Corso di Laurea Specialistica in Ingegneria Meccanica e in Ingegneria dell'Energia

Progetti numerici al calcolatore

FORTRAN Frequently (e altro ancora) Asked Questions

14 novembre 2008

Indice

Introduzione

Nelle pagine che seguono si cerca di rispondere alle domande abbastanza frequenti negli studenti che si stanno preparando all'esame di Metodi Numerici per l'Ingegneria, del corso di laurea in Ingegneria Meccanica e in Ingegneria dell'Energia.
Sono pagine che sono aggiornate, a partire dalle domande e dai problemi riscontrati durante la preparazione dei progetti numerici al calcolatore, e tengono conto delle domande e dei dubbi posti dagli studenti nel corso degli anni.

Annamaria Mazzia

Capitolo 1

Problemi di carattere teorico

Problemi che si incontrano quando si deve implementare un algoritmo al calcolatore... o nello studiare la teoria...

Sommario

1.1 Sul Progetto 1

1.1.1 Controllo della convergenza nella Regula Falsi?

Sappiamo che bisogna operare due controlli sull'algoritmo della Regula Falsi.

Il primo è sul numero massimo di iterazioni possibili: se si effettuano troppe iterazioni (`itmax`) e si osserva che gli scarti sulle variabili x e y non diminuiscono nè descrescono ma rimangono stazionari o aumentano, allora anche con un numero infinito di iterazioni non si arriverebbe a convergenza e quindi è bene arrestare l'algoritmo.

Il secondo controllo è sugli scarti $|x_{n+1} - x_n|$ e $|y_{n+1} - y_n|$: perchè ci sia convergenza, entrambi gli scarti devono essere minori di una tolleranza prefissata (`toll`).

Se solo uno dei due scarti è minore della tolleranza allora non si è ancora raggiunta la convergenza e bisogna proseguire nelle iterazioni del metodo.

In FORTRAN, un ciclo `do while` che traduce tutte queste richieste è:

```
do while  (it.le.itmax).and.((scartox.ge.tol).or.(scartoy.ge.tol))
   applicazione Regula Falsi
end do
```

La proposizione, che fa andare avanti il ciclo `do while` fintantochè rimane vera, si può leggere nel modo seguente: è vero che (`it` $\leq$ `itmax`) ed è vero che (`scartox` $\geq$ `tol` oppure che `scartoy` $\geq$ `tol`), cioè almeno uno dei due scarti su x o su y è maggiore o uguale a `tol`.

Quando l'intera proposizione precedente diventa falsa noi usciamo dal ciclo `do while`. Scrivere che non è vera la proposizione precedente dà come risultato la seguente proposizione[1]:

`(it.gt.itmax).or.((scartox.lt.tol).and.(scartoy.lt.tol))`

vale a dire che

o (`it` $>$ `itmax`) oppure è vero l'asserto (`scartox`$<$`tol` e `scartoy`$<$`tol`), quindi entrambi gli scarti sono minori della tolleranza prefissata.

È perciò sbagliato porre come condizione del ciclo `do while` la seguente proposizione

```
do while  (it.le.itmax).and.((scartox.ge.tol).and.(scartoy.ge.tol))
```

in quanto si uscirà dal ciclo quando solo uno dei due scarti sarà minore della tolleranza prefissata.

1.1.2 Sulle due funzioni del sistema

Il sistema da risolvere è

$$\begin{cases} f_1(x,y) = 0 \\ f_2(x,y) = 0 \end{cases}$$

Da un punto di vista strettamente matematico, sono fissati i valori degli angoli δ, ψ, γ e

$$f_1(x,y) = \tan\delta - \tan x / \sqrt{1 + \tan^2\Gamma(x,y)\tan^2 x}$$

$$f_2(x,y) = \tan\psi - \tan\beta(x,y)\frac{\left[\cos\Gamma(x,y) + (\cos\gamma/\sin y)\tan\Gamma(x,y)\tan^2 x\right]}{\sqrt{\cos^2\Gamma(x,y) + \tan^2 x}}$$

[1] Ricordiamo che in logica la negazione di .and. ha come risultato .or. mentre la negazione di .or. ha come risultato .and.

Dire che è falsa la proposizione *Piove e il cane dorme* vuol dire che non è vera una delle due parti che compongono la proposizione stessa, cioè *O non piove oppure il cane non dorme.*

Dire che è falsa la proposizione *Ho perso il cellulare o me lo hanno rubato* vuole dire che *Non ho perso il cellulare e non me ho hanno rubato*

dove

$\Gamma(x, y) = \text{asin}(\sin y \cos\gamma - \sin\gamma\sqrt{\cos^2 y + \tan^2 x})$

e

$\beta(x, y) = \text{atan}(\dfrac{\sin y \cos\Gamma}{(\cos\gamma - \sin y \sin\Gamma)})$

dipendono non solo da x e y ma anche da γ e γ e Γ rispettivamente.

Quando si va ad implementare un codice per risolvere il sistema delle due equazioni tramite la Regula Falsi, bisogna definire le due funzioni f_1 e f_2. Occorre allora tenere presente che i parametri da cui dipende f_1 non sono solamente le incognite x e y ma anche gli angoli δ e γ mentre f_2 dipende non solo da x e y ma anche da ψ e γ. Gli angoli δ e ψ possono variare entro un certo intervallo di ammissibilità e quindi possono assumere valori diversi a seconda del problema che si vuole risolvere. L'angolo γ è fissato una volta per tutte come $\gamma = 7^o20'03''$ ma rimane pur sempre un parametro da cui dipendono entrambe le funzioni.

Lavorando in FORTRAN, la `function` che definisce f_1 avrà come valori da cui dipende, esattamente x, y, γ e δ e la prima istruzione che la definisce sarà, ad esempio:

```
real*8  function fun1(x,y,gamma,delta)
```

mentre f_2 sarà introdotta da

```
real*8  function fun2(x,y,gamma,psi)
```

Nel programma principale dove le due funzioni vengono chiamate, bisogna tenere presente il significato delle variabili e l'ordine in cui sono date e continuare quindi a chiamare `fun1(x,y,gamma,delta)` e `fun2(x,y,gamma,psi` (e non `fun1(x,y)`).

Non conviene passare gli angoli δ e ψ come parametri interni alle `functions` perchè, se si cambia tipo di problema, cambiano anche quei valori e, di conseguenza, bisogna andare a cambiare i parametri.

Per il problema che si deve risolvere solo γ è un dato che non cambia mai e quindi, volendo, lo si potrebbe definire come un parametro all'interno delle due `functions`. In realtà, se lo si vuole definire come parametro, conviene definirlo tale nel programma principale e, quindi, continuare a passarlo tra i dati di input delle due functions così come abbiamo scritto prima.

Per quanto riguarda Γ e β si possono costruire altre due `functions` che vengono richiamate all'interno di `fun1` e `fun2` oppure le si può introdurre come variabili ausiliarie all'interno delle `functions` `fun1` e `fun2` stesse - riscrivendo la stessa istruzione per Γ sia all'interno di `fun1` che di `fun2`.

Per $\tan\beta$ non conviene definire prima β come l'arcotangente di una certa quantità e poi fare la tangente di β ma conviene calcolare direttamente $\tan\beta$.

Inoltre, per evitare divisioni per zero nella f_2 dovute a $\dfrac{1}{\sin y}$ (si veda il secondo addendo da moltiplicare per $\tan\beta$) è bene scrivere $\tan\beta$ come $\sin y\mathcal{A}$ dove

$\mathcal{A} = \cos\Gamma/(\cos\gamma - \sin y \sin\Gamma)$ e scrivere la f_2 come

$$f_2(x,y) = \tan\psi - \mathcal{A}\frac{\left[\sin y\cos\Gamma(x,y) + \cos\gamma\tan\Gamma(x,y)\tan^2 x\right]}{\sqrt{\cos^2\Gamma(x,y) + \tan^2 x}}$$

La `function` `fun2` deve dunque essere scritta tenendo conto di queste semplificazioni di forma.

Facciamo un esempio di come scrivere una `function` che dipende da due incognite x e y ma anche da altri parametri che chiamiamo δ e γ.

Sia dunque

$$f(x,y) = \cos\delta\sin x + \tan\Gamma^2\sqrt{\tan\alpha^2 + \sin y^2}$$

con $\sin\Gamma = \cos x\sin\gamma$ e $\tan\alpha = \sin y\cos\Gamma$.

La `function` avrà perciò come parametri di input non solo x e y ma anche γ e δ, mentre Γ e α, poichè sono utilizzate solo per definire la funzione stessa saranno definite solo all'interno della `function`.

```
real*8  function funprova(x,y,gamma,delta)
implicit none
real*8 x,y, gamma, delta
real*8 Gaux, talpha
Gaux= asin(cos(x)*sin(gamma))
talpha= sin(y)*cos(Gaux)
funprova= cos(delta)*sin(x)+tan(Gaux)**2*sqrt(talpha**2+sin(y)**2)
return
end
```

Osserviamo che la variabile Γ ha preso il nome di `Gaux` nella `function` (in FORTRAN non c'è differenza tra caratteri maiuscoli e minuscoli) e $\tan\alpha$ è la variabile `talpha`.

Se volessimo invece introdurre altre due `functions` per Γ e $\tan\alpha$ siamo liberi di farlo ma per l'esempio dato non semplifichiamo il codice e la sua leggibilità ma la complichiamo un po'. Infatti dovremmo scrivere qualcosa come:

```
real*8  function funprova(x,y,gamma,delta)
implicit none
real*8 x,y, gamma, delta
real*8 Gaux, talpha, fungamma, funalpha
Gaux= fungamma(x,gamma)
talpha= funalpha(y, Gaux)
funprova= cos(delta)*sin(x) + tan(Gaux)**2*sqrt(talpha**2+sin(y)**2)
return
end
```

```
real*8  function fungamma(x,gamma)
implicit none
real*8 x, gamma
fungamma= asin(cos(x)*sin(gamma))
return
end

real*8 function funalpha(x,G)
implicit none
real*8  x,G
funalpha = sin(x)*cos(G)
return
end
```

Osserviamo come le due `functions` che abbiamo introdotto e chiamato `fungamma` e `funalpha` sono state dichiarate anche in `funprova`.

Inoltre, quando abbiamo scritto la `funalpha` abbiamo usato come variabili `x` e `G`: l'importante non è il nome dato alle variabili di input ma il loro significato e la corrispondenza con i parametri dati la dove la `function` viene chiamata. Se, all'interno della `funprova` scrivessi `talpha= funalpha(Gaux, y)` allora farei un errore perchè nel calcolare la `funalpha` verrebbe dato a `x` il valore di `Gaux` e a `G` il valore di `y` mentre deve accadere il contrario.

1.1.3 La trasformazione radianti-gradi va fatta ad ogni iterazione?

Lo schema della Regula Falsi viene implementato usando tutte le variabili in radianti. Perciò il controllo sugli scarti viene fatto nel sistema di misura in radianti. Non ha senso, perciò, trasformare ogni volta le approssimazioni x_k e y_k da radianti a gradi.

La trasformazione radianti-gradi va fatta solo una volta che il metodo è arrivato a convergenza per stampare i risultati nel sistema di misura in radianti.

1.2 Sul Progetto 2

1.2.1 Sul precondizionatore diagonale

Il precondizionatore diagonale è $K^{-1} = D^{-1}$ dove la matrice D è la matrice diagonale costituita dagli elementi diagonali della matrice A, che abbiamo memorizzata in `SYSMAT`.

Quindi sappiamo già quali sono gli elementi della matrice D e, quindi, della sua inversa. Non c'è bisogno di andare a memorizzarli in un nuovo vettore.

La matrice D^{-1} ci serve per fare dei prodotti matrice-vettore del tipo $D^{-1}\mathrm{v} = \mathrm{w}$. Conviene implementare l'algoritmo che esegue questo prodotto utilizzando una

`subroutine` (perchè questo prodotto è da ripetersi più volte all'interno delle iterazioni delle CR e del CGM).

Teniamo presente che gli elementi della diagonale principale della matrice A per $i = 1, \ldots, n$ sono memorizzati in `SYSMAT(IA(i))` per $i = 1, \ldots, n$, quindi sono questi i valori diagonali della matrice D. Per l'inversa di D, basta considerare i valori reciproci: $1/$SYSMAT(IA(i)).

Le componenti del vettore w sono dunque date da:

$$w_i = v_i/\text{SYSMAT(IA(i))} \quad i = 1, \ldots, n$$

I dati di input per ricavare w sono dunque i vettori `SYSMAT, IA` e il vettore v con le corrispondenti dimensioni.

1.2.2 Sul vettore termine noto b

Se noi, a priori, sappiamo quale deve essere la soluzione del sistema $A\mathbf{x} = \mathbf{b}$, allora possiamo costruire il vettore b facendo il prodotto di A per il vettore soluzione x.

Negli esempi proposti (per l'anno accademico 2006/2007) due matrici hanno alcuni elementi della diagonale principale molto grandi $\gg 10^{25}$. Per queste matrici dobbiamo costruire un vettore soluzione x che ha componente nulla in corrispondenza di quelle righe il cui elemento diagonale è molto grande, e ha componente unitaria altrove.

Per le altre matrici (dell'anno accademico 2006/2007 e per tutte quelle dell'a.a. 2007/2008 e dell'a.a.2008/2009), invece, il vettore soluzione deve avere tutte le componenti uguali ad 1.

a.a. 2006/2007

Tenendo presente che in queste ultime matrici non ci sono elementi diagonali $\gg 10^{25}$, possiamo costruire il vettore soluzione in questa maniera (è un esempio di soluzione del problema): Definiamo un vettore ausiliario `xaux` in cui memorizziamo quella che deve essere la soluzione del sistema, quale che sia la matrice test delle quattro proposte:

```
001     Per i = 1, n
002         Se a_ii ≥ 10^25
003             xaux(i) = 0
004         altrimenti
003             xaux(i) = 1
004         Fine Se
004     Fine Per
```

In questo modo, per le prime due matrici avremo il vettore con 0 e 1 mentre per le ultime due matrici avremo un vettore con tutti 1 (perchè non ci sono elementi diagonali $\gg 10^{25}$).

———-

a.a. 2007/2008 e a.a. 2008/2009

Il vettore xaux viene posto semplicemente uguale ad un vettore di tutti 1.

———-

Questo vettore xaux può essere quindi utilizzato per ricavare il vettore termine noto, effettuando il prodotto matrice-vettore.

Il vettore che approssimerà la soluzione del sistema ottenuto mediante l'algoritmo CGM dovrà essere vicino a xaux a meno della tolleranza prefissata.

1.2.3 Su $\mathbf{x}_0 = K^{-1}\mathbf{b}$

Lo schema delle Correzioni Residue (CR) è definito tramite le equazioni:

$$\begin{aligned} \mathbf{x}_0 &= K^{-1}\mathbf{b} \\ \mathbf{x}_1 &= \mathbf{x}_0 + K^{-1}\mathbf{r}_0 \\ &\vdots \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + K^{-1}\mathbf{r}_k \end{aligned}$$

Effettuare un'iterazione preliminare con lo schema CR prima di implementare lo schema CGM vuol dire calcolare, tramite le equazioni precedenti, $\mathbf{x}_0$ e $\mathbf{x}_1$. L'approssimazione $\mathbf{x}_1$ viene dunque utilizzata come approssimazione iniziale per il CGM.

Analogamente, effettuare 5 iterazioni preliminari con lo schema CR, vuol dire calcolare $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, $\ldots, \mathbf{x}_5$. L'approssimazione $\mathbf{x}_5$ viene poi utilizzata come approssimazione iniziale per il CGM.

L'iterazione $\mathbf{x}_0 = K^{-1}\mathbf{b}$ è il punto di partenza per lo schema CR ma non è un'approssimazione ottenuta dall'applicazione dello schema CR.

Se si implementa il CGM partendo da $\mathbf{x}_0 = K^{-1}\mathbf{b}$ allora l'approssimazione iniziale del CGM è presa considerando la stessa approssimazione iniziale dello schema CR ma senza implementare lo schema CR stesso.

1.3 Prodotti matrice-vettore nel GCM

Si è detto che nel GCM sono solo due i prodotti matrice-vettore da farsi e, precisamente, $A\boldsymbol{p}_k$ e $K^{-1}\boldsymbol{r}_{k+1}$.

Nel calcolare β_k si ha:

$\beta_k = -\frac{\boldsymbol{r}_{k+1}^T K^{-1} A\boldsymbol{p}_k}{\boldsymbol{p}_k^T A\boldsymbol{p}_k}$

Ma allora c'è da calcolare anche $K^{-1}A\boldsymbol{p}_k$?

No, perchè $\boldsymbol{r}_{k+1}^T K^{-1} A\boldsymbol{p}_k = (K^{-1}\boldsymbol{r}_{k+1})^T A\boldsymbol{p}_k$ essendo K^{-1} simmetrica, per la proprietà che il trasposto di un prodotto matrice-vettore è uguale al vettore trasposto per la matrice stessa (se la matrice è simmetrica) e per le proprietà del prodotto scalare.

Quindi un'implementazione corretta del GCM prevede solo questi due prodotti matrice-vettore (fare un terzo prodotto $K^{-1}A\boldsymbol{p}_k$ è sovrabbondante).

1.4 Prima e Seconda identità di Green; teorema di Green.

La prima identità di Green si deduce dal teorema della divergenza e dalle relazioni che legano una funzione u e il gradiente di una funzione w.

In particolare, il teorema della divergenza stabilisce:

$$\int_\Omega div(\mathbf{A})\ \mathrm{d}\Omega = \int_\Gamma \mathbf{A}\cdot\mathbf{n}\ \mathrm{d}s$$

dove $\mathbf{A}$ è una funzione vettoriale, per esempio, in due dimensioni, $\mathbf{A} = (A_1, A_2)$, $\mathbf{n} = (n_x, n_y)$ è la normale uscente, Ω è il dominio con contorno Γ. Ricordiamo che $div(\mathbf{A}) = \nabla\cdot\mathbf{A} = \frac{\partial A_1}{\partial x} + \frac{\partial A_2}{\partial y}$. Quindi, $\nabla\cdot$ indica l'operatore di divergenza, ∇ il gradiente e ∇^2 l'operatore laplaciano.

Si ha, inoltre,

$$\nabla\cdot(u\nabla w) = u\nabla^2 w + (\nabla u)\cdot(\nabla w)$$

$$\nabla\cdot(w\nabla u) = w\nabla^2 u + (\nabla w)\cdot(\nabla u)$$

La **prima identità di Green** è la seguente:

$$\int_\Omega \nabla u\cdot\nabla w\ \mathrm{d}\Omega = \int_\Gamma u\frac{\partial w}{\partial n}\mathrm{d}\Gamma - \int_\Omega u\nabla^2 w\ \mathrm{d}\Omega$$

o, equivalentemente,

$$\int_\Omega \frac{\partial u}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial w}{\partial y}\ \mathrm{d}\Omega = \int_\Gamma u\frac{\partial w}{\partial n}\mathrm{d}\Gamma - \int_\Omega u((\frac{\partial w}{\partial x})^2 + (\frac{\partial w}{\partial y})^2\ \mathrm{d}\Omega$$

La **seconda identità di Green** è:

$$\int_\Omega u\nabla^2 w\ \mathrm{d}\Omega = \int_\Omega w\nabla^2 u\ \mathrm{d}\Omega + \int_\Gamma (u\frac{\partial w}{\partial n} - w\frac{\partial u}{\partial n})\mathrm{d}\Gamma$$

In due dimensioni, il **teorema di Green** stabilisce:

$$\int_\Gamma u\ \mathrm{d}x + w\ \mathrm{d}y = \int_\Omega (\frac{\partial w}{\partial x} - \frac{\partial u}{\partial y})\ \mathrm{d}x\ \mathrm{d}y$$

In forma compatta si può scrivere:

$$\int_\Gamma \mathbf{F}\cdot\ \mathrm{ds} = \int_\Omega (\nabla\times\mathbf{F})\cdot\ \mathrm{da}$$

dove $\mathbf{F} = (u, w)$ e $\times$ rappresenta il prodotto vettoriale.

Capitolo 2

Problemi di carattere tecnico operativo

Problemi che si incontrano quando si programma, si compila e si esegue un programma FORTRAN.

Sommario

2.1 Errore di compilazione: `Invalid declaration...`

In FORTRAN conviene sempre dichiarare tutte le variabili, siano esse di tipo intero, reale, logiche, etc... L'istruzione `implicit none` va messa subito dopo la riga di codice che definisce la struttura del programma (il programma principale, una subroutine, una function): con questa istruzione, tutte le variabili devono essere dichiarate, cioè nessuna variabile può essere dichiarata implicitamente. Se una va-

riabile non viene dichiarata, durante la compilazione si ha un messaggio di errore relativo alla variabile non dichiarata.

Esempio: supponiamo che nel programma la variabile `a` venga utilizzata per memorizzare la radice quadrata di una certa espressione, `a= sqrt( (x1-x2)**2 +(y1-y2)**2)`, ma `a` non è stata dichiarata. Durante la compilazione si avrà il seguente messaggio (o uno analogo a seconda del compilatore utilizzato):

```
 a= dsqrt( (x1-x2)**2 +(y1-y2)**2)
           ^
Invalid declaration of or reference to symbol 'a' at (^) [initially
seen at (^)]
```

L'errore viene eliminato se si dichiara correttamente come variabile di tipo reale la variabile `a`.

2.2 Errore di compilazione: `Missing operand for operator...`

Messaggi del tipo `Missing operand for operator at (1) at end of expression at (2)` sono dovuti al fatto che si è scritto oltre la 72^a colonna.

In FORTRAN77 tutte le istruzioni vanno scritte tra la 7^a e la 72^a colonna.

Se un'istruzione è molto lunga e richiede di essere scritta oltre la 72^a colonna allora si deve andare a capo, mettendo un carattere alfanumerico (per esempio `a` o `b` o `1`, etc) sulla 6^a colonna della riga successiva a quella su cui si sta scrivendo e dove si continuerà a scrivere l'istruzione troppo lunga.

Esempio:

```
a=sin(2.d0*x)-cos(2.d0*x)+0.5d0*exp(x-3.d0)+sin(3.0d0*y)-cos(5.0d0*y)
```

non può essere scritto su un'unica riga perchè si va oltre la 72^a colonna. Si va perciò a capo

```
 a=sin(2.d0*x)-cos(2.d0*x)+0.5d0*exp(x-3.d0)
1      +sin(3.0d0*y)-cos(5.0d0*y)
```

Abbiamo messo `1` sulla 6^a colonna.

2.3 L'istruzione `OPEN`

È un'istruzione che serve per aprire files di lettura e/o scrittura di dati.

Nella stessa directory (cartella) in cui si eseguirà il programma FORTRAN, devono essere collocati i files di lettura che saranno letti durante l'esecuzione del programma. Nella stessa directory, saranno scritti i files di scrittura che verranno creati durante l'esecuzione del programma stesso.

Supponiamo quindi di avere un file chiamato `dati.dat` che dobbiamo fare leggere al programma perchè in esso sono contenuti i dati di input del programma. Il file `dati.dat` deve essere un file di testo.

Se si lavora con un sistema operativo WINDOWS, il file deve essere creato e modificato usando il BloccoNote.

Nel programma FORTRAN, l'istruzione che permette di aprire quel file per poter successivamente leggere i dati in esso contenuti, è:

```
open(unit1, FILE='dati.dat', STATUS='status')
```

Questa istruzione va scritta dopo aver dichiarato tutte le variabili del programma stesso.

Con questa istruzione, diciamo di aprire il file `dati.dat` associando ad essa la *label* `unit1`: `unit1` è un numero intero che può variare nell'intervallo $[1, 999]$ che sarà associato al file `dati.dat` all'interno del programma. Invece `status` indica lo stato del file, se esiste già (`old`), se non esiste ancora (`new`) o se non si conosce il suo stato (`unknown`).

Per esempio:

```
open(11, FILE='dati.dat', STATUS='old')
```

indica che il file `dati.dat` esiste già e ad esso associamo la *label* 11.

```
open(11, FILE='dati.dat', STATUS='new')
```

indica che il file `dati.dat` non esiste ma sarà creato durante l'esecuzione. In questo caso, `dati.dat` sarà un file di dati di output.

```
open(11, FILE='dati.dat', STATUS='unknown')
```

In tal caso non sappiamo se il file `dati.dat` esista già o meno. Conviene usare questo tipo di STATUS quando vogliamo sovrascrivere i dati di output ogni volta che facciamo girare il codice.

In uno stesso programma possiamo avere uno o più file da aprire. A ciascun file dobbiamo associare una *label* diversa.

L'istruzione di `OPEN` può essere semplificata omettendo lo `STATUS`

```
open(unit1, FILE='dati.dat')
```

2.4 Errore durante la lettura dei files di input nel CGM

Nel compilare il codice si ha:

```
list in: end of file
apparent state: unit 2 named  dati.dat
last format: list io
lately reading direct formated external IO
```

Cosa fare?

Le matrici test per il CGM sono date mediante i tre vettori `SYSMAT`, `JA` e `IA`. Nel file di testo (che chiamiamo per semplicità `dati.dat`) sono messi nell'ordine le dimensioni `N` e `NT` (su una stessa riga) e poi i tre vettori (diamo un esempio molto semplice in cui la matrice non è definita positiva ma solo simmetrica):

```
N, NTERM
8, 12
VETTORE SYSMAT
1. 2. 3. 4.
5. 6. 7. 8.
9. 10. 11. 12.
VETTORE JA
1 3 2 4 3 6 4 5 6 6 7 8
VETTORE IA
1 3  5  7 8  10 11 12 13
```

Le scritte di caratteri `N, NT, VETTORE SYSMAT, VETTORE JA, VETTORE IA` sono messe solo per far capire quali variabili sono date e in quale ordine e, perciò, devono essere cancellate dal file prima di darlo in input al programma FORTRAN.

Nel programma FORTRAN possiamo aver posto le seguenti istruzioni di lettura:

```
read(11,*) n, nt
read(11,*) (sysmat(i), i=1,nt)
read(11,*) (ja(i), i=1,nt)
read(11,*) (ia(i), i=1,n+1)
```

Associamo, quindi, al file `dati.dat` la *label* 11 e andiamo a leggere la prima riga in cui sono dati i valori di `n` e di `nt`; successivamente leggiamo il vettore `sysmat`, quindi i vettori `ja` e `ia`. L'istruzione `read(11,*) (sysmat(i), i=1,nt)` è un'istruzione che legge in forma compatta gli `nt` valori numerici che sono messi a partire da quella riga in poi (nel nostro caso dalla seconda riga in poi) e li memorizza nel vettore `sysmat`. Alla stessa maniera, una volta terminata la lettura degli `nt` valori di `sysmat` si passa alla riga successiva del file e si leggono i successivi `nt` valori che vengono memorizzati in `ja`. Analogo discorso vale per `ia`.

Nelle istruzioni di lettura non viene letta alcuna stringa di caratteri (quale `VETTORE SYSMAT` o `VETTORE JA`), ragion per cui dobbiamo cancellarle dal file di dati di input. Il file `dati.dat` deve dunque essere:

```
8, 12
1. 2. 3. 4.
5. 6. 7. 8.
9. 10. 11. 12.
1 3 2 4 3 6 4 5 6 6 7 8
1 3  5  7 8  10 11 12 13
```

In alcuni casi, tuttavia, anche se compaiono delle stringhe di caratteri, il codice non dà problemi di lettura così come avviene per questo file. Il motivo è semplice: se il file `dati.dat` fosse:

```
8  12        N, NTERM
1. 2. 3. 4.
```

```
5. 6. 7. 8.
9. 10. 11. 12.  VETTORE SYSMAT
1 3 2 4 3 6 4 5 6 6 7 8 VETTORE JA
1 3  5  7 8  10 11 12 13 VETTORE IA
```

allora le stesse istruzioni di prima andrebbero bene ugualmente perchè le stringhe di caratteri sono poste alla fine dell'ultimo valore numerico che viene letto per la riga presa in esame o dopo gli `nt` o `n` valori numerici letti in forma compatta. Quando si passa da un'istruzione `read` alla successiva, la stringa di caratteri non viene considerata dal FORTRAN perchè si passa a leggere il dato (o i dati) posti a partire dalla riga successiva a quella appena letta.

Nel nostro esempio, la prima `read` legge i valori di `n` e `nt` cioè 8 e 12, con la seconda `read` si va alla riga successiva (perciò vengono saltate le stringhe `N` e `NT`) e si leggono sequenzialmente, riga dopo riga, gli `nt`=12 valori da memorizzare in `sysmat`. Un volta letto l'ultimo valore, si passa alla `read` che legge il vettore `JA` andando nella riga successiva e quindi saltando la scritta `VETTORE SYSMAT`. Perciò non ci sono problemi di lettura.

Scrivere dei commenti sul significato dei numeri messi in un file di dati di input è utile quando ci sono pochi valori scritti. Nel nostro caso, scrivere `VETTORE SYSMAT` dopo 289 o 376 numeri non è molto utile perchè poco leggibile, perciò conviene avere un file di dati in cui non compaia nesssuna stringa di caratteri.

Ricordiamo inoltre, ancora una volta, che i file di dati di input devono essere aperti, scritti e modificati, in ambiente WINDOWS, sempre con il BloccoNote: solo in questo modo le variabili sono salvate nel formato di lettura (ASCII) che vuole il FORTRAN. Diversamente si avranno altri problemi nella lettura dei files di input, dovuti ad un formato che il FORTRAN non riesce ad interpretare.

2.5 L'istruzione CLOSE

L'istruzione `CLOSE` serve a chiudere i files aperti tramite l'istruzione `OPEN`.

Se il file è associato alla *label* `unit1`, allora il comando da scrivere è

`CLOSEunit1`

Conviene scrivere le istruzioni di `CLOSE` prima della `end` che chiude il programma.

Se ci si dimentica l'istruzione `CLOSE`, il programma viene compilato ugualmente e non dà errori nei risultati.

2.6 Come dare un significativo numero di cifre decimali a π?

Per rappresentare la variabile $\pi = 3.141592654...$ conviene ricordare che π si può esprimere utilizzando le seguenti formule:

$$\pi = 2\text{asin}(1)$$

$$\pi = 4\text{atan}(1)$$

In FORTRAN, quindi, la variabile `pi` dichiarata come `real*8` può essere definita come

`pi= 2.0d0 * asin(1.0d0)` o `pi= 2.0d0 * dasin(1.0d0)` (`dasin` è la funzione arcoseno in doppia precisione)

o, alternativamente, come

`pi= 4.0d0 * atan(1.0d0)` o `pi= 4.0d0 * datan(1.0d0)` (`datan` è la funzione arcotangente in doppia precisione)

2.7 Sull'inizializzazione dei vettori in FORTRAN77

Quando si dichiarano dei vettori in un programma FORTRAN occorre dichiarare anche la loro dimensione.

Uno stesso vettore, tuttavia, può avere una dimensione `n` che varia al variare del problema che si vuole risolvere. Ad esempio, il sistema lineare $A\mathbf{x} = \mathbf{b}$ ha dimensione n dove n è la dimensione della matrice e la lunghezza dei vettori. A seconda del sistema lineare da risolvere avremo un certo valore di n (5, 10, 20, 10000,....).

Il programma FORTRAN che, mediante un appropriato algoritmo - come il CGM - deve approssimare la soluzione x del sistema, deve poter risolvere sistemi lineari fino ad una dimensione che chiamiamo, ad esempio, $nmax$.

In FORTRAN77, dunque, (non ci riferiamo al FORTRAN 90 dove c'è la locazione dinamica delle variabili) nel programma principale bisogna dichiarare i vettori dando la loro dimensione massima. In base a queste dimensioni massime ci sarà un certa locazione di memoria riservata per i vettori. Invece nelle `functions` e `subroutines` si potrà dare la loro dimensione effettiva, una volta che è nota la dimensione effettiva dei vettori.

Analogo discorso vale per tutti i vettori: nel programma principale si deve dare la loro dimensione massima, nei sottoprogrammi si può dare la loro dimensione effettiva[1].

Come dichiarare, dunque, la dimensione massima di un vettore?

Lo vediamo in due modi. Supponiamo che la dimensione massima sia 10000 per il vettore x.

Nella dichiarazione delle variabili nel programma principale, scriveremo

```
real*8 x(10000)
```

[1] Il discorso sul passaggio della dimensione dei vettori è più complicato di come lo stiamo presentando ma qui ci limitiamo a queste poche osservazioni.

Al posto di 10000 si può introdurre un parametro nmax che rappresenta la dimensione massima del vettore e che vale 10000:

```
integer nmax
parameter(nmax=10000)
real*8 x(nmax)
```

Indicando con n la dimensione effettiva del vettore x, una volta che si conosce tale valore, si potrà dichiarare x in un sottoprogramma (function o subroutine) con la sua effettiva dimensione

```
integer n
real*8 x(n)
```

Queste istruzioni possono essere scritte in un sottoprogramma ma non nel programma principale in quanto, nel programma principale, non è ancora noto il valore di n e quindi non è possibile sapere quanta locazione di memoria occuperà la variabile x.

2.8 Sulla dimensione massima dare ai vettori JA e SYSMAT

Quando i vettori JA e IA, che individuano la topologia della matrice di rigidezza, si costruiscono a partire dalla griglia triangolare (e si ha come riferimento la subroutine topol.f), bisogna dare una dimensione massima ai due vettori.

Mentre per IA sappiamo che la dimensione è n +1 dove n è il numero dei nodi - e quindi la dimensione massima è uguale alla dimensione massima dei nodi cui si aggiunge uno (per esempio nmax+1) - , per quanto riguarda JA bisogna tener presente che, all'interno della subroutine topol.f il massimo numero di elementi non nulli che possono essere memorizzati in JA è dato da nt=n1 · n dove n1 rappresenta il massimo numero di contatti nodali.

Quindi la dimensione massima con cui dichiarare JA (e di conseguenza SYSMAT) non può essere inferiore a n1·nmax. Indicado con ntmax il numero massimo di elementi di JA e SYSMAT si può fare un controllo all'interno del programma per verificare che nmax≥ n1·nmax. In caso contrario, infatti, il vettore JA può risultare molto diverso da quello che si deve ottenere.

2.9 Cosa significa iout nella kersh.f ?

La subroutine kersh.f ha tra i parametri di input un'unità chiamata iout:

```
subroutine kersh(iout,nequ,nterm,ia,ja,sysmat,prec)
```

iout è dichiarata come variabile intera e viene utilizzata nelle seguenti righe della subroutine

```
      if(a.le.zero) then
          write(iout,100) k,a
          write(iout,101) prec(ia(kk-1))
         a = (prec(ia(kk-1)))**2
      end if
     :
     :
     if(a.le.zero) then
         write(iout,100) nequ ,a
         write(iout,101) prec(ia(nequ-1))
         a = (prec(ia(nequ-1)))**2
      end if
      prec(k) = sqrt(a)
      :
      :
100   format('*** Subroutine Kersh: diagonal',
     1       ' element <= zero at position: ',I5,2X,E16.5)
101   format('***** using previous diagonal value: ',E16.8)
```

Quindi iout identifica il file dove vengono scritti dei messaggi di avvertimento relativamente ad elementi diagonali che dovrebbero essere radici quadrate di valori negativi, ragion per cui il valore di quell'elemento diagonale viene sostituito dal valore dell'elemento diagonale immediatamente precedente.

Perciò nel programma principale dove viene chiamata la kersh conviene aprire un file con un'unità che corrisponderà a quella che si darà poi come input nella subroutine stessa.

Esempio

Nel programma principale apriamo il file 'messaggikersh.txt':

```
open(13, file='messaggikersh.txt', status='unknown')
```

Quando chiamiamo la kersh scriviamo

```
call kersh(13, nequ,nterm,ia,ja,sysmat,prec)
```

All'interno della subroutine kersh non dobbiamo fare alcuna modifica perchè alla variabile iout verrà associato il valore 13 che passiamo noi in input.

Se, all'interno della subroutine, si entra nei cicli if che determinano l'esecuzione delle istruzioni di scrittura prima dette, allora verrà scritto il file 'messaggikersh.txt', altrimenti non verrà scritto nessun messaggio sul file e quindi, dopo l'esecuzione del file, non troveremo nessun file denominato 'messaggikersh.txt'.

2.10 Problemi in AUTOCAD con lo script griglia.scr

Se l'AUTOCAD con il quale vogliamo visualizzare la griglia legge ed esegue tutte le istruzioni in italiano, allora non riuscirà a comprendere cosa voglia dire LINE - in inglese - perchè ha bisogno della medesima istruzione in italiano, vale a dire LINEA.

Il problema può essere risolto in due modi: manualmente si corregge il file griglia.scr scrivendo LINEA al posto di LINE oppure si va a modificare il programma FORTRAN griglia_cad.f.

La variabile COMMAND deve essere allora dichiarata come CHARACTER*5 e non CHARACTER*4 perchè LINEA è costituita da 5 caratteri diversamente da LINE che è costituita da 4 caratteri.

L'istruzione `WRITE(7,'(A4)') COMMAND`
deve essere corretta con
`WRITE(7,'(A5)') COMMAND`.

Si compila e si esegue il codice con queste correzioni e si otterrà un file di output che potrà essere letto in AUTOCAD senza problemi.

2.11 Per chi non lavora in FORTRAN

Per coloro che utilizzano un linguaggio di programmazione diverso dal FORTRAN, come l'ambiente MATLAB o il Pascal o un altro ancora, e vuole tradurre le subroutines messe in rete a disposizione degli studenti, deve tenere conto di alcune caratteristiche proprie del FORTRAN per evitare di tradurre in maniera scorretta una suboroutine.

Qui nel seguito diamo alcuni consigli utile per tradurre la subroutine kersh.f.

In FORTRAN l'istruzione ELSE IF si puo' scrivere come ELSE IF o ELSEIF. In altri linguaggi, come ad esempio in MATLAB, l'analoga istruzione deve essere scritta tutta attaccata ELSEIF altrimenti si apre un altro ciclo IF.

In FORTRAN un ciclo del tipo `DO k=i,j ... END DO` dichiara la variabile k come variabile intera e, appena entra nel ciclo la pone uguale a i $k = i$. Se $j < i$ il ciclo non dà risultati in uscita ma la variabile k rimane comunque del valore uguale a i. Invece, quando $i < j$, il ciclo DO va avanti e k viene incrementato di una unita'. Quando $k = j$, si lavora per l'ultima volta dentro il ciclo perche' poi k viene ancora incrementato di un'unita' e quindi il suo valore diventa $k = j + 1 > j$ e si esce dal ciclo. Il valore in uscita di k e' dunque $j + 1$. In PASCAL l'analogo ciclo dà in uscita un valore di $k = j$. In MATLAB, invece, una volta che si esce da un ciclo `do k=i:j` la variabile k e' come cancellata e quindi non puo' essere utilizzata come variabile all'esterno del ciclo.

Si puo' dunque osservare come la variabile $k2$, nella subroutine FORTRAN kersh.f, e' usata all'esterno del ciclo `do k2=i,j-1` con il valore $k2 = j$ (sia se $i = j$ sia per $i < j$)

```
if(j.ge.i) then
```

```
        prec(ia(ja(j))) = prec(ia(ja(j))) + prec(k2)**2
      end if
```

Lo stesso discorso non si puo' fare in MATLAB o in PASCAL perche' k2 viene cancellato o ha un valore diverso da quello che si ha in FORTRAN al di fuori del ciclo. In MATLAB, ad esempio, si deve scrivere

```
 if  j >= i
      prec(ia(ja(j))) = prec(ia(ja(j))) + prec(j)^2
end
```

2.12 Un primo test sul CGM

Una volta scritto e compilato il codice sul CGM, i risultati che si ottengono possono non essere quelli giusti perchè, ad esempio, l'algoritmo non è stato ben implementato o una `subroutine` presenta qualche errore. Per controllare passo dopo passo ciò che il codice fa conviene provarlo su sistemi lineari, simmetrici e definiti positivi di piccole dimensioni.

Una matrice che soddisfa tali requisiti è

$$\begin{pmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{pmatrix}$$

Si può dunque testare il GCM per la soluzione del sistema lineare $A\mathrm{x} = \mathrm{b}$ dove A è la matrice data prima (con una dimensione $n = 4$ o $n = 10$...) e il vettore termine noto può essere scelto in modo che la soluzione del sistema sia il vettore con tutti 1.

Costruire i vettori `SYSMAT, JA, IA` per questa matrice è difatti immediato per piccole dimensioni.

2.13 Esempio test: sulla matrice test 112×112

La matrice 112×112 può dare risultati di convergenza o meno alla soluzione del sistema a seconda del numero di iterazioni preliminari effettuate con lo schema delle Correzioni Residue (CR).

Quando si implementa lo schema CR con la decomposta incompleta di Cholesky, si osserverà come, per questa matrice, lo schema CR non converga ma diverga (in quanto il residuo relativo che viene calcolato aumenta sempre più).

Perciò, solo utilizzando poche iterazioni preliminari con lo schema CR, il metodo del CGM convergerà alla soluzione cercata. Invece, aumentando il numero delle CR (con 5 CR il fenomeno è già ben visibile) si osserverà come il CGM non dà la soluzione esatta del sistema. Il residuo relativo calcolato utilizzando la formula del CGM diventa minore della tolleranza prefissata, ma il residuo calcolato tramite la formula $\mathrm{b} - A\mathrm{x}_k$ dà un valore molto distante dalla tolleranza prefissata e la soluzione che si ottiene è anch'essa molto lontana dalla soluzione esatta.

Invece, utilizzando il precondizionatore diagonale, con 10 o 30 iterazioni preliminari CR, pur notando che lo schema CR diverge, il CGM converge alla soluzione esatta, con un numero, comunque, elevato di iterazioni. Aumentando ulteriormente il numero delle Correzioni Residue, tuttavia, anche con il precondizionatore diagonale si osserva divergenza dalla soluzione esatta del CGM.

Il fatto che lo schema CR diverga, con un residuo relativo che tende all'infinito molto rapidamente, fa sì che il CGM non riesca a sfruttare il fatto di ottenere dallo schema CR una soluzione iniziale con errore ortogonale agli autovettori del sistema precondizionato. Perciò non si riesce ad ottenere la soluzione esatta del sistema.

Ricordiamo che lo schema CR converge se e solo se il raggio spettrale della matrice di iterazione è minore di 1 (per lo schema CR applicato al sistema precondizionato è $I - K^{-1}A$).

Nel problema in esame (il sistema da risolvere con la matrice test 112×112) la matrice di iterazione dello schema CR ha perciò un raggio spettrale maggiore di 1.

Un'ultima osservazione sulla matrice test: è un esempio di matrice mal condizionata.

2.14 Esempio test: sulla matrice test bcsstk04

Con la matrice test bcsstk04 132×132, i risultati vanno bene solo con poche iterazioni preliminari delle correzioni residue. Aumentando il numero delle correzioni residue, non si ha convergenza alla soluzione esatta del sistema, qualunque precondizionatore si usi. Da cosa dipende questo comportamento?

Il metodo delle Correzioni Residue, applicato alla matrice bcsstk04 di dimensione 132×132 proposta come esempio test, non converge alla soluzione esatta. Difatti, il residuo relativo fornisce valori via via crescenti. Nel caso in cui si applica la decomposta incompleta di Cholesky, la divergenza la si nota già dopo poche iterazioni, mentre nel caso del precondizionatore diagonale, devono essere applicate diverse iterazioni per osservare tale divergenza.

Quindi poche iterazioni preliminari delle Correzioni Residue permettono comunque, al CGM, di abbattere alcune componenti dell'errore e di arrivare a convergenza (arrivando alla soluzione esatta del problema a meno della tolleranza richiesta). Invece, aumentando il numero di CR il punto di partenza che viene dato al CGM dista notevolmente dalla soluzione esatta: il CGM arriva a convergenza (il residuo relativo calcolato con la formula ricorsiva del CGM diminuisce fino alla tolleranza richiesta, non così per il residuo calcolato con la formula vera) ma la soluzione

che si ottiene è ben lontana dalla soluzione del problema. Tale fenomento è ben visibile già dopo 10 correzioni preliminari delle CR nel caso di precondizionatore di Cholesky e dopo oltre 80 iterazioni preliminari delle CR con il precondizionatore diagonale.

Vediamo più in dettaglio il comportamento del metodo CR. Nel caso del precondizionatore di Cholesky, dopo 10 iterazioni del metodo CR, il residuo relativo è circa 125, dopo 15 iterazioni è cresciuto a circa 11370, dopo 20 iterazioni è dell'ordine di 10^{18}. Il metodo diverge molto rapidamente.

Nel caso del precondizionatore diagonale, dopo 10 iterazioni il residuo relativo è dell'ordine di 10^{-2}, dopo 15 è sempre dell'ordine 10^{-2}, dopo 20 è aumentato a 10^{-1}, dopo 40 a 10^{2}, dopo 70 è dell'ordine di 10^{7}, dopo 80 è dell'ordine di 10^{9}. Il metodo diverge più lentamente rispetto all'altro tipo di precondizionatore e, infatti, gli effetti sulla soluzione del CGM si notano utilizzando un maggior numero di iterazioni preliminari delle CR.

In entrambi i casi, la matrice di iterazione del metodo CR ha raggio spettrale maggiore di 1.

Ricordiamo che il metodo delle CR ha lo scopo di abbattere alcune componenti dell'errore rispetto agli autovettori corrispondenti, e quindi non è necessario che il metodo CR converga perchè si verifichi ciò.

Tuttavia, se il metodo diverge molto velocemente allontanandosi dalla soluzione con errori via via crescenti, (come avviene con la matrice proposta in questo caso test) allora solo poche iterazioni delle CR fanno sì che il vantaggio di abbattere alcune componenti dell'errore acceleri il CGM, altrimenti la soluzione iniziale che viene data al CGM, essendo molto lontana dalla soluzione del problema e con un errore molto elevato, porta ad una soluzione sballata anche nel CGM, pur osservando che il residuo relativo (calcolato facendo uso della formula dell'algoritmo del CGM) decresce fino a raggiungere la tolleranza prefissata. Invece il residuo relativo vero rimane molto lontano dalla tolleranza prefissata, (per osservare meglio il fenomeno, si può fare un grafico in cui si mette a confronto il residuo relativo calcolato con la formula ricorsiva del CGM con il residuo relativo vero, iterazione per iterazione).