

Programma in FORTRAN per il calcolo rapido dei Numeri Grandi di Fibonacci mediante gli algoritmi di Cristiano Teodoro

Giuseppe Matarazzo

Ottobre 2006

Sommario

Questo lavoro nasce dalla necessità di rendere disponibili i raffinati algoritmi di calcolo dei numeri di Fibonacci con migliaia di cifre pubblicati dall'ing. Cristiano Teodoro e reperibili nel seguente documento: <http://astrodinamica.altervista.org/PDF/fibo.pdf>.

Il sorgente QuickBasic, che egli mi ha gentilmente trasmesso in una comunicazione privata, non è purtroppo utilizzabile in forma universale in quanto, per ragioni connesse all'uso del dimensionamento dei vettori, non se ne può ricavare un eseguibile .EXE. Necessita allora ricorrere agli interpreti QB45 o QB71, che per loro natura sono più lenti di un programma compilato, non sono affatto trasportabili da un sistema operativo a un altro ed infine sono dei 'software proprietari' di difficile reperibilità.

Il sorgente del programma, scritto in Fortran, sarà esposto in un capitolo mentre gli eseguibili, compilati con GNU-G77 e leggibili sia in ambiente Windows che Linux, sono disponibili nel link telematico citato più avanti.

Introduzione

La serie dei **Numeri di Fibonacci**, che gode di una ben nota *proprietà armonica*, si ottiene sommando i due numeri interi precedenti quello esaminato. La procedura di calcolo è di tipo *ricorsivo* a partire dai primi due elementi $F(0)=0$ e $F(1)=1$; da cui si ottiene $F(2)=1$, $F(3)=2$ e così via.

Nel calcolo computerizzato, si manifestano subito due **grosse difficoltà**: la prima riguarda il modo come calcolare il valore *esatto* dell'*i* - *esimo* numero dopo che si superano le *nove* cifre; sappiamo bene come in questo caso il computer tronca il valore e lo scrive in notazione esponenziale. Il secondo *intoppo* invece è strettamente legato alla natura iterativa del procedimento, per cui, se si vuole calcolare il valore $F(n)$ con $n = 100\,000$ per esempio, bisogna determinare tutti i precedenti $(n - 1)$ elementi della serie di Fibonacci, un metodo **bruto** che, come vedremo, presenta degli inconvenienti pratici che 'non' sono legati al tempo di calcolo del computer.

L'autore del metodo ha *by-passato* il primo problema usando delle routine a precisione multipla e le ha scritte in linguaggio Qbasic. Qui sono state convertite in Fortran. Riguardo al secondo inconveniente egli ha sfruttato alcune proprietà dei numeri di Fibonacci, legate ad un'altra serie (meno nota della precedente), quella di **Lucas**, che è "quasi" identica alla prima in quanto cambia solo il primo termine $L(0)=2$. Il secondo è uguale $L(2)=F(2)=1$, e i successivi

diventano, sommando i due precedenti, $L(2)=3$, $L(3)=4$, $L(4)=7$, etc. Il calcolo di un termine F_i della serie di Fibonacci si effettua tramite i due precedenti F_{i-1} , L_{i-1} ma ricorrendo ad un numero *limitatissimo* di iterazioni, che sono solo 17 per il calcolo di $F(200,000)$, vale a dire un numero composto di 41798 cifre e calcolato in circa 4 secondi dal programma in Fortran che vedremo più avanti.

Come detto in sommario, la descrizione del procedimento è disponibile in questo file PDF: <http://astrodinamica.altervista.org/PDF/fibo.pdf>.

Metodo tradizionale di calcolo in Assembler

Pensare di calcolare (e trascrivere su file) tutta la serie di Fibonacci fino ad arrivare ad un *numero grande* con migliaia di cifre è un'*impresa* impegnativa. Essa è strettamente legata al tempo di esecuzione del computer, che dipende sia dal processore del PC che (soprattutto) dalla bontà del linguaggio di programmazione utilizzato. I moderni personal computer, equipaggiati con Intel-Pentium4 (o simili) a 2.4GHz (o oltre), consentono una rapidità esecutiva notevole, per cui tutto il "peso" del calcolo va affidato al software.

Come dicevamo prima, nel metodo classico va scandagliata tutta la serie di Fibonacci, da $F(0)$ a $F(n)$. Niente di meglio allora che farlo con un programmino in linguaggio macchina, che viene tradotto in file .COM dal compilatore NASM ed è composto di soli 211 bytes. Il sorgente è il seguente e l'ho prelevato dal web.

```
; A small program that calculates and prints terms of the Fibonacci series
;
; fibo.asm
; assemble using nasm:
; nasm -o fibo.com -f bin fibo.asm
;
;*****
; Alterable Constant
;*****
; You can adjust this upward but the upper limit is around 150000 terms.
; the limitation is due to the fact that we can only address 64K of memory
; in a DOS com file, and the program is about 211 bytes long and the
; address space starts at 100h. So that leaves roughly 65000 bytes to
; be shared by the two terms (num1 and num2 at the end of this file). Since
; they're of equal size, that's about 32500 bytes each, and the 150000th
; term of the Fibonacci sequence is 31349 digits long.
;
maxTerms equ 150000 ; number of terms of the series to calculate

;*****
; Number digits to use. This is based on a little bit of tricky math.
; One way to calculate F(n) (i.e. the nth term of the Fibonacci seeries)
; is to use the equation  $\text{int}(\phi^n/\text{sqrt}(5))$  where ^ means exponentiation
; and  $\phi = (1 + \text{sqrt}(5))/2$ , the "golden number" which is a constant about
; equal to 1.618. To get the number of decimal digits, we just take the
; base ten log of this number. We can very easily see how to get the
; base phi log of F(n) -- it's just  $n*\text{lp}(\phi)+\text{lp}(\text{sqrt}(5))$ , where lp means
```

```

; a base phi log. To get the base ten log of this we just divide by the
; base ten log of phi. If we work through all that math, we get:
;
; digits = terms * log(phi) + log(sqrt(5))/log(phi)
;
; the constants below are slightly high to assure that we always have
; enough room. As mentioned above the 150000th term has 31349 digits,
; but this formula gives 31351. Not too much waste there, but I'd be
; a little concerned about the stack!
;
        digits      equ (maxTerms*209+1673)/1000

; this is just the number of digits for the term counter
cntDigits  equ 6 ; number of digits for counter

        org        100h                ; this is a DOS com file
;*****
;*****
main:
; initializes the two numbers and the counter. Note that this assumes
; that the counter and num1 and num2 areas are contiguous!
;
mov  ax,'00'    ; initialize to all ASCII zeroes
mov  di,counter ; including the counter
mov  cx,digits+cntDigits/2 ; two bytes at a time
cld                ; initialize from low to high memory
rep  stosw       ; write the data
inc  ax         ; make sure ASCII zero is in al
mov  [num1 + digits - 1],al ; last digit is one
mov  [num2 + digits - 1],al ;
mov  [counter + cntDigits - 1],al

        jmp  .bottom    ; done with initialization, so begin

.top
; add num1 to num2
mov  di,num1+digits-1
mov  si,num2+digits-1
mov  cx,digits ;
call AddNumbers ; num2 += num1
mov  bp,num2 ;
call PrintLine ;
dec  dword [term] ; decrement loop counter
jz  .done ;

; add num2 to num1
mov  di,num2+digits-1
mov  si,num1+digits-1
mov  cx,digits ;
call AddNumbers ; num1 += num2

```

```

.bottom
    mov bp,num1    ;
    call PrintLine ;
    dec dword [term] ; decrement loop counter
    jnz .top      ;
.done
    call CRLF     ; finish off with CRLF
    mov ax,4c00h ; terminate
    int 21h      ;

;*****
;
; PrintLine
; prints a single line of output containing one term of the
; Fibonacci sequence. The first few lines look like this:
;
; Fibonacci(1): 1
; Fibonacci(2): 1
; Fibonacci(3): 2
; Fibonacci(4): 3
;
; INPUT:      ds:bp ==> number string, cx = max string length
; OUTPUT:     CF set on error, AX = error code if carry set
; DESTROYED:  ax, bx, cx, dx, di
;
;*****
PrintLine:
    mov dx,eol    ; print combined CRLF and msg1
    mov cx,msg1len+eolllen ;
    call PrintString ;

    mov di,counter ; print counter
    mov cx,cntDigits ;
    call PrintNumericString

    call IncrementCount ; also increment the counter

    mov dx,msg2    ; print msg2
    mov cx,msg2len ;
    call PrintString ;

    mov di,bp     ; recall address of number
    mov cx,digits ;
    ; deliberately fall through to PrintNumericString

;*****
;
; PrintNumericString
; prints the numeric string at DS:DI, suppressing leading zeroes
; max length is CX

```

```

;
; INPUT:      ds:di ==> number string, cx = max string length
; OUTPUT:     CF set on error, AX = error code if carry set
; DESTROYED: ax, bx, cx, dx, di
;
;*****
PrintNumericString:
; first scan for the first non-zero byte
mov  al,'0'    ; look for ASCII zero
cld          ; scan from MSD to LSD
repe scasb    ;
mov  dx,di    ; points to one byte after
dec  dx      ; back up one character
inc  cx      ;
; deliberately fall through to PrintString

;*****
;
; PrintString
; prints the string at DS:DX with length CX to stdout
;
; INPUT:      ds:dx ==> string, cx = string length
; OUTPUT:     CF set on error, AX = error code if carry set
; DESTROYED: ax, bx
;
;*****
PrintString:
mov  bx,1     ; write to stdout
mov  ah,040h  ; write to file handle
int  21h     ; ignore return value
ret          ;

;*****
;
; AddNumbers
; add number 2 at ds:si to number 1 at es:di of width cx
;
;
; INPUT:      es:di ==> number1, ds:si ==> number2, cx= max width
; OUTPUT:     CF set on overflow
; DESTROYED: ax, si, di
;
;*****
AddNumbers:
std          ; go from LSB to MSB
clc         ;
pushf       ; save carry flag
.top
mov  ax,0f0fh ; convert from ASCII BCD to BCD
and  al,[si]  ; get next digit of number2 in al

```

```

and ah,[di] ; get next digit of number1 in ah
popf ; recall carry flag
adc al,ah ; add these digits
aaa ; convert to BCD
pushf ;
add al,'0' ; convert back to ASCII BCD digit
stosb ; save it and increment both counters
dec si ;
loop .top ; keep going until we've got them all
popf ; recall carry flag
ret ;

;*****
;
; IncrementCount
; increments a multidigit term counter by one
;
; INPUT: none
; OUTPUT: CF set on overflow
; DESTROYED: ax, cx, di
;
;*****
IncrementCount:
mov cx,cntDigits ;
mov di,counter+cntDigits-1
std ; go from LSB to MSB
stc ; this is our increment
pushf ; save carry flag
.top
mov ax,000fh ; convert from ASCII BCD to BCD
and al,[di] ; get next digit of counter in al
popf ; recall carry flag
adc al,ah ; add these digits
aaa ; convert to BCD
pushf ;
add al,'0' ; convert back to ASCII BCD digit
stosb ; save and increment counter
loop .top ;
popf ; recall carry flag
ret ;

;*****
;
; CRLF
; prints carriage return, line feed pair to stdout
;
; INPUT: none
; OUTPUT: CF set on error, AX = error code if carry set
; DESTROYED: ax, bx, cx, dx
;

```

```

;*****
CRLF: mov dx,eol      ;
      mov cx,eollen   ;
      jmp PrintString ;

;*****
; static data
;*****
eol db 13,10      ; DOS-style end of line
eollen equ $ - eol

msg1 db 'Fibonacci(' ;
msg1len equ $ - msg1

msg2 db ')': '      ;
msg2len equ $ - msg2
;*****
; initialized data
;*****
term dd maxTerms      ;
;*****
; unallocated data
;
; A better way to do this would be to actually ask for a memory
; allocation and use that memory space, but this is a DOS COM file
; and so we are given the entire 64K of space. Technically, this
; could fail since we *might* be running on a machine which doesn't
; have 64K free. If you're running on such a memory poor machine,
; my advice would be to not run this program.
;
;*****
; static data
counter:      ;
num1 equ counter+cntDigits ;
num2 equ num1+digits      ;

```

La ragione per cui ho voluto riportare *per intero* questo listato in *Assembler*, ossia in linguaggio macchina, è per mostrare come, usando la *forza bruta* del computer, si riesce a calcolare tutti i numeri di Fibonacci dal primo fino al 150,000-esimo termine, vale a dire fino al valore introdotto nella variabile (maxTerms equ 150000). Naturalmente se si dispone di PC con una memoria superiore a 64k questo valore può essere incrementato, ma come vedremo tra poco non serve farlo data l'enormità di memoria necessaria per raccogliere in un file *tutt'e* 150,000 numeri.

Quindi, compilando questo listato con il comando `nasm -o fibo.com -f bin fibo.asm`, otteniamo l'eseguibile *fibo.com* di 211b. Nel nostro deposito dei programmi eseguibili (cfr. più in basso), abbiamo inserito oltre a questo file che produce un *output* esageratamente grande, uno più contenuto che arriva fino al calcolo dei primi 15,100 elementi della serie di Fibonacci, proprio quello esposto nella relazione di Cristiano Teodero. Il nome è 15p1.com.

Vediamo adesso un “frammento” di che cosa si ottiene dopo aver dato il seguente comando DOS: `15p1 > 15p1.txt`:

```
.....  
Fibonacci(220): 4244200115309993198876969489421897548446236915  
Fibonacci(221): 6867260041627791953052057353082063289320596021  
Fibonacci(222): 11111460156937785151929026842503960837766832936  
Fibonacci(223): 17978720198565577104981084195586024127087428957  
Fibonacci(224): 29090180355503362256910111038089984964854261893  
Fibonacci(225): 47068900554068939361891195233676009091941690850  
Fibonacci(226): 76159080909572301618801306271765994056795952743  
Fibonacci(227): 123227981463641240980692501505442003148737643593  
Fibonacci(228): 199387062373213542599493807777207997205533596336  
Fibonacci(229): 322615043836854783580186309282650000354271239929  
Fibonacci(230): 522002106210068326179680117059857997559804836265  
Fibonacci(231): 844617150046923109759866426342507997914076076194  
Fibonacci(232): 1366619256256991435939546543402365995473880912459  
Fibonacci(233): 2211236406303914545699412969744873993387956988653  
Fibonacci(234): 3577855662560905981638959513147239988861837901112  
Fibonacci(235): 5789092068864820527338372482892113982249794889765  
Fibonacci(236): 9366947731425726508977331996039353971111632790877  
Fibonacci(237): 15156039800290547036315704478931467953361427680642  
Fibonacci(238): 24522987531716273545293036474970821924473060471519  
Fibonacci(239): 39679027332006820581608740953902289877834488152161  
Fibonacci(240): 64202014863723094126901777428873111802307548623680  
Fibonacci(241): 103881042195729914708510518382775401680142036775841  
Fibonacci(242): 168083057059453008835412295811648513482449585399521  
Fibonacci(243): 271964099255182923543922814194423915162591622175362  
Fibonacci(244): 440047156314635932379335110006072428645041207574883  
Fibonacci(245): 712011255569818855923257924200496343807632829750245  
Fibonacci(246): 1152058411884454788302593034206568772452674037325128  
Fibonacci(247): 1864069667454273644225850958407065116260306867075373  
Fibonacci(248): 3016128079338728432528443992613633888712980904400501  
Fibonacci(249): 4880197746793002076754294951020699004973287771475874  
Fibonacci(250): 7896325826131730509282738943634332893686268675876375  
.....
```

Si capisce subito come l’inconveniente più *fastidioso* di questo file di output risieda nel fatto che ogni riga è praticamente larga quanto le cifre del numero di Fibonacci. Basta ricordare che $F(150,000)$ occupa 31 348 colonne per rendersi conto come sia “disagevole”, anche con dei potenti editor, effettuare dei *taglia e cuci* proprio per andare prelevare il numero finale.

Ricordiamo, infine, che attivando il comando (`fibonacci > 150000.txt`) il tempo di esecuzione del programma è di circa 6 minuti, mentre il file risultato occupa la *ragguardevole* memoria di circa 2.4 GigaByte; per cui, pensare di muoversi a eliminare l’intera zavorra per prelevare solo $F(150,000)$ è da ... stakanovisti della tastiera!

Il programma che invece esporremo nel prossimo capitolo sarà di ben altra agilità e il lettore avrà modo di stabilire quale dei due preferisce.

Il sorgente Fibvelo.for

Prima di mostrare il listato, diciamo che il *benemerito* 'Software libero' (e assolutamente gratuito) permette di disporre del potente compilatore G77 (licenza GNU) in tutti i sistemi operativi del mondo. Nel primo *item* del mio sito, denominato SOFA, c'è la possibilità di prelevare il compilatore G77 nei 3 ambienti: Windows, DOS e Linux. Ecco il link: <http://astrodinamica.altervista.org>.

Per quanto riguarda il **Big Loop**, che trascrive tutte le cifre del numero di Fibonacci impostato in 10 settuple (per una comodissima larghezza di 80 colonne), ho preferito non dare i commenti 'REM-ati' in quanto appartengono alla proprietà intellettuale dell'ing. Cristiano Teodoro.

```
* -----
*   PROGRAM FIBVELO.FOR      Serie Fibonacci in Fortran (09/19-Ott-2006)
*
*   Compilazione programma (Linux):  g77 fibvelo.for -o fibvelo
*   "           "           (DOS):    g77 fibvelo.for -o fibvelo.exe
*   -----
*
*   N.ro Fibonacci [trascritto su 80 colonne] F( 610)=
* 0000013 5823697 9127826 6616906 2844948 0673556 5776939 5021071 8307561 2628409
* 0342094 5290185 0178519 3631898 3433611 3240870 2477150 6039819 2490855 0000000
*
* ... eliminando manualmente gli zeri del primo gruppo e le settuple NULLE
* dell'ultima riga, otteniamo:
* (infatti non serve appesantire il sorgente Fortran con una selva di
*   IF...THEN...END IF per questa banale 'manovra' di taglio)
*
*   N.ro Fibonacci [trascritto su 80 colonne] F( 610)=
*      13 5823697 9127826 6616906 2844948 0673556 5776939 5021071 8307561 2628409
* 0342094 5290185 0178519 3631898 3433611 3240870 2477150 6039819 2490855
*
*       implicit double precision (a-h,o-z)
*       integer z, g
*- Con i parametri del dimensionamento vettoriale seguente si arriva a calcolare
*- (in 4-5 sec.) F(260,000), ovvero le 54337 cifre del Numero di Fibonacci
*-----
*       PARAMETER ( NX=6000, NY=3100, NZ=2000, NW=20 )
*-----
*       PARAMETER ( ZERO= 0.0 )
*       integer a(0:NW), xx(0:NW), ps
*       double precision UF(0:NX), VL(0:NX), P(0:NX)
*       double precision s(0:NY), Q(0:NZ), UF0(0:NX)
*       double precision f, F0, L, digits
*       double precision cf, cl, cf0, w, D, pr, r, x
*-----
*
*       print 1000
*       print 2007
```

```

read (*,3007,iostat=nerr) z
a(0)=z
x=z
xx(0)=x
ps=int(LOG10(real(z))/LOG10(2.0))+1    ! OK real(z)
digits= real(z)*LOG10((1.0+SQRT(5.0))/2.0)
write(*,*)
write(*,*) ' Numero iterazioni stimate=',ps

n=-1
write(*,*) '   n   a(k)   x(k)'
write(*,*) ' -----'
10  n=n+1
    if (x.eq.int(x/2)*2) then
        a(n)=0
    else
        a(n)=1
    end if
    xx(n)=x

write(*,6001) n, a(n), xx(n)

    if (x.eq.1) then
        write(*,*) ' -----'
        write(*,*) ' Numero iterazioni necessarie=', n
        go to 20
    end if

x=int(x/2)
go to 10

20  continue
write(*,*)
f=1
L=1

**
do k = n-1, 0, -1
    F0=f
    if (a(k).eq.0) then
        f=f*L
        L=L*L-2*(-1)**a(k+1)
    else
        f=L*(f+L)/2 -1*(-1)**a(k+1)
        L=L*(L+5*F0)/2 -1*(-1)**a(k+1)
    end if
    if (L.gt.10**7) goto 30
end do

30  continue
k0=k

```

```

g=7
pr=10**g
r=0
  UF(1) = int(f/pr)
  UF(0) = f - UF(1)*pr
  VL(1) = int(L/pr)
  VL(0) = L - VL(1)*pr
cf=1
  if (UF(1).eq.0) then cf=0
cl=1
  if (VL(1).eq.0) then cl=0

*** ----- Inizio LOOP -----
do kk = k0-1, 0, -1
  cf0=cf
  do k = cf0, 0, -1
    UFO(k)=UF(k)
  end do

  if (a(kk).eq.0) then
*-----
    do j=0, NX
      P(j) = ZERO           ! Erase P
    end do
*-----

    w = cf + cl + 1
    do k = 0, cl
      r=0
      do h = 0, cf
        x=P(h+k) + UF(h)*VL(k) + r
        r=int(x/pr)
        P(h+k) = x - r*pr
      end do
      P(h+k)=r
    end do
    if (P(w).eq.0) then w = w - 1
*-----

    do j=0, NX
      UF(j) = ZERO         ! Erase UF
    end do
*-----

    do h = w, 0, -1
      UF(h)=P(h)
    end do
    cf=w
    w=0
*-----

    do j=0, NX
      P(j) = ZERO         ! Erase P
    end do

```

```

*-----
w = cl + cl + 1
do k = 0, cl
  r=0
  do h = 0, cl
    x=P(h+k) + VL(h)*VL(k) + r
    r=int(x/pr)
    P(h+k) = x - r*pr
  end do
  P(h+k)=r
end do
if (P(w).eq.0) then w = w - 1
*-----
do j=0, NX
  VL(j) = ZERO
end do
! Erase VL
*-----
do h = w, 0, -1
  VL(h) = P(h)
end do
cl=w
w=0
*-----
do j=0, NX
  P(j) = ZERO
end do
! Erase P
*-----
w0=(-1)**a(kk+1)
if (w0.gt.0) then
  VL(0)= VL(0) - 2
else
  VL(0)= VL(0) + 2
end if

else
**----- 2^ parte a(kk)=1
*-----
do j=0, NY
  s(j) = ZERO
end do
! Erase s
*-----
cs=cl
r=0
do h = 0, cs
  s(h) = VL(h) + UF(h) + r
  r=0
  if (s(h).ge.pr) then
    s(h) = s(h) - pr
    r=1

```

```

        end if
    end do

    if (r.ge.1) then
        cs= cs+1
        s(cs)=r
    end if

*--
    w = cs + cl + 1
    do k = 0, cs
        r=0
        do h = 0, cl
            x=P(h+k) + s(k)*VL(h) + r
            r=int(x/pr)
            P(h+k) = x - r*pr
        end do
        P(h+k) = r
    end do
    if (P(w).eq.0) then w = w - 1

*--
        r=0
        do k = w, 0, -1
            Q(k)= INT((P(k)+r*pr)/2)
            r= MOD(P(k),2)
        end do
    if (Q(w).eq.0) then w = w - 1

        cf=w
        w=0
        do k = cf, 0, -1
            UF(k)=Q(k)
        end do

*-----
        do j=0, NX
            P(j) = ZERO                ! Erase P
        end do

*----
        do j=0, NZ
            Q(j) = ZERO                ! Erase Q
        end do

*-----

    w1=(-1)**a(kk+1)
    if (w1.gt.0) then
        UF(0)= UF(0) - 1
    else
        UF(0)= UF(0) + 1
    end if

**----- Fine Calcolo di F -----

```

```

**----- Inizio Calcolo di L con a(kk)=1
*-----
      do j=0, NX
        P(j) = ZERO                ! Erase P
      end do
*-----

      D=5
      w = cf0 + 1
      do k = 0, cf0
        r=0
        x=P(k) + UF0(k)*D + r
        r=int(x/pr)
        P(k) = x - r*pr
        P(k+1) = r
      end do
      if (P(w).eq.0) then w = cf0
*-----

      do j=0, NY
        s(j) = ZERO                ! Erase s
      end do
*-----

      cs=c1
      if (w.gt.c1) then cs = w
      r=0
      do h = 0, cs
        s(h)=VL(h) + P(h) + r
        r=0
        if (s(h).ge.pr) then
          s(h) = s(h) - pr
          r=1
        end if
      end do
      if (r.eq.1) then
        cs = cs + 1
        s(cs)=r
      end if
*-----

      do j=0, NX
        P(j) = ZERO                ! Erase P
      end do
*-----

      w = cs + c1 + 1
      do k = 0, cs
        r=0
        do h = 0, c1
          x=P(h+k) + s(k)*VL(h) + r
          r=int(x/pr)
          P(h+k) = x - r*pr
        end do
        P(h+k) = r

```

```

end do
if (P(w).eq.0) then w = w - 1
*--
  r=0
  do k = w, 0, -1
    Q(k)= INT((P(k)+r*pr)/2)
    r= MOD(P(k),2)
  end do
if (Q(w).eq.0) then w = w - 1

  cl=w
  w=0
  do k = cl, 0, -1
    VL(k)=Q(k)
  end do

  w1=(-1)**a(kk+1)
  if (w1.gt.0) then
    VL(0)= VL(0) - 1
  else
    VL(0)= VL(0) + 1
  end if
*-----
  do j=0, NY
    s(j) = ZERO           ! Erase s
  end do
*---
  do j=0, NX
    P(j) = ZERO          ! Erase P
  end do
*---
  do j=0, NZ
    Q(j) = ZERO          ! Erase Q
  end do
*-----
**----- Fine Calcolo di L -----
* ----- Fine IF principale del loop -----
  end if
  end do      ! ..... chiusura del Loop

**----- Risultati -----
  write (*,5000) z
*-----
  upper= int(digits/7.0)+10  ! +10, per sicurezza sui round-off!
  write(*,*)
  do j = upper, 0, -10
    if (int(UF(j)).ne.0) then
      write(*,6004) int(UF(j)), int(UF(j-1)), int(UF(j-2)),
        .int(UF(j-3)),int(UF(j-4)),int(UF(j-5)),int(UF(j-6)),

```

```

        .int(UF(j-7)),int(UF(j-8)),int(UF(j-9))
        end if
    end do
* -----
    write(*,*)
    write(*,*) '   n.cifre= ', ANINT(digits)
    write(*,*)
* -----
*   -----
*   Formati
*   -----
*
1000 format (2x,'Calcolo dei Numeri Grandi di Fibonacci',
           .   /2x,'-----')
2007 format (/2x,'Immettere il numero della serie F(z)'
           .   /2x,'di Fibonacci z= -----> ', $)
3007 format (i10)
*----- New Formats-----
5000 format (2x,'N.ro Fibonacci [trascritto su 80 colonne] F(',i6,')=')
6001 format (3x,i2,5x,i1,4x,i8)
6004 format (10i8.7)
*
7001 format (5i8.7)
*=====
        end      ! Fine del Programma
*=====

```

L'output del numero F(15100), che va depurato delle settuple nulle finali e degli zeri iniziali del primo gruppo, è il seguente. Ricordiamo solo che i comandi per ottenerlo su file sono: ./fibvelo > 15p1.txt (in Linux, via terminale) e fibvelo > 15p1.txt (da DosPrompt). Dopo l'invio inserire il numero e ancora invio.

Calcolo dei Numeri Grandi di Fibonacci

Immettere il numero della serie F(z)

di Fibonacci z= ----->

Numero iterazioni stimate= 14

n	a(k)	x(k)
0	0	15100
1	0	7550
2	1	3775
3	1	1887
4	1	943
5	1	471
6	1	235
7	1	117
8	0	58

9	1	29
10	0	14
11	1	7
12	1	3
13	1	1

Numero iterazioni necessarie= 13

N.ro Fibonacci [trascritto su 80 colonne] F(15100)=

```

0231144 0787374 3545746 7992087 0523086 4105782 5793372 6209923 2867641 7486540
3785580 3953843 4890384 4794568 6144899 1375905 8819993 5804837 0129113 4609641
0355557 3850021 6929503 9957333 1725854 9478234 0375222 4326830 6203195 2728334
8274071 3586269 8213317 7253839 1478709 9467805 4694063 3373925 2180736 3705676
7122460 0517956 6716837 3017437 2396136 8200476 9088025 1056146 6527563 8311386
5667460 3358444 6623730 1288688 3686108 9884608 2235328 7261804 9636731 1107627
5454554 6420230 8808416 3136546 2754195 6150139 2854828 4835028 5607477 7848887
2936778 9278956 9778469 5782119 9907607 3206254 3295402 8738374 2432654 8414053
3045299 8801793 8823167 0496394 5307897 3776281 7718857 0309250 4092565 2391219
3933065 9138824 4297275 4647807 6988373 0925169 7087335 0993671 5451455 5745487
0619073 6817880 1307816 0449002 8374454 1967523 8753713 4251462 8924126 8613196
7010313 1550213 9912885 1832382 2713817 6735902 7401620 2837777 3066392 9689057
9347341 7498940 6097268 7234208 4208004 5693528 0372549 7647460 7318238 6773380
1131910 0220544 6897219 4393070 3740132 0767804 9738956 4048608 3520102 3528564
9388565 1560904 9203855 5429270 7352483 3524257 2075556 2201513 0661779 5066139
0889784 7587663 0206780 2317404 6734140 1480995 8651157 3547071 4912724 1345699
8911896 7584212 5049625 7973038 9458930 8638990 8409843 0479430 9938857 1503121
3041997 2082773 8649545 1814126 9144632 0251584 4597725 1568188 8843251 4455136
1501351 2101082 7686318 3454166 8361274 7018807 8717416 4856177 5077369 3957382
4184340 8474534 6020126 5175812 4605699 5650956 2637604 3695895 6548897 1500118
8613486 5869871 8925952 4700728 1859790 0920768 3569393 0245024 5949624 0732698
7129732 4372413 3042545 1774518 3659468 6852114 7224199 2879301 2497080 0956317
6593747 7598683 5865484 9053615 6389250 2058704 2384106 9233311 2888325 2492629
7715853 7585985 0804719 7655545 5982990 6132233 6473668 6555497 7724923 4215278
6880794 0464689 2692752 0389126 2270789 4189135 9920336 0687514 3031591 9928493
1907186 2058442 9194925 2843232 1544338 2276209 7699918 8216412 0472345 9446179
1395487 0686413 1992912 5047436 9152862 9170769 2021004 2388994 5787212 6036340
4199528 9696234 9498412 4209826 4870125 7359550 1911761 6533181 0445138 8030419
1387918 4019385 2145703 3233389 7831913 8880533 9604641 3086554 0752697 2748360
3742096 0740312 1747891 9336300 4974902 7757318 7400503 6363834 8330554 7867782
1117911 1992937 0956648 8182471 5163873 4375561 6648433 4890274 6671251 3944229
4818300 6757560 2785821 5059458 7894895 9731957 5615084 6137008 0236750 7244066
9325564 6704397 7726507 2329189 4932136 2792241 0787609 2241177 3478061 7263490
2827363 9022566 8134178 2753045 1382760 2090947 5036361 3012567 8452885 1579294
2950248 0182346 9586276 6012493 7418925 0531919 0263897 5446981 4153502 1715122
5553816 5107848 3566291 3454185 5473930 3292600 9197151 9102688 4683936 0122689
7230716 0966753 7242828 3443088 5582142 7853154 4766902 4790338 9283548 0590507
7436699 1663248 5515110 7177346 9996512 3988588 0215113 8428733 7485985 6832076
1291992 6906772 4073720 8370127 3003550 9764102 1571039 6606361 2191065 6868845
0478823 9018594 8885032 3320915 2829365 6783462 3934184 0651813 2545626 2832034

```

7074742 9291692 0045007 6381809 2873283 8036172 8439260 5430704 3480206 5294164
3989650 7024260 7054476 6682528 4082369 7004092 5883788 3548225 4310447 5805120
5928774 0468947 8969028 8433726 0047571 3243167 8156074 2525089 6800807 9715122
5594279 8694459 2932047 9358063 6188829 7774524 3465969 4086610 0550317 3394014
2255376 5404018 6792023 0277705 6990077 4035979 5353719 5834911 7027104 6307934
2145075 0000000 0000000 0000000 0000000 0000000 0000000 0000000 0000000 0000000

n.cifre= 3156.

Gli eseguibili

Sono stati inseriti in questo link: <http://astrodinamica.altervista.org/gzip/fibo.tar.gz>.
E l'elenco è il seguente:

75.075 fibvelo.exe	Eseguibile di fibvelo.for in DOS
34.134 fibvelo	Eseguibile di fibvelo.for in Linux
211 15p1.com	Eseguibile di fibo.asm -> MAX 15,100 Numeri
211 fibo.com	Eseguibile di fibo.asm -> MAX 150,000 Numeri

8.659 fibo.asm	Sorgente Assembler
10.028 fibvelo.for	Sorgente Fortran

Conclusione

E' significativo osservare come non ci possa essere alcun dubbio nel sostenere che l'algoritmo esposto sia superiore al metodo tradizionale non solo per la sua *intrinseca eleganza* ma anche per la praticità nel raccogliere e conservare i risultati ottenuti.

..... 19-10-2006