
Laboratorio
di
Calcolo Numerico

Luca De Franceschi

Università degli studi di Padova

Indice

1 Esercitazione 2

1.1 Ambienti software dedicati al calcolo scientifico

È ben noto che per creare programmi al “calcolatore” sia necessario disporre di un “ambiente di programmazione”. Tale ambiente, in parte inserito nel sistema operativo ed in parte costituito da software aggiuntivo, fornisce strumenti indispensabili alla creazione e verifica dei programmi, come: un editor, un compilatore, un linker, un debugger, ...

Un ambiente software dedicato al calcolo scientifico aggiunge a questi strumenti degli altri strumenti dedicati allo sviluppo di applicazioni che utilizzano principalmente le capacità di calcolo del calcolatore. Da non confondere l'ambiente software dedicato al calcolo scientifico con la libreria di calcolo scientifico, che fornisce “solamente” un insieme di metodi numerici, richiamabili da un programma.

Per applicazioni del calcolo scientifico vengono tipicamente utilizzate assieme più d'una libreria. Ambienti molto diffusi di questo tipo sono: **Matlab** (il capostipite, 1983 ca., commerciale), **Octave** (open-source), **Scilab** (open-source). Recentemente, si sta affermando l'utilizzo di **Python**, grazie alla sua shell interattiva ed alle estensioni numeriche disponibili.

1.1.1 Il workspace

L'ambiente di calcolo, che si presenta con una linea di comando, mantiene disponibili in memoria tutte le variabili che sono state inizializzate dal momento dell'ingresso nell'ambiente, sia tramite singoli comandi che tramite istruzioni contenute nei programmi eseguiti, entrambi impartiti dalla linea di comando dell'ambiente. Questa è una sostanziale differenza rispetto alla programmazione in C o in FORTRAN, dove le variabili vengono allocate all'inizio dell'esecuzione del programma e poi de-allocate (cioè cancellate) al termine dell'esecuzione stessa. Una conseguenza immediata di questo fatto è che, ad esempio, in un ambiente come il Matlab non è necessario salvare esplicitamente su file i valori delle variabili che si vogliono osservare (es. graficare) dopo l'esecuzione del programma.

1.1.2 La grafica

Non è sufficiente produrre dei (buoni) risultati numerici, bensì è anche necessario poterli osservare a valutare in modo adeguato. Solitamente questa operazione richiede la produzione di qualche tipo di grafico, per cui l'esistenza di capacità grafiche è un elemento fondamentale di un ambiente software dedicato al calcolo scientifico. Matlab possiede capacità grafiche evolute ed Octave si appoggia a Gnuplot (applicativo anch'esso open-source).

1.1.3 L'esecuzione dei programmi

Per mandare in esecuzione un programma scritto per Matlab o per Octave (e quindi un file di testo con il suffisso “.m” contenente istruzioni che rispettano la sintassi del linguaggio di Matlab, ovvero un “m-file”) è sufficiente scrivere il nome del m-file sulla riga di comando e premere il tasto invio. Se il

programma non è nella directory corrente, è necessario prima mettersi nella directory che lo contiene. Il calcolo dei tempi di esecuzione è semplice, e quindi è possibile misurare la velocità di esecuzione dell'algoritmo numerico: conoscendo il numero di operazioni floating-point (flops) richieste per il calcolo della soluzione, si può calcolare la velocità $MFLOPS = \text{numero}_{\text{flops}} / \text{tempo}_{\text{rascorso}}$.

Notare che il tempo di esecuzione di un algoritmo numerico è oggi il principale parametro per misurarne l'efficienza. Infatti, mentre fino a qualche anno fa le operazioni in virgola mobile venivano fatte via software/firmware, e dunque una moltiplicazione costava più di un'addizione, ora sono implementate via hardware, e dunque richiedono sostanzialmente lo stesso numero di cicli di clock del processore. Pertanto, mentre fino a qualche tempo fa il parametro più usato era il numero di operazioni floating-point richieste, ora lo è il tempo di esecuzione dove, attenzione, pesano anche gli aspetti non numerici dell'algoritmo (esistenza di istruzioni di scelta, movimento dei dati da e verso la memoria, ...).

Per quanto riguarda l'ambiente di calcolo, è necessario tenere presente che Matlab (ed Octave) eseguono il programma utente in maniera interpretata, e cioè viene letta, compilata ed eseguita un'istruzione alla volta. Questo incide moltissimo sulla velocità di esecuzione di un algoritmo. In particolar modo è bene evitare di creare troppi cicli innestati.

1.1.4 Il linguaggio di programmazione “domain specific”

L'ambiente Matlab (Octave) mette a disposizione un linguaggio di programmazione che da un lato contiene tutti i costrutti tipici della programmazione strutturata (if-then-else, ciclo for, ciclo while, ecc...) e dall'altro permette di manipolare in modo molto semplice e sintetico le strutture dati fondamentali per il calcolo matriciale, e cioè vettori e matrici, in modo analogo a quanto viene fatto dal linguaggio tradizionale più evoluto per il calcolo, il Fortran90 e successive varianti, ed a quanto può essere fatto a partire dai linguaggi orientati agli oggetti, come il C++ , Java, ...

Il linguaggio di Matlab (Octave) offre il vantaggio, rispetto ai linguaggi di programmazione tradizionali, di semplificare molto l'uso del linguaggio per la costruzione di un programma, lasciando quindi l'attenzione del programmatore libera il più possibile dagli aspetti puramente informatici e concentrata sugli aspetti algoritmici e numerici. Anche per questo è molto diffuso, sia come strumento di studio che come strumento di lavoro, in particolare per la prototipazione dei codici di calcolo. Per un utilizzo in “produzione”, i compilatori dei linguaggi tradizionali sono però preferibili perchè permettono di ottimizzare maggiormente il codice eseguito dalla macchina e dunque, generalmente, l'algoritmo implementato viene eseguito più velocemente (che, nella realtà delle applicazioni, è molto rilevante).

1.2 Rappresentazione dei numeri al calcolatore

*I calcolatori oggi in commercio rispettano in genere il medesimo standard per la rappresentazione dei numeri in virgola mobile (floating-point): lo standard **IEEE 754**.*

Nello standard IEEE la precisione singola occupa una parola da 32 bit, mentre la precisione doppia occupa due parole consecutive da 32 bit. Un numero non-nullo normalizzato “X” ha la seguente rappresentazione binaria:

S E F

$$X = (-1)^S * 2^{(E - \text{bias})} * (1.F)$$

dove $\text{bias} = 127$ (single) o 1023 (double)

Lo standard richiede che il risultato delle operazioni di addizione, sottrazione, moltiplicazione e divisione sia arrotondato esattamente, cioè il risultato deve essere calcolato esattamente e poi arrotondato al numero in virgola mobile più vicino.

Problemi in tal senso provengono dalla sottrazione. Per questo motivo, nei microprocessori moderni la sottrazione viene effettuata utilizzando la tecnica bit di guardia: l’operando più piccolo viene troncato a $p+1$ cifre binarie, mentre il risultato della sottrazione viene troncato a p cifre (binarie). In realtà, il calcolo con un solo bit di guardia non sempre produce lo stesso risultato che si avrebbe calcolando il risultato esatto e poi arrotondando a p cifre binarie. Introducendo un secondo bit di guardia ed un terzo bit sticky, si ottiene il pieno rispetto dello standard con un extra-costo computazionale relativamente piccolo.

1.3 Propagazione degli errori di arrotondamento

Ogni calcolatore ha a disposizione un numero finito M di cifre per rappresentare un numero reale. Se il numero da rappresentare possiede più di M cifre significative, il calcolatore lo arrotonda alle M cifre più significative. A causa di questo fatto, le operazioni aritmetiche di base sono in generale soggette ad un errore nel risultato, ed è dunque necessario conoscere l’entità di questo errore e , considerando un intero algoritmo, l’effetto potenziale della propagazione di questo tipo di errore sul risultato finale. In generale, la somma, la divisione e la moltiplicazione producono un errore relativo piccolo, mentre la sottrazione può anche produrre un errore relativo grande, rispetto al risultato (ciò avviene quando i due operandi sono molto vicini tra di loro e si ha dunque una notevole perdita di cifre significative nel risultato).

Il fatto che l’errore relativo nel risultato sia piccolo non è comunque una garanzia di accuratezza in generale: la somma tra due operandi troppo distanti in magnitudo può addirittura far scomparire dal risultato il più piccolo dei due ed avere comunque un errore relativo piccolo. Per questo motivo, ad esempio, quando si deve eseguire una sommatoria, è buona regola eseguirla mettendo gli operandi in ordine crescente, in modo che la somma parziale ed il prossimo numero da sommare siano il più vicini possibile in magnitudo. Vediamo di seguito alcuni esempi relativi a questi fenomeni:

1.3.1 Calcolo delle radici di un’equazione di secondo grado

Vediamo ora un esempio concreto di calcolo in cui introdurre una sottrazione potenzialmente pericolosa conduca effettivamente a problemi di instabilità della soluzione, e come rimediare.

Data: $a * x^2 + b * x + c$, calcolare le sue radici.

La formula classica, $x = (-b \pm \sqrt{b^2 - 4ac}) / (2a)$, è potenzialmente instabile a causa della sottrazione tra $b/2$ e $\sqrt{b^2 - 4ac}$. Provare ad implementarla e verificarne la stabilità. Ripetere poi lo stesso tipo di indagine su una formula alternativa, stabile, che si ottiene calcolando prima la radice positiva (in cui non si effettua la sottrazione) e poi calcolando l'altra sapendo che vale: $c = a * x_1 * x_2$.

1.3.2 Approssimazione del pi greco

È un altro esempio di sottrazione pericolosa, questa volta in un metodo numerico iterativo per l'approssimazione di π -greco. Provare ad implementare i seguenti tre metodi iterativi ed a graficarne l'errore relativo, in scala logaritmica, per le prime 100 iterate (un valore molto accurato di π -greco in Matlab/Octave è contenuto nella variabile di sistema "pi"). Per $n = 2, 3, 4, \dots$ calcolare le tre approssimazioni di π -greco, $y(n)$, $w(n)$ e $z(n)$:

1. $s(2) = 1 + 1/4$;
 $s(n+1) = s(n) + 1/((n+1)^2)$; $y(n+1) = \sqrt{6 * s(n+1)}$; $w(2) = 2$;
 $w(n+1) = 2^{(n-1)/2} * \sqrt{1 - \sqrt{1 - 4^{(1-n)} * w(n)^2}}$;
2. $z(2) = 2$; $z(n+1) = (\sqrt{2} * z(n)) / (\sqrt{1 + \sqrt{1 - 4^{(1-n)} * z(n)^2}})$;

1.3.3 Successioni calcolabili praticamente

Si consideri la seguente successione:

$$y(0) = 1/e; y(n+1) = 1 - (n+1) * y(n);$$

Essa converge (in aritmetica con infinite cifre) a zero per $n \rightarrow \infty$. Implementarla e verificare cosa accade nel calcolatore, cercando di giustificare i risultati. Inoltre, sapendo che $y(n)$ è circa $= 0$ per n sufficientemente grande, provare ad implementarla all'indietro e vedere cosa risulta per $y(0)$. Perché?