software Apple

a cura di Valter di Dio

Prima parte

Precisione Multipla per Applesoft

di Stefano Laporta - Bologna

Introduzione

Se si dovesse stilare una classifica dei linguaggi più utilizzati sugli Apple, sicuramente il primo posto toccherebbe al Basic Applesoft.

Il fatto che questo linguaggio sia incorporato nelle ROM di sistema, che sia pronto all'uso senza schede aggiuntive, e che sia ben documentato ha reso questa particolare versione del linguaggio Basic il linguaggio più utilizzato sugli Apple. Negli anni passati sono stati scritti in Applesoft migliaia di programmi nei campi più disparati, ed in particolare anche nel settore matematico-scientifico dove, se si vuole essere onesti, l'Applesoft non brilla certo per le prestazioni.

Il suo ben conosciuto difetto è la scarsa precisione delle variabili: 9 cifre possono essere sufficienti per un uso generale, ma certo non per programmi a sfondo scientifico.

Altri calcolatori più adatti ad un uso scientifico come gli HP o i Casio FX forniscono all'utente precisioni ben superiori. Al povero utente Apple non è rimasta altra soluzione che o adattarsi a quello che la macchina permetteva o, in casi più disperati, procurarsi una scheda con un altro microprocessore (tipicamente lo Z80) che supportasse altre versioni del Basic od altri linguaggi con prestazioni superiori.

Passando a tempi più recenti, l'arrivo dei nuovi modelli Apple non ha migliorato di molto la situazione.

Infatti per quanto il nuovo Apple IIGS possa facilmente raddoppiare la velocità di esecuzione di vecchi programmi Applesoft, non può certo aumentare la precisione di calcolo; per ottenere significativi miglioramenti in tal senso bisognerà aspettare il nuovo linguaggio Basic con le routine matematiche SANE in precisione multipla, ma ahimè tutto questo serve molto poco a chi non possiede un IIGS.

Ma è veramente utile fare calcoli ad alta precisione? A prima vista, a meno di non dover far calcoli per porre in orbita satelliti artificiali, 8-9 cifre di precisione sembrerebbero essere più che sufficienti; purtroppo questo non è sempre vero.

Alcuni algoritmi tendono ad accumulare gli errori effettuati nei calcoli elementari con la conseguenza che i risultati finali sono molto meno precisi di quanto ci si aspetterebbe.

Oltre a questa, vi sono altre cause che possono ridurre la precisione dei risultati di un calcolo, cause che talvolta sono poco conosciute o sottovalutate.

Conseguenza di questo: si incolpa il calcolatore o il programma di «errori», o peggio, si prendono per buoni risultati errati.

Perdite di precisione

Le principali cause che possono portare a perdite di precisione nel risultato di un calcolo sono queste:

Errori di conversione di base
Errori di arrotondamento
Propagazione degli errori
Errori di Overflow
Bug

Vediamole in dettaglio, una per una:

Errori di conversione di base

I calcolatori, in genere, rappresentano internamente i numeri «a virgola mobile» in notazione esponenziale, cioè con una mantissa di un certo numero di cifre significative moltiplicata per una potenza della base di rappresentazione, che normalmente è 2 oppure 10.

Se la base è 2 si parla di rappresentazione binaria, nel secondo caso si tratta di rappresentazione BCD.

Per esempio il numero 17 può essere rappresentato

in BCD: 0.17 *10^2
in base 2: 0.10001 *2^5

La rappresentazione BCD è, per così dire, «più umana»: ha il vantaggio di rappresentare internamente i numeri così come vengono mostrati sullo schermo, ma è intrinsecamente più lenta nell'esecuzione dei calcoli rispetto alla rappresentazione binaria.

La rappresentazione binaria invece ha il vantaggio di una rapida esecuzione dei calcoli, ma obbliga il calcolatore ad effettuare un pesante lavoro di conversione binario/decimale ad ogni istruzione di INPUT/OUTPUT.

Vediamo un esempio semplice (e sfortunato) di questo fatto, una semplice assegnazione:

A=0.1

Il numero 0.1 è ovviamente esprimibile esattamente in base 10, ma non in base 2, infatti:

$0.1 = 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} \ldots$

che in binaria si scrive

0.00011001100110011...

per cui 0.1 risulta essere un numero periodico se espresso in binario!

A chi rimane sconcertato di fronte a questa affermazione ricordo che, se in base 10 i soli numeri non periodici sono solo le frazioni con una potenza di 10 a denominatore, in base 2 lo saranno solo le frazioni con una potenza di 2 a denominatore, e questo non è il caso del numero precedente (1/10).

Purtroppo il calcolatore, dovendo convertire il numero da decimale a binario e potendo contenere un numero finito di bit, dovrà troncare in un certo punto lo sviluppo di cui sopra, per cui il numero che verrà

memorizzato nella variabile A non potrà mai essere esattamente 0.1, anche se (apparentemente) non è stato ancora eseguito alcun calcolo.

Da questo fatto nascono i risultati «strani» di alcuni test, come il sommare 10 volte 0.1 a −1 che non dà come risultato zero. Per poter eliminare questa potenziale fonte di imprecisione bisogna utilizzare dove possibile numeri che siano esprimibili esattamente in binario; se nell'esempio precedente si usa il numero 0.125 = 1/8 al posto di 0.1 non si avrà chiaramente alcuna perdita di precisione.

La rappresentazione binaria possiede un altro piccolo vantaggio rispetto al BCD: permette di risparmiare memoria.

Si può infatti dimostrare che a parità di occupazione di memoria la rappresentazione binaria permette una precisione superiore di circa il 20%.

Se una variabile possiede N byte di mantissa, può contenere numeri con una precisione (espressa in cifre decimali) pari a:

N•2 cifre nel caso BCD e
N•2.408 cifre nel caso binario
(più precisamente N•LOG10(256))

Quindi per esempio 5 byte di mantissa permettono 10 cifre decimali in BCD e 12 cifre decimali in binario.

Errori di arrotondamento

Questo tipo di errore è forse più conosciuto del precedente ed è causato dal troncamento che il calcolatore effettua sui risultati di ogni operazione o funzione matematica calcolata. Questo troncamento è reso obbligatorio dal fatto che il calcolatore usa una rappresentazione interna dei numeri con una mantissa di lunghezza finita.

Se il risultato di un calcolo possiede più bit del massimo gestibile dal calcolatore in quella circostanza (dipendente dal livello

Per chi vuole il listato

I listato principale di questo programma è molto lungo. In conseguenza di ciò, si è ritenuto opportuno non pubblicarlo, sia perché avrebbe occupato troppo spazio sulla rivista sottraendone ad altri argomenti, sia perché una digitazione senza errori di un listato così lungo appare poco probabile. Chi è interessato al programma può ordinare secondo il solito sistema, il disco in redazione.

Questo programma è disponibile su disco presso la redazione. Vedere l'elenco dei programmi disponibili e le istruzioni per l'acquisto a pag. 259.

Bug Applesoft

```
10  HOME : PRINT "TEST SU ALCUNI BUGS DELL'INTERPRETE": PRINT "A
    PPLESOFT": NORMAL : VTAB 9
1000  HOME : INVERSE : PRINT "TEST 1-BUGS NELL'ESECUZIONE DELLE"
    : PRINT "MOLTIPLICAZIONI": NORMAL : PRINT
1010  GOSUB 10000: GET A$
2000  HOME : INVERSE : PRINT "TEST 2-BUGS NELL'ESECUZIONE DI CON
    FRONTI": NORMAL
2010  PRINT : PRINT "VERIFICA SE I*0.1 E I*0.1 SONO UGUALI": PRINT

2030  GOSUB 20000
2040  GET A$
3000  HOME : INVERSE : PRINT "TEST 3-IMPRECISIONE NEL CALCOLO DE
    LLE": PRINT "RADICI QUADRATE": NORMAL
3010  PRINT : PRINT "CONTROLLA SE SQR(I*I)-I E' ZERO": PRINT "PE
    R I=1,..25": PRINT
3030  GOSUB 30000
3040  PRINT : PRINT "FINE DEL TEST": GET A$
3500  HOME : INVERSE : PRINT "TEST 4-IMPRECISIONE NEL CALCOLO DE
    LLE": PRINT "FUNZIONI TRIGONOMETRICHE": NORMAL : PRINT
3510  GOSUB 35000: GET A$
4000  HOME : INVERSE : PRINT "TEST 5-IMPRECISIONE NEL CALCOLO DE
    I LOGARITMI": NORMAL : PRINT
4010  GOSUB 40000: GET A$
5000  HOME : INVERSE : PRINT "TEST 6-IMPRECISIONE NEL CALCOLO DE
    LLE
POTENZE INTERE DI NUMERI INTERI": NORMAL : PRINT
5010  GOSUB 50000: GET A$
5200  HOME : INVERSE : PRINT "TEST 7-BUG NEL CALCOLO DELLA FUNZI
    ONE INT": NORMAL : PRINT : PRINT "DEFINITO A=1/I, B=I ,C=A*
    B": PRINT "CONTROLLA SE INT(A*B)=INT(C) PER I=1,40": PRINT

5210  GOSUB 52000: GET A$
9999  VTAB 20: PRINT "FINE DEL PROGRAMMA": END
10000  FOR I = 25 TO 29
10010 A = 1:B = 2 ^ I + 2 ^ (I - 23) - 2 ^ (I - 26)
10020  PRINT A" * "B" = "A * B;: IF A * B - B * A <  > 0 THEN  INVERSE
    : HTAB 30: PRINT "DIFF=";A * B - B * A;: NORMAL
10025  PRINT : PRINT B" * "A" = "B * A: PRINT : PRINT : NORMAL
10030  NEXT : RETURN
20000  FOR I = 0 TO 80 STEP 5
20005  PRINT "I="I"  ";
20010  IF I * 0.1 = I * 0.1 THEN  PRINT "CONFRONTO CORRETTO"
20020  IF I * 0.1 <  > I * 0.1 THEN  PRINT "CONFRONTO ERRATO
20030  NEXT
21010  RETURN
30000  FOR I = 1 TO 25
30010  IF  SQR (I * I) <  > I THEN  PRINT "SQR("I"*"I")-"I"=
    " SQR (I * I) - I
30020  NEXT
31001  RETURN
35000  FOR I = 14 TO 99
35010  IF 2 ^ ( - I) + 1 - 1 - 2 ^ ( - I) = 0 THEN : PRINT "SIN(
    2^-"I")/2^-"I"= " SIN (2 ^  - I) / 2 ^  - I: NEXT
35020  PRINT : PRINT "IL LIMITE DEVE ESSERE 1"
35040  RETURN
40000  FOR I = 16 TO 999
40010  IF 2 ^ ( - I) + 1 - 1 - 2 ^ ( - I) = 0 THEN : PRINT "LOG(
    1+2^-"I")/2^-"I"= " LOG (1 + 2 ^  - I) / 2 ^  - I: NEXT
40020  PRINT : PRINT "IL LIMITE DEVE ESSERE 1"
41001  RETURN
50000 : PRINT "3^2-9=           "3 ^ 2 - 9: PRINT "5^3-125=
     "5 ^ 3 - 125: PRINT "3^10-59049=     "3 ^ 10 - 59049: PRINT
    "19^7-893871739= "19 ^ 7 - 893871739
50010  RETURN
52000  FOR I = 1 TO 40
52010 A = 1 / I
52020 B = I
52030 C = A * B
52040  IF  INT (A * B) <  >  INT (C) THEN  PRINT "I="I;"  PARTE
    INTERA ERRATA"
52050  NEXT
53001  PRINT : PRINT "FINE DEL TEST": PRINT : RETURN
```

di precisione richiesta) i restanti bit dovranno essere necessariamente «eliminati» senza speranza.

Questa operazione di «eliminazione» di conseguenza comporta l'introduzione di un errore pari alla differenza tra il risultato «teorico» e quello effettivamente introdotto in memoria.

Per poter ridurre l'errore associato a questa eliminazione quasi tutti i calcolatori effettuano un processo di Arrotondamento: invece di eliminare brutalmente tutti i bit che non possono entrare nell'area di memoria destinata a contenere il numero, il numero viene arrotondato per difetto o per eccesso a seconda che il primo bit «eliminato» valga 0 od 1.

Così facendo l'errore massimo possibile non può superare 1/2 bit, mentre l'operazione di troncamento può portare ad errori di quasi un bit.

Facciamo un esempio:

Calcolo di 1/10

sia 1 che 10 sono numeri interi esatti ma, come abbiamo visto in precedenza, il risultato (0.1) non lo è.

Il risultato teorico espresso in binario dovrebbe essere

0.0001100110011001100110011001100110011001100
11.... ecc... ecc...

La mantissa binaria dei numeri sull'Apple è lunga 4 byte pari a 32 bit significativi, per cui la mantissa sarà arrotondata per eccesso a:

0.00011001100110011001100110011001101

Propagazione degli errori

Come visto in precedenza, in generale ogni risultato di calcoli può possedere un errore che interessa l'ultimo bit significativo. Se gli errori non superassero mai questo limite non vi sarebbero più problemi, ma purtroppo vi sono situazioni in cui processi di amplificazione o accumulazione portano a errori molto più significativi (e pericolosi).

Scriviamo per esempio

PRINT 1/3

avremo ovviamente

.333333333

Ora scriviamo

PRINT 1/3+600000−600000

Dato che sommiamo e sottraiamo la stessa quantità l'effetto sul risultato dovrebbe essere nullo e ci si aspetterebbe lo stesso risultato, invece compare:

.333251953

Cosa è successo?

Quando l'Apple ha eseguito la prima somma, che varrebbe 600000.333333333..., dato che può rappresentare numeri con un

Test Applesoft

```
1  DATA 104,168,104,166,223,154,72,152,72,96
5  FOR I = 768 TO 777: READ K: POKE I,K: NEXT
10  READ NK:PI =  ATN (1) * 4:L2 =  LOG (2)
20  DIM R1$(30),R2$(30),R3$(30)
30  HOME : INVERSE : PRINT "PROGRAMMA PER IL CONTROLLO DELLA
       PRECISIONE DELLE FUNZIONI ELEMENTARI     ": NORMAL : HTAB
    20: PRINT "STEFANO LAPORTA-1987"
40  PRINT : PRINT : PRINT : PRINT "QUESTO PROGRAMMA HA LO SCOPO
    DI VERIFICARE I LIMITI ED IL GRADO DI PRECISIONE   DELLE RO
    UTINE MATEMATICHE DEL BASIC     APPLESOFT."
45  PRINT : PRINT "IN BASE AL RISULTATO DI CALCOLI VIENE DETERMI
    NATA IL TIPO DI RAPPRESENTAZIONE   DEI NUMERI REALI, I SUOI
     LIMITI ED IL   GRADO DI PRECISIONE DEL CALCOLO DELLE   FUN
    ZIONI ELEMENTARI."
50  PRINT : FLASH : VTAB 24: HTAB 15: PRINT "ATTENDERE";: NORMAL

100  GOSUB 10000: GOSUB 20000
105  HTAB 10: PRINT "(PREMERE UN TASTO)";: GET A$
110  HOME : PRINT : PRINT "QUESTA VERSIONE DEL BASIC APPLESOFT":
     PRINT "RAPPRESENTA I NUMERI REALI": PRINT : PRINT "CON UNA
     MANTISSA DI ";CI;" CIFRE IN BASE ";BA: PRINT "ED UN ESPONE
    NTE  "EI"<=EXP<"EA
120  PRINT : PRINT "L'EPSILON MACCHINA E'": PRINT : PRINT "EPS=
    ";EP$;" = ";EP: PRINT "EPN= ";EN$;" = ";EN:
200  PRINT : PRINT "I LIMITI DI UNDERFLOW/OVERFLOW SONO": PRINT
    : PRINT "XMIN=";XI$; SPC(  LEN (XA$) -  LEN (XI$))" = ";XI:
     PRINT "XMAX=";XA$;" = ";XA: PRINT "
210  PRINT "SONO RAPPRESENTATI ESATTAMENTE TUTTI": PRINT "GLI IN
    TERI COMPRESI FRA 1 E IMAX": PRINT : PRINT "IMAX= "BA"^"CI"
     = "IMAX
230  VTAB 24: PRINT "(PREMERE UN TASTO)";: GET A$
240  HOME : VTAB 2: IF BR THEN  PRINT "LE OPERAZIONI ARITMETICHE
     SONO": PRINT "EFFETTUATE CON 1 CIFRA EXTRA": PRINT "CHE VI
    ENE ELIMINATA ARROTONDANDO": PRINT "IL RISULTATO DURANTE LA
     MEMORIZZAZIONE":
250  VTAB 12: IF FW THEN  PRINT "L'OVERFLOW E' RAGGIUNGIBILE SOL
    O CON": PRINT "SOMME E SOTTRAZIONI": VTAB 24: PRINT "(PREME
    RE UN TASTO)";: GET A$
270  HOME : VTAB 12: PRINT "ORA VERRANNO ESEGUITI "NK" TEST DI V
    ARIO": PRINT "TIPO SULLE FUNZIONI ELEMENTARI": PRINT : INPUT
    "NUMERO PUNTI PER TEST? (p.es.100) N=";NP: HOME
300  FOR K = 1 TO NK
325  ON K GOSUB 40010,40020,40030,40040,40050,40060,40070,40080,
    40090,40100,40110,40120,40130,40140,40150,40160,40170,40180
    ,40190,40200,40210,40220,40230
330  HOME : INVERSE : PRINT  SPC( 38);: HTAB 1: PRINT "TEST "K"
    ..FUNZIONE ";F$
335  NORMAL : PRINT Y1$" contro "Y2$
340  PRINT : PRINT "CONTROLLO DI "NP" VALORI CASUALI"
350  PRINT "COMPRESI NELL'INTERVALLO": PRINT "[ "X1$" , "X2$" ]"

370  GOSUB 30000
380  PRINT "----------------------------------------"
390  PRINT F$" E'STATA MAGGIORE "C(0): PRINT  SPC(  LEN (F$) + 9
    )"UGUALE   "C(1): PRINT  SPC(  LEN (F$) + 9)"MINORE   "C(2)
    " VOLTE"
400  PRINT "----------------------------------------": PRINT "L'E
    RRORE RELATIVO MASSIMO E'": PRINT ER" = "BA"^" LEFT$ ( STR$
    (LR),7): IF ER > 0 THEN  PRINT "SI E' AVUTO PER X=";EX
410  PRINT "LA PERDITA DI PRECISIONE STIMATA E'": INVERSE :R2$(K
    ) =  LEFT$ ( STR$ ((CI + LR) * (CI + LR > 0.01)),4): PRINT
    R2$(K);: NORMAL : PRINT " CIFRE SIGNIFICATIVE IN BASE "BA
450  PRINT "----------------------------------------": PRINT "L'E
    RRORE RELATIVO QUADRATICO MEDIO E'": PRINT EM" = "BA"^" LEFT$
    ( STR$ (LM),7)
460  PRINT "LA PERDITA DI PRECISIONE MEDIA E'": INVERSE :R3$(K) =
     LEFT$ ( STR$ ((CI + LM) * (CI + LM > 0.01)),4): PRINT R3$(
    K);: NORMAL : PRINT " CIFRE SIGNIFICATIVE IN BASE "BA
500  VTAB 24: PRINT "(PREMERE UN TASTO)";: GET A$: NEXT K
600  HOME : INVERSE : PRINT "TEST    PERDITA MAX   PERDITA MEDIA
    ": NORMAL
650  FOR K = 1 TO NK: VTAB K + 1: HTAB 1: PRINT R1$(K);: HTAB 15
    : PRINT R2$(K);: HTAB 28: PRINT R3$(K);
660  NEXT
670  END ::
10000  REM  Determina il tipo di rappresentazione (binaria o BCD
    ) e la lunghezza della mantissa in cifre
10001  REM  EP,EN=epsilons; BA=base di rappresentazione, CI=lung
    h.mantissa; IM=massimo intero
10010 EP = 1:R = 1 / 9
10020  IF 1 + EP = 1 THEN 10100
10030 Z = EP:EP = EP * R: GOTO 10020
10100 EP = Z:R = (R + 1) / 2: IF R <  > 1 THEN 10030
10120 BR = 0: IF EP + EP + 1 = EP + 1 THEN EP = EP * 2:BR = 1
10130 EN = EP / 2
10200  FOR BA = 2 TO 10:R =  LOG (EP) /  LOG (BA): IF  ABS (( INT
    (R + .5) - R) / R) < 100 * EP THEN 10300
10210  NEXT
10220 BA = 0:R = 0: RETURN
10300 CI = 1 - R:EP$ =  STR$ (BA) + "^" +  STR$ (1 - CI):EN$ =  STR$
    (BA) + "^" +  STR$ ( - CI):IMAX = 1 / EN: RETURN :::
```

```
20000  REM  Determina il range dell'esponente mediante ricerca d
    ei limiti di underflow e overflow
20010 Z = 1 / BA / BA / BA:EI =  - 3
20015 R = BA:DR = 1
20020 XI = Z:Z = Z / R: IF Z > 0 THEN EI = EI - DR:R = R * R:DR =
    DR * 2: GOTO 20020
20030  IF DR > 1 THEN Z = XI: GOTO 20015
20040 XI$ =  STR$ (BA) + "^" +  STR$ (EI)
20050  REM  XI=limite underflow=BA^UE:
20200 Z = 1 - EN:EA = 0: ONERR  GOTO 20350
20210 R = BA:DR = 1
20320 XA = Z:Z = Z * R:EA = EA + DR:R = R * R:DR = DR * 2: GOTO
    20320
20350  CALL 768: IF DR > 1 THEN 20210
20400  ONERR  GOTO 20500
20410 Z = XA:Z = Z + Z: IF BA = 10 THEN Z = Z + Z + Z + Z + Z
20420 FW = 1:XA = Z:EA = EA + 1: GOTO 20600
20500  CALL 768:FW = 0
20600  POKE 216,0:XA$ =  STR$ (BA) + "^" +  STR$ (EA) + "*(1-" +
    EN$ + ")": RETURN : REM  XA=limite overflow=BA^EA::::
30000 C(0) = 0:C(1) = 0:C(2) = 0:S2 = 0:ER = 0
30020 D = X2 - X1
30050  FOR II = 1 TO NP
30100 X =  RND (II) * D + X1
30150  ON K GOSUB 40011,40021,40031,40041,40051,40061,40071,4008
    1,40091,40101,40111,40121,40131,40141,40151,40161,40171,401
    81,40191,40201,40211,40221,40231
30200 R1$(K) = F$
30300 IJ =  SGN (Y1 - Y2) + 1:C(IJ) = C(IJ) + 1
30310  IF Y1 = Y2 THEN 31000
30320 R = (Y1 - Y2) / Y2: IF  ABS (R) > ER THEN ER =  ABS (R):EX
    = X
30330 S2 = S2 + R * R
31000  NEXT
31100 EM =  SQR (S2 / NP)
31110 LR =  - 1E5: IF ER THEN LR =  LOG (ER) /  LOG (BA)
31120 LM =  - 1E5: IF EM THEN LM =  LOG (EM) /  LOG (BA)
31900  RETURN ::::::
32000  REM  PACCHETTO DI SUBROUTINE contenente le specifiche di
    ogni test: X1,X2=limiti in X; Y1,Y2 la funzione valutata in
    X in 2 modi
39999  DATA  22:
40010 X1 = 1:X2 = 2:X1$ = "1":X2$ = "2":F$ = "1/X":Y1$ = "1/(1/X
    )":Y2$ = "X": RETURN
40011 Y1 = 1 / (1 / X):Y2 = X: RETURN ::
40020 X1 =  SQR (.5):X2 =  SQR (2):X1$ = "SQR(2)/2":X2$ = "SQR(2
    )":F$ = "SQR(X)":Y1$ = "SQR(X*X)":Y2$ = "X": RETURN
40021 Y1 =  SQR (X * X):Y2 = X: RETURN ::
40030 X1 = 1 - 1 / 16:X2 = 1 + 1 / 16:X1$ = "1-1/16":X2$ = "1+1/
    16":F$ = "LOG(X)":Y1$ = F$:Y2$ = "SERIE DI TAYLOR di LOG(1+
    Y)": RETURN
40031 Y1 =  LOG (X):Y2 = 0:XX = X - 1:N =  INT ( LOG (EPS) /  LOG
    ( ABS (XX))) + 5: FOR I = N TO 1 STEP  - 1:Y2 = 1 / I - XX *
    Y2: NEXT :Y2 = Y2 * XX: RETURN ::
40040 X1 =  SQR (.5):X2 = 15 / 16:X1$ = "SQR(2)/2":X2$ = "1-1/16
    ":Y2$ = "LOG(17X/16)-LOG(17/16)": RETURN
40041 X = X + 32 - 32:Y1 =  LOG (X):Y2 =  LOG (X * 17 / 16) -  LOG
    (17 / 16): RETURN ::
40050 X1 =  - 1 / 16:X2 = 1 / 16:X1$ = "-1/16":X2$ = "1/16":F$ =
    "EXP(X)":Y1$ = F$:Y2$ = "SERIE DI TAYLOR di EXP(X)": RETURN

40051 Y1 =  EXP (X):Y2 = 0:N =  INT ( LOG (EPS) /  LOG ( ABS (X)
    )) + 5: FOR I = N TO 1 STEP  - 1:Y2 = Y2 * X / I + 1: NEXT
    : RETURN ::
40060 :X1 = 1 / 16 - L2 / 2:X2 = L2 / 2:Q =  EXP (1 / 16):X1$ =
    "1/16-LN(2)/2":X2$ = "LN(2)/2":Y1$ = "EXP(X-1/16)":Y2$ = "E
    XP(X)/EXP(1/16)": RETURN
40061 Y1 =  EXP (X - 1 / 16):Y2 =  EXP (X) / Q: RETURN ::
40070 X1 =  - L2 / 2:X2 =  INT (L2 * EI + 6):Q =  EXP (45 / 16):
    X1$ = "-LN(2)/2":X2$ =  STR$ (X2):Y1$ = "EXP(X-45/16)":Y2$ =
    "EXP(X)/EXP(45/16)": RETURN
40071 X = X - 45 / 16 + 45 / 16:Y1 =  EXP (X - 45 / 16):Y2 =  EXP
    (X) / Q: RETURN ::
40080 X1 = L2:X2 =  INT (L2 * EA - 1):X1$ = "LN(2)":X2$ =  STR$
    (X2): RETURN
40081 Y1 =  EXP (X - 45 / 16):Y2 =  EXP (X) / Q: RETURN ::
40090 X1 = .5:X2 = 1:X1$ = "1/2":X2$ = "1":F$ = "X^1":Y1$ = F$:Y
    2$ = "X": RETURN
40091 Y1 = X ^ 1:Y2 = X: RETURN ::
40100 X1 = .5:X2 = 1:X1$ = "1/2":X2$ = "1":F$ = "(X*X)^1.5":Y1$ =
    F$:Y2$ = "(X*X)*X": RETURN
40101 Y1 = (X * X) ^ 1.5:Y2 = X * X * X: RETURN ::
40110 X1 = 1:X2 = 2 ^  INT (EA / 3 - 2):X1$ = "1":X2$ = "2^" +  STR$
    ( INT (EA / 3 - 2)): RETURN
40111 Y1 = (X * X) ^ 1.5:Y2 = X * X * X: RETURN ::
40120 X1 =  - 1 / 16:X2 = 1 / 16:X1$ = "-1/16":X2$ = "1/16":F$ =
    "SIN(X)":Y1$ = F$:Y2$ = "SERIE DI TAYLOR di SIN(X)": RETURN

40121 Y1 =  SIN (X):Y2 = 0:XX = X * X:N =  INT ( LOG (EPS) /  LOG
    ( ABS (X))) + 5: FOR I = 2 *  INT (N / 2) + 1 TO 3 STEP  -
    2:Y2 = Y2 * XX / (I - I * I) + 1: NEXT :Y2 = Y2 * X: RETURN
    ::
```

(continua a pagina 244)

massimo di circa 9 cifre significative ha dovuto «buttar via» le ultime 6 cifre decimali (tutti 3) per lasciare il posto alle 6 cifre di 600000 «appena arrivate»; diventa quindi chiaro che quando si sottrae 600000 il risultato ha solo 9−6=3 cifre significative corrette.

L'esempio precedente ci dice che se effettuiamo sottrazioni di quantità all'incirca uguali (600000+1/3 e 600000) e non esattamente esprimibili in binario (600000.3333) si ha una forte perdita di precisione del risultato.

Questo è un classico esempio di «propagazione dell'errore»; l'errore viene commesso nel calcolo della prima somma e viene amplificato enormemente dalla successiva sottrazione.

Se ora ripetiamo il calcolo dell'esempio sostituendo ad 1/3 un numero che sia esattamente esprimibile in binario, per esempio 1/2048, otterremo un risultato esatto (cioè 1/2048) in barba al fatto che il calcolo coinvolga in base 10 ben più di 9 decimali:

1/2048 = 0.00048828125 e
600000 + 1/2048 − 600000 = 0.00048828125

Questo conferma che la propagazione degli errori ha effetto nullo se gli operandi sono sempre numeri esattamente esprimibili in binario.

Rimane un dubbio: perché il risultato del primo esempio non è stato esattamente 0.333?

Questo è dovuto ancora una volta al fatto che l'Apple effettua i calcoli in rappresentazione binaria; infatti se si ripete questo test su un calcolatore che effettua i conti in BCD con 9 cifre significative (p.es. una calcolatrice) si avrà il risultato 0.333 che è sicuramente più presentabile ma, attenzione, non più preciso.

(Chi vuole può verificare che il risultato mostrato dall'Apple è un numero esattamente rappresentabile in base 2 e vale esattamente 1365/4096).

Errori di questo tipo compaiono spesso nel calcolo di sommatorie o sviluppi in serie e bisogna fare attenzione; il loro effetto può essere ridotto con una riorganizzazione dell'ordine di calcolo, con artifizi matematici o al limite aumentando il numero di cifre significative coinvolte nei calcoli.

Errori di Overflow

Questo tipo di errore è dovuto al fatto che, così come esistono limiti per la lunghezza della mantissa, così esistono limiti alla lunghezza dell'esponente binario di un numero.

Nel caso dell'Applesoft e di altri linguaggi l'esponente occupa solo 1 byte, il che permette un range da 2^(−127) a 2^(+127); ciò corrisponde in decimale ad una variazione da circa 10^(−38) a 10^(+38), intervallo un poco ristretto che per certe applicazioni è del tutto insufficiente e provoca degli Overflow Error indesiderati o, più insidiosamente degli Underflow Error che sull'Apple non vengono segnalati.

La maggior parte dei linguaggi usati per calcoli scientifici permette una variazione

come minimo fra −99 e +99, ma limiti come −300,300 sono già accettabili.

Bug

Questi errori sono gli errori... dei progettisti; in teoria questi errori non dovrebbero esistere, ma, come si dice, errare humanum est... Si possono dividere in due classi:

1) errori di programmazione
2) imprecisioni di calcolo delle funzioni speciali.

I primi sono dovuti a vere e proprie sviste dei progettisti e possono comparire in qualunque routine matematica.

I secondi sono dovuti al fatto che, differentemente dal caso delle quattro operazioni, i cui algoritmi di calcolo sono invariabili e ben definiti, le altre funzioni possono essere calcolate con algoritmi diversi a seconda che si privilegi la precisione, la velocità di calcolo o l'occupazione di memoria richiesta. Talvolta scelte un po' infelici dei progettisti provocano infauste conseguenze sulla precisione dei risultati ottenuti.

Nel caso dell'Applesoft si è data molta importanza (forse troppa) all'occupazione di memoria sacrificando la precisione cosicché i risultati di alcune funzioni matematiche sono affetti da errori molto superiori a quel «mezzo bit» minimo teorico di cui si è parlato.

Vediamo i principali problemi che affliggono la parte «matematica» dell'Applesoft.

I bug matematici dell'Applesoft

Moltiplicazione

Ai progettisti dell'Applesoft è scappato un vero e proprio bug nella routine di moltiplicazione; vediamo un esempio:

PRINT 67108871*1 dà giustamente 67108871

mentre

PRINT 1*67108871 dà 67108867.5 (!!!!)

Questo bug è presente in tutte le moltiplicazioni in cui il secondo operando abbia nulli il 2' e 3' byte della mantissa (è il caso di 67108871) e comporta un errore che coinvolge le ultime 2 cifre del risultato.

Purtroppo questo errore, comparendo in una routine così fondamentale come la moltiplicazione, provoca una catena di errori più o meno gravi in tutte le routine di calcolo delle funzioni trascendenti (p. es. COS (1E-7) dà 0.999999968 invece di 1).

Trattandosi di un errore non troppo conosciuto ma molto grave spendiamo qualche parola per capire come è nato.

Il problema nasce dal fatto che la routine di moltiplicazione chiama, al fine di risparmiare qualche byte di RAM, una subroutine usata dalla funzione INT e (purtroppo) nella chiamata i progettisti si sono dimenticati di settare il flag di carry del 6502 come richiesto dalla subroutine con risultati a dir poco tragici. (In Assembler: manca un'istruzione SEC all'indirizzo $E9B2 prima dell'istruzione JMP SE8DA).

Confronti, INT

Queste due routine contengono alcuni bug piuttosto curiosi; per esempio:

PRINT (SQR(327)=SQR(327)) dà 0 (!)

il che significherebbe che SQR (327) è diverso da se stesso (!), ma ovviamente:

```
A=SQR(327)
PRINT (A=A) dà 1
```

Mentre per quanto riguarda la parte intera se:

```
A=1/7
        B=7
        C=A*B
        PRINT INT (A*B) dà 0
        PRINT INT (C) dà 1
```

Tutti questi piccoli «errorini» sono dovuti al fatto che l'Applesoft usa per i calcoli interni due registri (chiamati ARG e FAC) e inserisce nel registro FAC il risultato di ogni calcolo elementare; questo registro che può contenere risultati con ben 40 bit di mantissa (8 in più del normale); il bello è che i progettisti non avevano le idee molto chiare riguardo all'utilizzo di questi 8 bit «in più», cosicché questi bit vengono trascurati da alcune routine e invece utilizzati da altre routine con le infauste conseguenze di cui sopra.

Radice quadrata

La routine di calcolo della radice quadrata non ha veri e propri bug dal momento che... non esiste!

Infatti la funzione SQR viene calcolata mediante elevamento alla potenza 0.5, il che richiede il calcolo di un logaritmo e di un esponenziale.

Ciò provoca un rallentamento dei calcoli e notevoli imprecisioni, particolarmente evidenti nel calcolo della radice quadrata di quadrati perfetti, che dà il risultato esatto in media solo nel 2% dei casi.

SIN, COS, TAN, LOG

Non considerando i bug dovuti ai problemi dovuti alla moltiplicazione, l'unico problema compare nel calcolo di funzioni con argomenti vicini a 0 per le trigonometriche e vicini a 1 per il logaritmo.

Per esempio fate girare il programmino:

```
10 FOR I=1 TO 32
20 X=2^(-I)
30 PRINT I, SIN(X)/X,LOG(1+X)/X
40 NEXT
```

e osservate i risultati tenendo conto che i limiti di SIN(X)/X e LOG(1+X)/X per X tendente a 0 dovrebbero valere 1.

(segue da pagina 243)

```
40130 X1 = 0:X2 = PI / 2:X1$ = "0":X2$ = "PI/2":Y2$ = "3*SIN(X/3
      )-4*SIN(X/3)^3": RETURN
40131 Y1 =  SIN (X):Y2 =  SIN (X / 3):Y2 = (3 - Y2 * Y2 * 4) * Y
      2: RETURN ::
40140 X1 = PI * 6:X2 = PI * 6.5:X1$ = "6*PI":X2$ = "6.5*PI":Y2$ =
      "3*SIN(X/3)-4*SIN(X/3)^3)": RETURN
40141 X = (X / 3 + 64 - 64) * 3:Y1 =  SIN (X):Y2 =  SIN (X / 3):
      Y2 = (3 - Y2 * Y2 * 4) * Y2: RETURN ::
40150 X1 = PI * 7:X2 = PI * 7.5:X1$ = "7*PI":X2$ = "7.5*PI":F$ =
      "COS(X)":Y1$ = F$:Y2$ = "4*COS(X/3)^3-3*COS(X/3)": RETURN
40151 X = (X / 3 + 64 - 64) * 3:Y1 =  COS (X):Y2 =  COS (X / 3):
      Y2 = (Y2 * Y2 * 4 - 3) * Y2: RETURN ::
40160 X1 = 0:X2 = PI / 4:X1$ = "0":X2$ = "PI/4":F$ = "TAN(X)":Y1
      $ = F$:Y2$ = "2*TAN(X/2)/(1-TAN(X/2)^2)": RETURN
40161 Y1 =  TAN (X):Y2 =  TAN (X / 2):Y2 = Y2 * 2 / (1 - Y2 * Y2
      ): RETURN ::
40170 X1 = PI * 7 / 8:X2 = PI * 9 / 8:X1$ = "7/8*PI":X2$ = "9/8*
      PI": RETURN
40171 Y1 =  TAN (X):Y2 =  TAN (X / 2):Y2 = Y2 * 2 / (1 - Y2 * Y2
      ): RETURN ::
40180 X1 = PI * 6:X2 = PI * 6.25:X1$ = "6*PI":X2$ = "6.25*PI": RETURN

40181 Y1 =  TAN (X):Y2 =  TAN (X / 2):Y2 = Y2 * 2 / (1 - Y2 * Y2
      ): RETURN ::
40190 X1 =  - 1 / 16:X2 = 1 / 16:X1$ = "-1/16":X2$ = "1/16":F$ =
      "ATN(X)":Y1$ = F$:Y2$ = "SERIE DI TAYLOR di ATN(X)": RETURN

40191 Y1 =  ATN (X):Y2 = 0:XX = X * X:N =  INT ( LOG (EPS) /  LOG
      ( ABS (X))) + 5: FOR I =  INT (N / 2) * 2 + 1 TO 1 STEP  -
      2:Y2 = 1 / I - XX * Y2: NEXT :Y2 = Y2 * X: RETURN ::
40200 X1 = 1 / 16:X2 = 2 -  SQR (3):X1$ = "1/16":X2$ = "2-SQR(3)
      ":Y2$ = "ATN(1/16)+ATN((X-1/16)/(1+X/16))": RETURN
40201 Y1 =  ATN (X):Y2 =  ATN ((X - 1 / 16) / (1 + X / 16)) +  ATN
      (1 / 16): RETURN ::
40210 X1 = 2 -  SQR (3):X2 =  SQR (2) - 1:X1$ = "2-SQR(3)":X2$ =
      "SQR(2)-1":Y1$ = "2*ATN(X)":Y2$ = "ATN(2*X/(1-X*X))": RETURN

40211 Y1 =  ATN (X) * 2:Y2 =  ATN (X * 2 / (1 - X * X)): RETURN
      ::
40220 X1 =  SQR (2) - 1:X2 = 1:X1$ = "SQR(2)-1":X2$ = "1": RETURN

40221 Y1 =  ATN (X) * 2:Y2 =  ATN (X * 2 / (1 - X * X)): RETURN
      ::
40230  RETURN
40231  RETURN
```

Generatore di numeri casuali

Questa routine ha il difetto di utilizzare per la generazione un algoritmo «fatto in casa» che talvolta dà risultati non troppo casuali: infatti entra facilmente in «loop» generando sempre gli stessi numeri «casuali» presi da una sequenza contenente al più qualche centinaio di valori diversi.

Per averne la prova basta far girare questo programmino:

```
10 X=RND(-0.25)
20 FOR I=1 TO 30000:X=RND(1):NEXT
30 FOR I=1 TO 2000
40 IF RND(1)=X THEN PRINT I,X
50 NEXT
```

Dopo qualche minuto verranno stampate le cifre 202, 404, 606, 808,... il che sta a significare che il generatore di numeri «casuali» è entrato in loop e sta generando numeri presi da una sequenza fissa di 202 valori.

Il bello è che il generatore entra in loop improvvisamente, impedendo il regolare funzionamento di un programma che solo pochi minuti prima funzionava regolarmente, e mandando alla disperazione il povero programmatore che nel tentativo di trovarne la causa pensa a tutto fuorché ad un «ammattimento» del generatore di numeri casuali.

Vorrei tanto che i progettisti di questo algoritmo, prima di scrivere altre routine del genere, leggessero una frase di Donald E. Knuth, autore della famosa «bibbia», il quale anni fa costrui un generatore di numeri «supercasuali» che doveva avere caratteristiche eccelse e che invece si dimostrò avere lo stesso problema del generatore dell'Apple:

«I numeri casuali non vanno generati con un metodo scelto a caso. Ci si deve servire di una qualche teoria».

EXP, ATN

Per fortuna almeno queste routine sono esenti da errori; non saranno molto veloci, ma almeno sono precise.

Con questo si conclude questo elenco (parziale) degli innumerevoli bug dell'Applesoft.

Purtroppo tutte queste manchevolezze non sono mai state corrette nel corso degli anni e si trovano immutate in tutte le copie dell'Applesoft contenuta nei IIE, IIC, e pure IIGS.

(Forse per ragioni di compatibilità totale, errori compresi??).

Per la verità l'Applesoft è stato leggermente modificato nel corso degli anni per adattarlo ai vari modelli Apple, ma la parte matematica (purtroppo) non è mai stata revisionata.

I programmi di test

Ho allegato al programma Mulprec due programmi dimostrativi.

Nel primo, «Bugs Applesoft», vengono mostrate alcuni dei bug matematici del Basic Applesoft descritti in precedenza; se si esegue questo programma dopo aver caricato Mulprec (od anche il solo piccolo AP2) si noterà come tutti i bug mostrati spariscano.

Nel secondo, «Test Applesoft», viene effettuato un test sulla precisione della matematica Applesoft.

In questo programma, liberamente ispirato ad un programma di controllo di precisione per calcolatori mainframe, vengono determinate quantità importanti come:

— La base di rappresentazione (2 o 10);
— gli EPSILON MACCHINA

EPN (il più piccolo numero tale che 1+EPS−1=EPS)

EPN (il più piccolo numero tale che EPN −1+1=EPN)

— La lunghezza della mantissa in bit.
— I limiti di Underflow e Overflow XMIN e XMAX

Vengono poi eseguiti alcuni test sulla precisione di calcolo delle funzioni elementari: vengono generati un certo numero di valori casuali (p. es. 1000) all'interno di un intervallo dipendente dal test scelto; per ogni valore viene calcolata la funzione sotto esame in due modi differenti, rispettivamente mediante la corrispondente routine del Basic e mediante sviluppi in serie o relazioni teoriche; i due valori ottenuti vengono confrontati, dopo di che si prosegue con la generazione di un nuovo valore di X.

```
N  FUNZ   ESPR1      contro  ESPR2
1  1/X    1/(1/X)     ! X
2  SQR    SQR (X*X)   ! X

3  LOG    LOG (X)     ! Sviluppo in serie
4  LOG    LOG (X)     ! LOG(A*X)-LOG(A)
5  EXP    EXP(X)      ! Sviluppo in serie
6  EXP    EXP(X-A)    ! EXP(X)/EXP(A)
7  EXP    EXP(X-B)    ! EXP(X)/EXP(B)
8  EXP    EXP(X-B)    ! EXP(X)/EXP(B)

9  X^Y    X^1         ! X
10 X^Y    (X*X)^1.5   ! X*X*X
11 X^Y    (X*X)^1.5   ! X*X*X

12 SIN    SIN(X)      ! Sviluppo in serie
13 SIN    SIN(X)      ! Sviluppo SIN(X/3)
14 SIN    SIN(X)      ! Sviluppo SIN(X/3)
15 COS    COS(X)      ! Sviluppo COS(X/3)

16 TAN    TAN(X)      ! Sviluppo TAN(X/2)
17 TAN    TAN(X)      ! Sviluppo TAN(X/2)
18 TAN    TAN(X)      ! Sviluppo TAN(X/2)

19 ATN    ATN(X)      ! Sviluppo in serie
20 ATN    ATN(X)      ! Sviluppo ATN(X-A)
21 ATN    ATN(X)      ! Sviluppo ATN(X-A)
22 ATN    ATN(X)      ! Sviluppo ATN(X-A)
```

Risultati dei test

TST	APPLE	AP2	MULPREC	CDC7600
1	0.5 0	0.5 0	0.5 0	----
2	1.5 0.2	1.0 0	0.5 0	0.5 0
3	17 11	1.9 0	1.6 0.1	1.5 0
4	4.2 2.7	1.7 0.1	1.3 0	----
5	1.0 0	1.0 0	1.0 0	----
6	4.3 0	1.1 0	1.5 0	2.0 0.3
7	3.9 1.6	3.8 1.5	1.7 0.3	1.8 0.2
8	3.8 1.6	3.8 1.6	2.1 0.3	2.0 0.2
9	6.3 1.1	0 0	0 0	1.0 0
10	2.6 1.3	1.3 0	1.0 0	2.0 0.2
11	4.2 3.3	3.9 3.0	1.0 0	9.9 8.6
12	8.8 3.5	1.0 0	1.0 0	----
13	4.7 0.3	1.5 0	1.5 0	2.0 0.6
14	14 9.1	1.0 0	1.0 0	2.4 0.6
15	16 11	14 9.0	1.0 0	2.5 0.7
16	4.7 0.5	2.1 0.2	1.8 0.2	2.6 0.9
17	1.9 0.2	2.1 0.2	1.7 0.1	2.4 0.5
18	2.0 0.2	2.2 0.2	1.9 0.2	2.6 0.9
19	1.0 0	1.0 0	1.0 0	1.0 0
20	3.1 0	1.4 0	2.0 0	2.0 0.4
21	0.9 0	0.9 0	0.9 0	2.1 0.6
22	1.3 0.1	1.3 0.1	1.0 0	2.0 0.1

Descrizione dei test

Al termine di ogni test vengono visualizzati:

il numero di volte in cui il valore calcolato dalla funzione è stato rispettivamente superiore, uguale od inferiore al valore ricavato nell'altro modo; l'errore relativo quadratico medio e l'errore relativo massimo commessi nel calcolo della funzione, errori espressi sia in valore assoluto, sia come numero di bit di mantissa errati; quest'ultimo dato è il più corretto da usare per confrontare le prestazioni di programmi o linguaggi diversi.

Per evitare di far girare il programma, ho creato una tabella, pubblicata in questa pagina con i risultati finali ottenuti rispettivamente: con l'Applesoft standard, con il programma AP2 (descritto più avanti), con il programma Mulprec settato a 9 cifre di precisione, e per confronto i risultati forniti dal FORTRAN CDC7600 (a 14 cifre).

Per ogni test sono mostrati l'errore relativo e l'errore relativo quadratico medio espressi in bit.

Il primo valore rappresenta la perdita di precisione del caso peggiore, il secondo il valore medio dell'errore dei 1000 punti controllati; se è 0 allora l'errore relativo medio è così piccolo da non poter influenzare alcun bit del risultato (in media).

Per valutare la grandezza dei valori riportati è bene sapere che ad ogni 3 bit di mantissa binaria errati corrisponde approssimativamente una cifra decimale errata.

Quindi nel test numero 3 (LOG di valori vicini ad 1) per l'Applesoft standard risulta un errore medio di 11 bit, che corrisponde ad un errore di più di 3 cifre decimali, ed un errore massimo di 17 bit che corrisponde a ben 5 cifre errate.

Faccio notare che questi test danno solo una indicazione sulle prestazioni complessive delle routine delle funzioni matematiche. Comunque vengono messi in luce tutti i bug di cui si è parlato prima (LOG, SIN, SQR) e si nota immediatamente un notevole aumento di precisione passando dall'Applesoft standard all'Applesoft «potenziato» col programma Mulprec.

Il programma di test può naturalmente essere utilizzato per provare la precisione delle routine matematiche di altri linguaggi per Apple o per altri computer.

Riguardo all'Apple ho provato il programma con il Pascal UCSD (in singola precisione) con risultati così deludenti che non meritano neppure di essere riportati; basta calcolare SQRT (25)−5 per capire che aria tira...

E sua maestà IBM (e compatibili)?

Per curiosità su di un M24 ho provato il GWBASIC 3.1 con risultati abbastanza buoni, tranne che con il test numero 3: 12 bit persi in media con una punta massima di 17, per di più su di una mantissa di soli 24 bit! (peggio dell'Applesoft...).

Ebbene sì, ancora una volta il famigerato bug sul logaritmo di valori vicini ad 1.

Per fortuna il bug sparisce completamente se si eseguono i calcoli in doppia precisione (con lo switch/D) o se si compila il programma, comunque credo proprio che i progettisti meritino una bella tirata d'orecchi... MC