lambda_k} r(A, z, g)$$

dove $r(A, z, g) = g - Ax$ è il residuo del sistema lineare Ax=g, e Λ_k è un opportuno parametro di rilassamento; per ogni livello k va fissato un parametro di rilassamento tali che $\rho(A_k) \leq \Lambda_k \leq C/diam_k^2$, con C un'opportuna costante e $\rho(A_k)$ raggio spettrale di A_k.
Per k>1, $MG(k, z_0, g)$, detto iterazione di k-esimo livello, si articola in 3 passi:

1. Passo di **pre-smussamento**: per $1 \leq l \leq m_1$ si pone

$$z_1 = S(A_k, z_{l-1}, g).$$

2. Passo di **correzione** dell'errore: sia

$$\overline{g} = I_k^{k-1} r(A_k, z_{m1}, g), \quad q_0 = 0;$$

per $1 \leq i \leq p$ sia

$$q_i = MG(k-1, q_{i-1}, \overline{g});$$

allora poniamo

$$z_{m1+1} = z_{m1} + I_{k-1}^k q_p.$$

3. Passo di **post-smussamento**: per $m_1 + 2 \leq l \leq m_1 + m_2 + 1$ poniamo:

$$z_l = S(A_k, z_{l-1}, g)$$

Da notare che per $m_2 = 0$ non viene eseguito alcun passo di post-smussamento.

Il risultato di tali operazioni è:

$$MG(k, z_0, g) = z_{m_1+m_2+1}.$$

4.2 Convergenza e costo

Sia Ω un poligono convesso. Poniamo:

$$a(u, v) = \int_\Omega (\alpha \nabla u \cdot \nabla v + \beta uv) dx$$

dove α e β sono funzioni smooth (differenziabili infinite volte in un punto) tali che per qualche $\alpha_0, \alpha_1, \beta_1 \in \mathbf{R}^+$, $\alpha_0 \leq \alpha(x) \leq \alpha_1$, $0 \leq \beta(x) \leq \beta_1$, $\forall x \in \Omega$.
Sia dato inoltre il problema di Dirichlet di trovare $u \in V$ tali che $a(u, v) = (f, v) \ \ \forall v \in V$; supponiamo di risolvere tale problema con la funzione MG, che implementa il metodo multigrid, vista prima, con K>1 livelli ottenuti tramite raffinamenti iterativi della mesh a livello 1, che ha t_1 triangoli.
Ricordando che $||v||_E = \sqrt{a(v, v)}$, e le considerazioni fatte prima, valgono i seguenti teoremi:

- Se m=max(m1,m2) è grande abbastanza, l'iterazione di k-esimo livello di un W-cycle è una contrazione con costante contrattiva indipendente da k;
- Per qualsiasi m>0 l'iterazione di k-esimo livello di un V-cycle è una contrazione con costante contrattiva indipendente da k;
- Se l'iterazione di k-esimo livello è una contrazione con costante di contrattività Γ indipendente da k e se il numero di esecuzioni dell'iterazione di k-esimo livello, $\overline{s}$ è sufficientemente grande, allora esiste una costante C>0 tali che:

  $$||u_k - \hat{u_k}||_E \leq Ch_k|u|$$

  dove u_k è la proiezione della soluzione esatta sui nodi di T_k e $\hat{u_k}$ è l'approssimazione ottenuta con l'algoritmo multigrid MG. In particolare la stima dell'errore visto sopra è in norma dell'energia, che riguarda essenzialmente le derivate delle funzioni;
- Il costo computazionale dell'algoritmo multigrid completo a k livelli è $O(n_k)$, dove $n_k = 2t_1 4^k$. In particolare n_k è una maggiorazione del numero di nodi in T_k, essendo t_1 il numero di trangoli nella mesh inzizale.

4.3 Algoritmi multilivello per generare griglie coarse

Uno dei problemi che si ha tutt'oggi con i metodi multigrid, in particolare nell'applicazione di tali metodi a mesh non strutturate, è che non esiste alcun algoritmo globalmente accettato, standard, per generare griglie corse; infatti la maggior parte degli algoritmi usati usano una varietà di tecniche di agglomerazione per generare una serie di griglie coarse. L'idea generale dietro tali approcci è quella di partire da un generico nodo della griglia (o grafo duale nel caso stiamo operando su di esso), e di unire alcuni dei nodi adiacenti a quello nella griglia più coarse, così da avere meno nodi; la scelta dei nodi da fondere viene fatta o in base alla connettività della griglia/grafo, oppure vengono scelti quei nodi che rendono la griglia più coarse localmente ottimizzata. In genere la scelta viene fatta in maniera *greedy*, ovvero si fondono quei nodi per i quali si compie il minor sforzo possibile per fonderli, ma ciò genera griglie coarse di scarsa qualità.
In questa sezione ci occuperemo quindi di presentare un algoritmo che risolva il problema della generazione delle griglie coarse; in particolare questo problema viene affrontato come un problema di ottimizzazione dove l'obiettivo è generare una griglia coarse che ottimizza una particolare funzione obiettivo che coglie la qualità complessiva di una griglia soggetta a vincoli di upper bound e lower bound riguardanti la dimensione della griglia stessa.
Introduciamo due concetti fondamentali: il primo è il *volume di controllo*, definito come una regione arbitraria delimitata da una superficie chiusa e fissa nello spazio, che può contenere quindi del fluido, ed è contenuto in una griglia. Il secondo concetto è in realtà un parametro che misura la qualità generale di un volume di controllo bidimensionale, ovvero

$$A = \frac{l^2}{S},$$

dove l è la lunghezza del perimetro del volume, S è l'area; nel caso di volumi tridimensionali si calcola

$$A = \frac{S^3 l^2}{V},$$

dove S è l'area della superficie, mentre V è il volume. Un volume è ben modellato se è il più compatto possibile, ovvero se il parametro A è il più piccolo possibile; è bene che i volumi bidimensionali e tridimensionali siano quanto più simili ad una circonferenza e ad una sfera rispettivamente.
In una griglia possono essere contenuti più volumi di controllo, che quindi sono gli elementi di una griglia, e quindi un buon modo di misurare la qualità generale di una griglia è sommare i singoli parametri A di ciascun volume di controllo: detto *NCoarse* il numero

di tali volumi, definiamo la funzione F_1 come:

$$F_1 = \sum_{i=1}^{NCoarse} A_i.$$

Visto che il nostro obbiettivo è ottenere una griglia con volumi di controllo (elementi) ben modellati, sarebbe utile avere una griglia che minimizzi F_1; tale funzione non tiene conto però della dimensione di ciascun volume di controllo, allora introduciamo una funzione obiettivo F_2 che fa la somma pesata dei vari parametri A, in modo da penalizzare quei volumi di controllo con un A maggiore:

$$F_2 = \sum_{i=1}^{Ncoarse} w_i A_i,$$

dove w_i è il numero di nodi fusi insieme per ottenere l'i-esimo volume di controllo.
In ogni caso sia F_1 che F_2 non tengono conto del fatto che se una griglia coarse ha una buona qualità complessiva, potrebbe avere ancora molti volumi di controllo di scarsa qualità; allora si introduce la funzione

$$F_3 = maxA_i \qquad i = 1, ..., NCoarse$$

anch'essa da minimizzare. Queste tre diverse funzioni possono essere usate come obiettivo da ottimizzare quando si genera la griglia coarse di livello successivo; si può quindi formulare il problema di costruzione di una griglia coarse come il seguente problema di ottimizzazione:

Data una griglia iniziale, generare una griglia coarse in modo tale che ciascuno dei suoi volumi di controllo contenga almeno L_min e al più L_max elementi della griglia iniziale, e la griglia coarse minimizza una delle tre misure di qualità F_1, F_2 e F_3.

In ogni caso, le varie misure di qualità possono essere combinate per realizzare un problema multi-obiettivo, per esempio F_3 può essere facilmente combinata con F_1 o F_2: in questo caso l'algoritmo genera prima una griglia che minimizza F_3, e tra le diverse soluzioni che danno lo stesso valore di F_3, cerca la particolare soluzione che ha anche F_1 o F_2 minimo.
Vedremo ora un algoritmo multilivello che risolve tale problema di ottimizzazione. L'idea è la seguente: dato un problema iniziale M, l'algoritmo multilivello costruisce una sequenza di approssimazioni di M $(M_1, M_2, ..., M_n)$ tali che il problema M_{i+1} è più piccolo di M_i.

Una volta ottenute tale sequenza, la soluzione del problema originale si ottiene a partire dalla soluzione del sottoproblema più coarse, M_n. Questa soluzione viene poi utilizzata per calcolare la soluzione del seguente sottoproblema più fine, ovvero M_{n-1}, la quale poi viene ottimizzata e usata per trovare la soluzione di M_{n-2} e così via fino a trovare la soluzione del problema iniziale M.
Quindi distinguiamo tre fasi in un algoritmo multilivello: *coarsening*, *soluzione iniziale* e *uncoarsening*.
Notiamo quindi che l'idea alla base del paradigma multilivello è molto simile a quella di un algoritmo multigrid.
In ogni caso un problema approssimato deve far si che la qualità della soluzione calcolata a tale livello sia simile (se non uguale) alla qualità della stessa soluzione se vista nel problema originale: ciò mi assicura che l'ottimizzazione fatta ad un livello più coarse ottimizza la soluzione del problema originale.
L'algoritmo che vedremo modella la griglia usando un grafo $G(V, E)$, dove ad ogni nodo del grafo corrisponde un elemento della griglia, e tra due nodi c'è un arco solo se i corrispettivi elementi della griglia sono legati da un segmento , o una faccia nel caso di griglia tridimensionale.
Mostriamo in figura un esempio di griglia triangolare e del suo corrispettivo grafo.

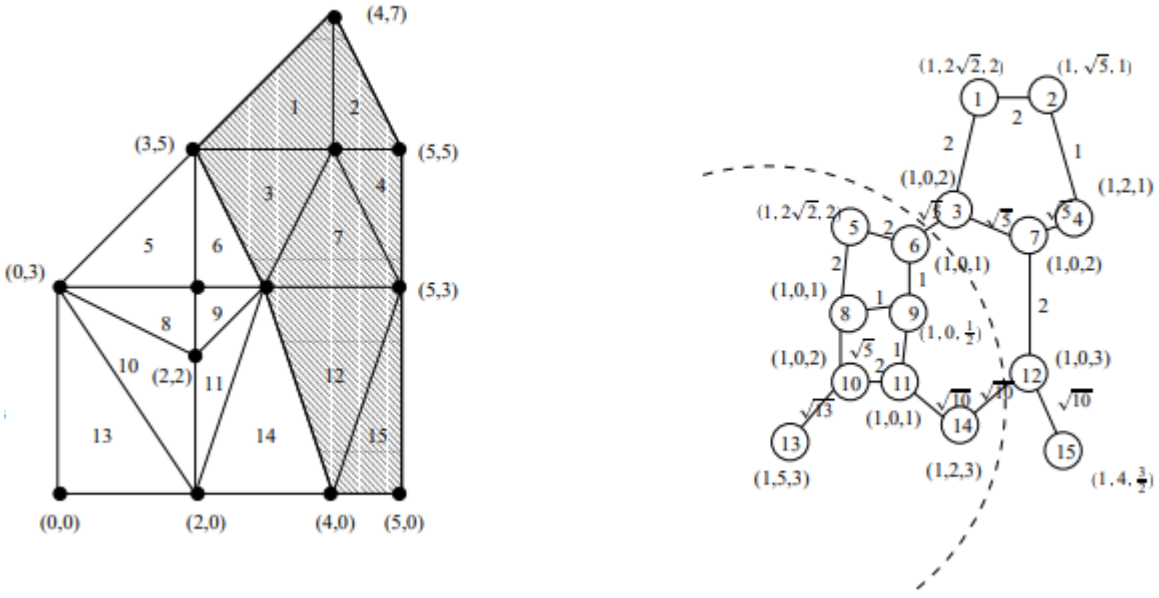


Fig. 4.2: Esempio di griglia triangolare e relativo grafo

Ad ogni nodo v del grafo sono associati tre valori:

- *peso* v^w, che indica il numero di elementi della griglia che quel nodo rappresenta. Inizialmente sono tutti settati ad 1, ma poi tale valore viene aggiornato quando passo ad una griglia più coarse.
- *Confine-superficie* v^s viene usato per indicare la lunghezza del segmento, o l'area della superficie, di quell'elemento della griglia che non è condiviso da nessun altro elemento, per esempio per gli elementi di confine della griglia. Da notare che tutti gli elementi interni hanno $v^s = 0$.
- *Volume* v^v indica l'area di un elemento in una grigia bidimensionale, o il volume nel caso di una griglia tridimensionale.

Ad ogni arco è poi associato un peso che indica la lunghezza del segmento, o della faccia, che unisce gli elementi a cui corrispondono i nodi estremi di quell'arco.
Il grafo pesato contiene quindi tutte le informazioni per calcolare i parametri di qualità di ciascun elemento della griglia: in particolare v^v rappresenta l'area di un elemento (nel caso di griglie bidimensionali), mentre il perimetro si ottiene sommando i pesi lungo gli archi più v^s, nel caso in cui tale elemento contiene più elementi/nodi (caso di griglia più coarse). Allo stesso modo si può ragionare per i volumi di controllo.
Data tale rappresentazione per la griglia, una griglia coarse può essere vista come un partizionamento in K sotto-insiemi del grafo originale, dove ogni partizione conterrà un numero di nodi tra L_{min} e L_{max}, e ottimizzerà una delle tre funzioni obiettivo viste prima. In pratica, i nodi in ogni partizione rappresenteranno gli elementi che verranno combinati per formare il volume di controllo della griglia più coarse.
Il nostro algoritmo opererà su tale grafo.
Vediamo ora le tre fasi dell'algoritmo.

4.3.1 Fase di Coarsening

Dato il grafo iniziale G che rappresenta la griglia inziale, si costruisce una sequenza di n grafi coarse $G_1, ..., G_n$. Ogni grafo G_i è ottenuto dal precedente grafo G_{i-1} trovando un insieme *massimale indipendente* I_{i-1} di archi e fondendo tutti i nodi che si trovano agli estremi di tali archi: quindi G_i conterrà un numero di nodi pari a quello di G_{i-1} meno $|I_{i-1}|$. Un insieme di archi è *indipendente* quando non ci sono due o più archi di quell'insieme che incidono sullo stesso nodo; un insieme indipendente è *massimale* quando non è possibile aggiungere altri archi senza violare l'indipendenza. Per mantenere le proprietà geometriche della griglia originale, i pesi degli archi e dei nodi del grafo G_1

vengono aggiornati per tener conto degli archi e nodi fusi. In particolare, se (v_1, v_2) è un arco contenuto in I_{i-1} e u_1 è un nodo di G_i ottenuto unendo i nodi (v_1 e v_2), allora l'algoritmo pone

$$u^w = v_1^w + v_2^w, \quad u^s = v_1^s + v_2^s, \quad u^v = v_1^v + v_2^v.$$

Mentre per preservare le informazioni di connettività, gli archi di u saranno l'unione degli archi di v_1 e v_2.
Siamo così in grado si calcolare il parametro A per ogni volume di controllo che ogni nodo rappresenta.
Il passo chiave sta nel trovare gli insiemi massimali indipendenti: par fare ciò utilizziamo un metodo chiamato **globular matching**. Lo scopo di tale metodo consiste nel trovare un insieme massimale indipendente in modo tale che i volumi di controllo creati fondendo i vertici agli estremi di tali archi abbiano parametro A il più basso possibile: per fare ciò si usa la seguente tecnica greedy.
I nodi sono visitati per ordine decrescente di grado; se un nodo non è ancora stato abbinato a nessun altro (per esempio perché non è l'estremo di nessun arco che si trova nell'insieme indipendente), lo abbiniamo ad un suo nodo adiacente che non è ancora stato abbinato e che mi da il più piccolo A, e il nodo risultante non viola il vincolo di dimensione massima di un volume di controllo.
L'arco relativo a questi due nodi viene aggiunto poi all'insieme indipendente. Tale metodo ha complessità $O(|E|)$.
La fase di coarsening termina quando non posso rendere il grafo più grafo di com'è senza violare il vincolo di massima dimensione di un volume di controllo L_{max}.

4.3.2 Fase di soluzione iniziale

Lo scopo di questa fase è di calcolare una soluzione del problema originale usando l'approssimazione più coarse, ovvero il grafo G_n i cui nodi sono stati ottenuti fondendo gli elementi adiacenti della griglia originale; è inoltre garantito che ogni nodo di G_n non ha più di L_{max} elementi della griglia originale. Considerando ciò, otteniamo la soluzione del problema originale semplicemente considerando i nodi di G_n come elementi coarse della griglia originale. Questa soluzione potrebbe però non rispettare il vincolo di L_{min}, visto che ogni nodo di G_n potrebbe contenere meno di L_{min} elementi della griglia originale: ciò andrà coretto nella fase successiva.

4.3.3 Fase di Uncoarsening

Lo scopo di tale fase è di prendere la soluzione trovata nel grafo più coarse e di propagarla indietro fino al grafo originale, passando per i grafi G_{n-1},..., G_1, G e di raffinarla durante tale propagazione.
La soluzione iniziale al grafo più coarse altro non è che un partizionamento in $|V_n|$ sottoinsiemi di tale grafo, dove l'i-esimo sottoinsieme del grafo contiene solo l'i-esimo nodo. Quando poi la soluzione verrà propagata al grafo più fine successivo, molti di questi $|V_n|$ conterranno più di un singolo nodo. La qualità generale di tale soluzione potrebbe essere migliorata spostando un nodo da una partizione ad un'altra, senza ovviamente violare i vincoli: tale movimento di nodi sposta un sottoinsieme di elementi da un volume di controllo ad un altro. Queste migliorie sono possibili per tre motivi:

1. gli insiemi massimali indipendenti sono calcolati da algoritmi greedy e sono ottimali;
2. la funzione obiettivo che proviamo ad ottimizzare potrebbe essere diversa dall'euristica usata per trovare gli insiemi indipendenti massimali;
3. i grafi più fini hanno un grado maggiore di libertà che può essere usato per migliorare la soluzione che arriva da un grafo più coarse.

Utilizziamo un algoritmo di raffinamento random che è simile a quello usato per partizionare. Esso ha un certo numero di iterazioni: ad ogni iterazione i nodi vengono visitati in maniera random, e per ogni nodo v calcoliamo la riduzione del valore di funzione obiettivo che si avrebbe spostando quel nodo da un sottoinsieme ad una altro. Se esiste uno spostamento che ci riduce la funzione obiettivo senza violare alcun vincolo, allora tale spostamento di v viene effettuato, altrimenti lo si lascia lì dov'è. L'algoritmo termina quando non possono essere spostati altri nodi; in genere tale algoritmo converge in due-cinque iterazioni.
Il lato negativo di questo algoritmo è che le partizioni che vengono scoperte potrebbero non essere contigue; per risolvere ciò, alla fine del raffinamento, si controllano quali partizioni hanno volumi di controllo non contigui, e per ciascuno di essi crea una diversa partizione. La soluzione finale potrebbe contenere quindi più di $|V_n|$ partizioni. Inoltre le partizioni potrebbero avere meno di L_{min} elementi: per correggere ciò, dopo che le partizioni sono state rese contigue, si cerca di fondere qualche partizione senza violare però il vincolo di L_{max}. Alla fine di tale operazione, i volumi di controllo che avranno meno di L_{min} elementi saranno molto pochi, e saranno quelli adiacenti ai volumi di controllo più grandi, quelli per i quali il vincolo di L_{max} è "quasi" violato. Allora si spostano dei vertici da questi volumi di controllo più grandi a quelli più piccoli, così alla fine i vincoli di L_{min} e L_{max} sono rispettati per tutti i volumi.