Fortran "in pillole": seconda parte

Annamaria Mazzia

Dipartimento di Metodi e Modelli Matematici per le Scienze Applicate

Corso di Metodi Numerici per l'Ingegneria

Algoritmo della Regula Falsi

Consideriamo l'algoritmo della Regula Falsi (o secante variabile) per trovare un'approssimazione della radice di una funzione f (in una variabile reale).

- dati di input: x_0, x_1, *itmax*, *tol*
- algoritmo da implementare: $x_{n+1} = x_n - \dfrac{f(x_n)}{K}$, dove $K = \dfrac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$
- quante iterazioni dobbiamo fare per poter dire che un certo x_n è una buona approssimazione della radice ξ della f?

I predicati

`Fino a quando`
$n \leq itmax$ e $|x_{n+1} - x_n| > tol$ `allora`

- $n = n + 1$
- `si applica l'algoritmo della Regula Falsi.`

L'espressione $n \leq itmax$ e $|x_{n+1} - x_n| > tol$ è un'espressione logica il cui risultato può essere solamente o vero o falso.

Operatori di confronto

Operatore	Significato	Esempio	Valore
.GT.	$>$	(a.gt.b)	Vero se $a > b$ Falso se $a \leq b$
.GE.	$>=$	(a.ge.b)	Vero se $a \geq b$ Falso se $a < b$
.LT.	$<$	(a.lt.b)	Vero se $a < b$ Falso se $a \geq b$
.LE.	$<=$	(a.le.b)	Vero se $a \leq b$ Falso se $a > b$
.EQ.	$==$	(a.eq.b)	Vero se $a = b$ Falso se $a \neq b$
.NE.	$\sim=$	(a.ne.b)	Vero se $a \neq b$ Falso se $a = b$

Operatori logici

Un predicato "composto" - come quello visto per la Regula Falsi - è composto da predicati semplici messi insieme da operatori logici del tipo "e", "oppure", "non". Si ha

Operatore	Significato	Esempio
.NOT.	negazione	.not.(P)
	~	
.AND.	congiunzione	(P1).and.(P2)
	&	
.OR.	disgiunzione inclusiva	(P1).or.(P2)
	|	

Operatore	Esempio	Valore
.NOT.	.not.(P)	P vero $\Longrightarrow$ .not.(P) falso P falso $\Longrightarrow$ .not.(P) vero
.AND.	(P1).and.(P2)	P1 e P2 veri $\Longrightarrow$ (P1).and.(P2) vero P1 o P2 falsi $\Longrightarrow$ (P1).and.(P2) falso
.OR.	(P1).or.(P2)	o P1 o P2 veri $\Longrightarrow$ (P1).or.(P2) vero P1 e P2 falsi $\Longrightarrow$ (P1).or.(P2) falso

Ci sono delle priorità sull'ordine con cui vengono presi gli operatori in FORTRAN. Comunque conviene sempre usare le parentesi tonde.

Esempio

Traduciamo in FORTRAN il predicato
$n \leq itmax$ e $|x_{n+1} - x_n| > tol$

- Introduciamo le variabili `iter, itmax, scartix, tol`.
- Si ha

```
((iter.le.itmax).and.((scartix.gt.tol) ) )
```

La struttura alternativa

- struttura condizionale
- struttura while
- if logico

Ciclo if

```
if (espressione logica) then
        { istruzione 1a }
        { istruzione 2a }
        ⋮
else
        { istruzione 1b}
        { istruzione 2b}
        ⋮
endif
```

Se è vera l'espressione logica allora si eseguono le istruzioni 1a, 2a, ... Altrimenti - cioè se è falsa l'espressione logica - allora si eseguono le istruzioni 1b, 2b, . . .

```
if (espressione logica) then
        { istruzione 1a }
        { istruzione 2a }
        ⋮
endif
```

Se è vera l'espressione logica allora si eseguono le istruzioni 1a, 2a, . . . , altrimenti non si fa nulla.

```
if (espressione logica1) then
        { istruzione 1a }
        { istruzione 2a }
        ⋮
elseif (espressione logica2)
then
        { istruzione 1b}
        ⋮
elseif (espressione logica3)
then
        { istruzione 1c}
        ...
⋮
else
        { istruzione 1z}
        ...
endif
```

Esempio

```
if ((-0.5d0.le.x)
.and.(x.gt.0.5d0)) then
        a=10.0d0
        fun=a*dsin(x)
elseif ((0.5d0.le.x)
.and.(x.gt.1.5d0)) then
        a=0.5d0
        fun=a*dcos(x)
else
        fun=1.0d0
endif
```

Ciclo while

Do while (espressione logica)

{ istruzione 1 }

{ istruzione 2 }

⋮

{ istruzione n }

end do

Fintantochè è vera l'espressione logica allora esegui istruzione 1, 2, . . . , n.

Le istruzioni 1, 2, . . . vengono ripetute ciclicamente (non una volta sola come nel ciclo if).

Quando si esegue l'ultima istruzione posta all'interno del ciclo, si torna all'espressione logica e si controlla se è vera o falsa. Se è vera, si eseguono di nuovo le istruzioni 1, 2, . . . ,n. Se non è vera, si esce dal ciclo while.

Attenzione a non creare cicli infiniti!

Esempio

Riprendiamo il ciclo della regula falsi. Il controllo su un numero massimo di iterazioni che possono essere effettuate, permette di uscire dal ciclo se non si arriva a convergenza (e quindi gli scarti su x e su y rimangono sempre maggiori della tolleranza).

```
do while  ((iter.le.itmax).and.((scartix.gt.tol)))
        iter = iter +1
        :
end do
```

If logico

If (espressione logica) { istruzione }

Se l'espressione logica è vera, allora esegui solo questa singola istruzione.

I sottoprogrammi

Quando l'algoritmo e il problema su cui stiamo lavorando sono complicati, conviene spezzare il problema in sottoproblemi in modo da semplificare la programmazione.
Analogamente, al posto di avere un unico programma in FORTRAN, conviene scrivere il programma facendo uso di sottoprogrammi.
Si hanno due tipi di sottoprogrammi in FORTRAN:

- subroutines
- functions

In tal modo un programma FORTRAN può risultare composto da:

- programma principale
- una o più subroutines
- una o più functions

Esempio di subroutine

Modifichiamo il programma `gradirad.f` in `gradirad_sub.f`

- la subroutine è chiamata nel programma principale tramite l'istruzione

  ```
  call nome_subroutine(parametri)
  ```

- la subroutine è scritta **dopo** la end del programma principale
- la subroutine incomincia con

  ```
  subroutine nome_subroutine(parametri)
  ```

- il "corpo" della subroutine è analogo a quello di un programma principale (implicit none, dichiarazione delle variabili, istruzioni, cicli...)
- la subroutine si chiude con le istruzioni

  ```
  return
  end
  ```

- Il fatto che noi chiamiamo una subroutine nel programma principale (`call`) dice che la subroutine non è un programma a sè stante. Quando termina l'esecuzione di ciò che è scritto all'interno della subroutine si torna indietro nel programma principale e si continua l'elaborazione da quel punto. L'istruzione `return` fa tornare al programma principale.
- Le variabili non devono avere necessariamente lo stesso nome nel programma principale e nella subroutine

  `call datin(xgradi, xprimi, xsecondi)` $\Longleftarrow$ nel programma principale

  `subroutine datin(xg,xp,xs)` $\Longleftarrow$ nella subroutine

  ma devono avere lo stesso significato e devono essere messe nello stesso ordine: xg deve avere lo stesso significato di xgrado e deve essere dello stesso tipo nella dichiarazione delle variabili. Lo stesso dicasi per xp- xprimi e xs- xsecondi.

All'interno della subroutine si possono utilizzare altre variabili oltre a quelle che sono presenti tra i parametri della stessa. L'importante è dichiararle nella subroutine. Tali variabili non passano nel programma principale ma sono usate solo nella subroutine.
Esempio: un programma per calcolare gli zeri di una funzione in una variabile mediante il metodo della Regula Falsi (`regfalsi_sub.f`).

Le functions

Dover scrivere ogni volta tutta l'espressione della funzione di cui cercare lo zero può portare facilmente a degli errori e inoltre rende più difficile la lettura del programma.
Modifichiamo il codice, introducendo una function per la f e una function per la derivata prima - anche se questa viene usata solo una volta all'interno del programma.
Vediamo il codice modificato (`regfalsi_subfun.f`).

Osservazioni

- Una function viene chiamata nel programma (o nelle subroutine) direttamente - senza la call come avviene per le subroutine.

  ```
  x1new = xold - f(xold)/df(xold)
  ```

- La function restituisce un valore ben preciso - il valore assunto dalla funzione stessa in funzione dei parametri. Perciò deve essere dichiarato il tipo della function (integer, real, real*8, ...)
- La function restituisce un solo valore:

  ```
  f= exp(2*x ...)
  ```

- La function può avere uno o più parametri in ingresso.
- Come per le subroutine, si usa l'istruzione `implicit none`, la dichiarazione delle variabili usate, tutte le istruzioni, e infine `return, end`.

Il formato

Abbiamo stampato i dati che interessano sul terminale.
Per avere un output elegante e ordinato, conviene usare l'istruzione format.
Vediamo il programma `regfalsi_format.f`.

Osservazioni

- Usare il formato mediante l'istruzione `format` piuttosto che all'interno della `write`.
- Un numero (etichetta) all'interno della `write` è collegato al formato che viene usato per quella stampa.

formato	Significato
i	formato intero
e	esponenziale
f	fisso
a	alfanumerico
x	spazi bianchi

formato	Esempio
i	i5 - 5 caratteri per un intero
e	e14.6 - 14 caratteri - 6 per la mantissa
	e14.6 - 14 caratteri - 6 per la mantissa
	e18.10 - 18 caratteri - 10 per la mantissa
f	f14.6 - 14 caratteri - 6 per le cifre decimali
f	f15.12 - 15 caratteri - 12 per le cifre decimali
a	a5 - una stringa di al più 5 caratteri
x	1x - 1 carattere bianco
	3x - 3 caratteri bianchi

Specificando il formato, occorre prestare attenzione al fatto che non tutti i numeri possono essere stampati correttamente. Per esempio se un intero ha più di 5 caratteri (per esempio 100150) ma il formato per esso è i5, vengono stampati degli * o altri caratteri a seconda del compilatore. Se si hanno strani risultati in output usando un formato: togliere il formato, compilare e rieseguire il programma per verificare se l'errore dipende dal formato!

Files di dati

La stampa dei risultati più che stamparla sul video del computer conviene salvarla in un file di dati, in modo da poterlo utilizzare in seguito (per esempio per fare i grafici)
Le istruzioni più semplici per ottenere questo risultato sono:

- aprire un file all'interno del programma principale associando ad esso un numero (label).
- le istruzioni di `write` saranno poi associate a quella label e scritte su quel file.
- chiudiamo il file con l'istruzione close.

Vediamo un esempio in `regfalsi_subfunformfile.f`

Sulla lettura dei dati di input

Un file di dati può essere usato, alla stessa maniera per leggere i dati di input.
Modifichiamo a tal proposito il programma sulla conversione gradi-radianti `gradirad_fileinp.f`

Vettori e matrici

Supponiamo ora di dover lavorare con vettori.
Come memorizzarli? Come lavorare con i vettori?
Scriviamo un programma che legge due vettori ed esegue il prodotto scalare tra essi in `prodscal.f`

Osservazioni

- in FORTRAN 77 ai vettori (e alle matrici) occorre dare una dimensione massima, come effetto della loro locazione di memoria `statica`. Perciò usiamo il parametro nmax (e abbiamo introdotto l'istruzione parameter).
- Ci sono due modi equivalenti per leggere i dati da file - il cosiddetto `do implicito` e il `do esplicito`.
- Il `do esplicito` viene usato anche successivamente per fare il prodotto scalare tra i due vettori.
- Il ciclo `do i=1,n` è equivalente ad un ciclo `do while (i.le.n)` con *i* che varia da 1 a *n* con incremento 1.
- Per fare il prodotto scalare, abbiamo introdotto una variabile di accumulo

Sul ciclo do

In genere, nel ciclo `do` si può avere anche un incremento diverso da 1.
Si ha, in generale:

```
 do indice = indiceinizio, indicefine, incr
        istruzioni
end do
```

La variabile intera indice parte da indiceinizio e viene incrementata della quantità incr, fino ad arrivare a indicefine.

Cenni sulle matrici

- Una matrice viene dichiarata, ad esempio, come
  `real*8 A(nmax,nmax)`
- Nel file di dati, si scrive la matrice come siamo abituati a vederla, in modo che la lettura possa essere fatta con le istruzioni:

```
do i=1,n
     read(10,*) (A(i,j), j=1,n)
end do
```

- I cicli `do` si usano sia per l'indice di riga che per l'indice di colonna.
- Perché solo dei cenni? Perchè quando lavoreremo con le matrici, le memorizzeremo in modo particolare utilizzando dei vettori.

Vettori e matrici nei sottoprogrammi

- Per le matrici occorre sempre dichiarare la dimensione massima sulle righe quando una matrice è un parametro di un sottoprogramma. Ad esempio
  `real*8 A(nmax,nmax)` nel programma principale
  `real*8 A(nmax,n)` nel sottoprogramma
  Quindi se A viene usata in un sottoprogramma, bisogna scrivere tra i parametri del sottoprogramma sia A sia nmax, oltre che n.
- Per i vettori, basta dichiarare la dimensione effettiva nei sottoprogrammi e quindi passare solo n tra i parametri. Ad esempio
  `real*8 x(nmax)` nel programma principale
  `real*8 x(n)` nel sottoprogramma .