An Efficient Algorithm for Locating the Global Maximum of an Arbitrary Univariate Function

Richard E. Haskell
Gabriel Castelino
Bahram Mirshab

Abstract

A Forth implementation of an algorithm for locating the maximum of an arbitrary univariate function using a global search technique is presented. The algorithm is based on a method suggested by Kushner. The method discussed in this paper will not get stuck on local maxima and does not depend on the function being well-behaved or easy to calculate or measure. For the case where the value of the global maximum is known *a priori* (a not uncommon case) the algorithm becomes very simple and efficient. Using a discrete test function that has two local maxima with values very close to the global maximum an experiment was performed in which the location of the global maximum was shifted to all possible values within a specified fixed range of the independent variable. The average number of search points needed to locate the global maximum was found to be 12% of the total number (compared to 50% for an exhaustive search) and the maximum number of search points needed (worst case) was found to be 25% (compared to 100% for the exhaustive case).

Introduction

Finding the maximum (or minimum) of a function is an important problem that often occurs in engineering design. Many optimization algorithms involve local search techniques which can get stuck on local maxima. Global search techniques strive to find a global maximum in the presence of other local maxima. Most search techniques evaluate the function at several points; the location of each new test point depends on the value of the function at the previous test points. In many situations the evaluation of the function is a very time-consuming process and in such cases it is important that the maximum be found using a minimum number of test points. If the value of the maximum is known *a priori* then the search can be terminated as soon as a test point gives the maximum value. The global search technique described in this paper is based on a method suggested by Kushner [1,2] in which the unknown function is modeled as a stochastic process conditioned on discrete measurements of the function. For a univariate function if the value of the global maximum is known (or an upper bound can be given), Kushner's method leads to a simple, elegant algorithm. The main advantage of the algorithm is that it does not make any assumptions about the differentiability of the function. The method has been extended to multi-variable functions by Stuckman [3,4].

Definitions:

Before describing the algorithm, we will define the parameters used. Figure 1 shows an arbitrary univariate function, $g(z)$, where z is the independent variable. Each iteration of the algorithm is performed on an interval $[z_{\min}, z_{\max}]$. We assume that the maximum value of the function is G. Then the other parameters may be defined as:

$$d_{\max} = G - g(z_{\max})$$
$$d_{\min} = G - g(z_{\min})$$

During each iteration a point within the interval is selected as a new candidate for the maximum. This point is called $\hat{z}$.

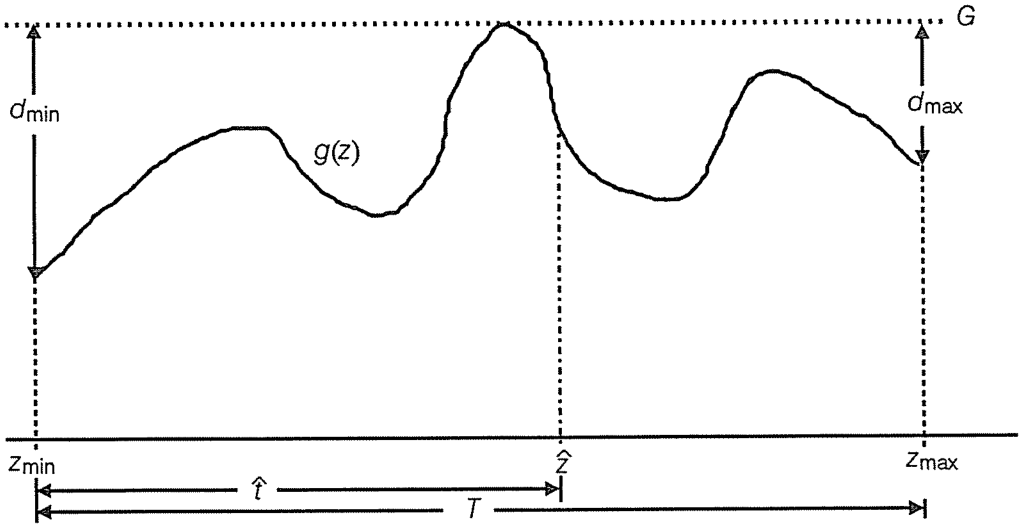

Fig. 1 Definition of variables used in the algorithm.

The Algorithm

The algorithm to be described is based on the analysis described in Appendix A. The algorithm is supplied with the initial test interval $[z_{\min}, z_{\max}]$ and then goes through a series of iterations until the maximum value has been determined. The algorithm may be described as follows:

1. If $g(z_{\min}) = G$ or $g(z_{\max}) = G$ then stop.

   DO the following steps n times or until a maximum has been found.
2. Compute the value of $\hat{z}$, as follows:
$$\hat{z} = z_{\min} + \hat{t}$$
   where, $\hat{t} = \dfrac{d_{\min} T}{(d_{\min} + d_{\max})}$

   and $T = z_{\max} - z_{\min}$ as shown in Figure 1.
3. If $g(\hat{z}) = G$ then stop.
4. Divide the interval into two smaller intervals— $[z_{\min}, \hat{z}]$ and $[\hat{z}, z_{\max}]$.

   (Note that $\hat{z}$ becomes the new $z_{\max}$ for the first interval, and the new $z_{\min}$ for the second interval.)
5. Compute a parameter $A_{\min}$ for each of the new intervals:
$$A_{\min} = \frac{d_{\min} d_{\max}}{T}$$
   and add each interval to a segment list; each entry in the list contains $A_{\min}$, $z_{\min}$, $z_{\max}$, $d_{\min}$ and $d_{\max}$.

6. Select the entry with the smallest $A_{\min}$ in the segment list; the new test interval is the corresponding $[z_{\min}, z_{\max}]$.

The number of iterations of the algorithm, n, may be specified when invoking the algorithm or may be preset to some fixed percentage (for example 10%) of the total search space. If a maximum is not found after n iterations, the result of the algorithm is the closest value to the maximum that is found.

The Forth Implementation

The Forth implementation of the algorithm described above is fairly straightforward. It assumes that the function has integer values at all points. This can be extended easily for a function with real values or the function may be quantized to appropriate integer values.

The main Forth word is **find.max** which is invoked using the following format:

$<z_{\min}>$ $<z_{\max}>$ $<$*#iter*$>$ **find.max**

where $z_{\min}$, and $z_{\max}$ identify the interval in which a global maximum is sought, and *#iter* is the number of iterations.

A listing of the Forth program is found in Appendix B. The function to be tested must be defined before the program is loaded. In the listing supplied, the function is defined on Screen 11, while the program occupies Screens 12 through 16. A new function may be defined by changing the definition of **g(z)**. When defining **g(z)**, the value of the global maximum must also be specified by initializing the constant **G**.

The function **find.max** first evaluates $d_{\min}$, and $d_{\max}$ by invoking the word **d(z)**. If either $d_{\min}$, or $d_{\max}$ is zero, then the global maximum has been found, **maxz** is set to $d_{\min}$, or $d_{\max}$ and **find.max** terminates. Otherwise, **find.max** calls **once** which performs the computations involved in one pass of the algorithm. The word **once** returns with a true flag if a maximum is found. In the case when a maximum is not found, **once** returns a new test interval under a false flag.

Each iteration of **once** computes the value of $\hat{z}$ using the word **z^**. The function is evaluated at $\hat{z}$ and tested against **G**. If the maximum is not found, the maximum value found so far, **maxg**, is updated.

In computing the value of $\hat{z}$, due to truncation, it is possible that the value of $\hat{t}$ will be zero. This would result in $\hat{z} = z_{\min}$. This is not desirable for two reasons — first, the function has already been evaluated at $z_{\min}$ without success, and second, it will lead to two new intervals that are already in the segment list. To avoid this possibility the value of $\hat{t}$ is forced to be at least 1.

If no maximum is found, then the interval is divided into two smaller intervals and the value of $A_{\min}$ is computed using the word **amin**. The value of $A_{\min}$ along with the endpoints of the interval and the values of d at the endpoints are then added to a segment list whose root is the variable **head [5]**. The list is maintained in ascending order of $A_{\min}$ by the word **putlist**; the new interval to be tested is the one with the smallest $A_{\min}$ value, and will always be at the top of the list. The word **nextseg** retrieves the new interval from the top of the segment list. As discussed in the paragraph above, in some cases $\hat{t}$ is forced to be 1 and the new interval will have a length of 1. Obviously, such an interval will be non-productive in subsequent tests, and hence the value of $A_{\min}$ is not computed for this interval and it is not placed in the segment list.

Test Results

The Forth implementation was successfully tested using several different functions. The function selected as a test case for this paper is one that has local maxima with values very close to the global maximum. The test function is a sawtooth function defined over an interval [0, 999]. Several trials were performed; for each trial a different sub-interval was selected such that the

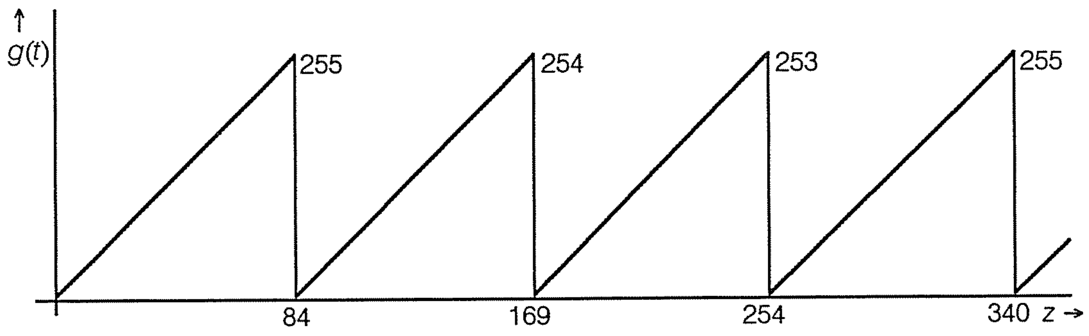

Fig. 2 Test function defined by look-up table in Screen #11.

position of the global maximum within the sub-interval was different from the previous trial. A section of the test function is shown in Figure 2. The test function is periodic and so the pattern is repeated for the entire interval [0, 999]. The function definition is on Screen 11.

A set of trials was conducted in which the length of the sub-interval was fixed but its location varied by increments of 1 (at both ends). Since the function is fixed, this has the effect of moving the maximum to a different location each time. For example the first interval chosen was [85, 340], while the second trial was performed on the interval [86, 341]. Continuing in this way the location of the maximum shifts from the right end of the interval to the left end. On the average, the global maximum was found after only 12% of the total search space was tested. This can be compared to the average of 50% for an exhaustive search. In no case was it necessary to test more than 25% of the total number of points in order to locate the global maximum. Again, this is far better than the worst case of 100% for an exhaustive search.

In the listing of the Forth implementation, the above trials are performed by the word `test1` on Screen 18, except for the fact that the sub-interval location is incremented by 26 rather than 1. This was done to obtain an output sample that would be suitable to include with this paper. The result of executing `test1` can be seen in Figure 3. Note in this example the average number of iterations needed to find the global maximum is only 8% of the total search space.

In another series of tests, the number of iterations was fixed at 20% of the search space and `find.max` was executed for different intervals. In this case the global maximum was found in 96% of the trials. In all cases that failed the maximum obtained was close to the global maximum.

```
 ok
test1
     Range            #iter   Max.   at z =    %
------------------------------------------------
 85    ->   340          1    255     340      0
111    ->   366         27    255     340     10
137    ->   392         16    255     340      6
163    ->   418         16    255     340      6
189    ->   444         50    255     340     19
215    ->   470          1    255     340      0
241    ->   496         15    255     340      5
267    ->   522         16    255     340      6
293    ->   548         37    255     340     14
319    ->   574         51    255     340     19
Avg. % iterations = 8   ok
```

Fig. 3 Sample run of `test1` in Screen #18.

Conclusions

This paper has introduced an efficient algorithm for locating the global maximum for the special case when the value of the global maximum is known *a priori*. The algorithm does not make any assumptions about the differentiability of the function and so can be applied to all functions. After a series of tests of the algorithm it was found that, on the average, the global maximum is found after only 12% of the search space has been tested. This makes it ideal for problems that involve functions that are difficult or expensive to compute.

References

1. Kushner, H.J., "A Versatile Stochastic Model of a Function of Unknown and Time Varying Form," *J. Math. Anal. & Appl.*, **5**, 150-167, 1962.
2. Kushner, H.J., "A New Method of Locating the Maximum of an Arbitrary Multipeak Curve in the Presence of Noise," *Proceedings of the Joint Automatic Control Conference*, 1963.
3. Stuckman, Bruce E., "A Global Search Method for Optimizing Non-linear Systems," Ph.D. Thesis, Oakland University, Rochester, MI, 1987.
4. Stuckman, Bruce E., "A Method of Global Optimization Using Brownian Motion for Non-linear Systems," *Proceedings of the 30th Midwest Symposium on Circuits and Systems*, Syracuse, NY, 1987.
5. Pountain, Dick, *Object-Oriented Forth, Implementation of Data Structures*, Chap. 3, Academic Press, London, 1987.

Dr. Haskell received a PhD in electrical engineering from Rensselaer Polytechnic Institute in 1963. He is currently Professor of Engineering at Oakland University in Rochester, Michigan where he teaches courses in computer science and engineering including a microprocessor design course that uses Forth. His current research interests are in the area of computer learning and pattern recognition.

Dr. Castelino received a PhD from the School of Engineering and Computer Science at Oakland University in Rochester, Michigan. He is currently on the faculty at the GMI Engineering & Management Institute in Flint, Michigan.

Mr. Mirshab is a graduate student in the School of Engineering and Computer Science at Oakland University in Rochester, Michigan.

APPENDIX A: Derivation of $\hat{t}$ and $A_{\min}$

In this section we will show how the expressions for $\hat{t}$ and $A_{\min}$ may be derived. To facilitate the derivation, we define a new variable $t = z - z_{\min}$ (this has the effect of moving the origin to $z_{\min}$), so that the test interval is now $[0, T]$ where $T = z_{\max} - z_{\min}$. The function being optimized is now $g(t)$.

Kushner [1,2] modeled the unknown function as a Brownian motion process and showed that if the value of the function is known at the endpoints of an interval, then within the interval, the expected value of the function varies linearly with its distance from one of the endpoints, and the variance of the function is quadratic in nature. This is illustrated in Figure 4.

The expected value of the function, and its variance can be expressed as

Expected value:
$$\mu(t) = g_0 + \frac{t}{T}(g_T - g_0) \tag{1}$$

Variance:
$$\sigma(t) = \frac{ct\,(T - t)}{T} \tag{2}$$

where $g_0 = g(0)\,, g_T = g(T)$

and c is a constant.

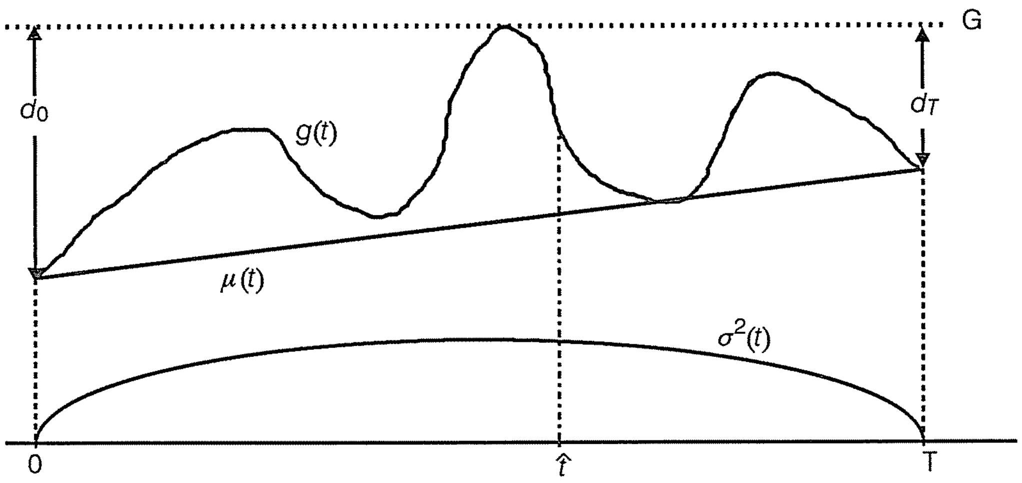

Fig. 4 Illustrating the linear expected value and quadratic variance of the unknown function.

We select the location of the next test point, $\hat{t}$, as the point within the interval at which the probability that $g(t) > G$ is maximum. Since $g(t)$ is normally distributed, this probability is

$$P(g > G) = 1 - \Phi\left(\frac{G-\mu}{\sigma}\right) = 1 - \Phi(\sqrt{A}\ ) \tag{3}$$

where Φ is the cumulative normal distribution function of $g(t)$ and

$$A = \left(\frac{G-\mu}{\sigma}\right)^2$$

Using Eq.(1), we can rewrite the quantity $(G - \mu)$ as

$$\begin{aligned} G - \mu &= G - g_0 - \frac{t}{T}(g_T - g_0) \\ &= d_0 + \frac{t}{T}(d_T - d_0) \end{aligned} \tag{4}$$

where $d_0 = G - g_0$
and $d_T = G - g_T$
The probability in Eq.(3) can be maximized by minimizing the quantity A. Combining Eqs.(2) and (4), A can be expressed as

$$A = \frac{T\left[d_0 + \frac{t}{T}(d_T - d_0)\right]^2}{c\,t\,(T-t)} \tag{5}$$

To minimize A, we set $\frac{dA}{dt} = 0$, and solve for $t = \hat{t}$. This gives

$$\hat{t} = \frac{d_0\,d_T}{d_T + d_0} \tag{6}$$

and the minimum value of A is obtained by substituting Eq.(6) in Eq.(5). Thus

$$A_{\min} = \left(\frac{4}{c}\right)\frac{d_0\,d_T}{T} = k\frac{d_0\,d_T}{T}$$

The constant k will not affect comparisons of $A_{\min}$ values for different intervals and is hence not included in computing the value of $A_{\min}$ in the algorithm described.

Appendix B: Source Listing

```
Scr # 11
\ Calculate function                              reh 20Dec88

VARIABLE val        0 val !

: getval  val @ 3 + 256 MOD DUP val ! ;

CREATE func.table 1000 ALLOT      \ function table

: fillup   ( addr -- )            \ fill function table
       1000 0 DO getval OVER C! 1+ LOOP DROP ;

: g(z)     ( z -- g )                \ evaluate function
       func.table + C@ ;

255 CONSTANT G          \ value of the global maximum

```

```
Scr # 12
\ Global search - List functions                  reh 12Sep88

VARIABLE head
VARIABLE mark.dp

: node   ( -- node)                 \ create new node
       HERE 14 ALLOT ;

: @node ( node -- dmin zmin dmax zmax )  \ get node values
       DUP 10 + @ OVER 6 + @ 2 PICK 12 + @ 3 ROLL 8 + @ ;

: !node ( dmin zmin dmax zmax damin node --) \ store node values
       DUP >R 2+ 2! R@ 8 + ! R@ 12 + ! R@ 6 + ! R> 10 + ! ;

: insert.node   ( node ptr -- )      \ insert new node into list
       2DUP @ SWAP ! ! ;
```

```
Scr # 13
\  List functions - cont                          reh 11Sep88

: findloc   ( damin -- ptr )  \ locate position for new node
    head  BEGIN
            DUP 2SWAP ROT @ ?DUP
          WHILE 2+ 2@ >R >R 2DUP R> R> DU<
            IF 2DROP EXIT THEN ROT @
          REPEAT 2DROP ;

: putlist  ( dmin zmin dmax zmax damin -- ) \ add values to list
                                            \ in ascending order
       2DUP findloc node DUP ROT insert.node !node ;

: showall       ( -- )   \ print out list  dmin zmin dmax zmax
       head CR BEGIN @ ?DUP
       WHILE DUP @node 3 ROLL . ROT . SWAP . . CR REPEAT ;
```

```
Scr # 14
\ Global optimization                             reh 20Dec88
: G-g(z)  ( g -- d )
       G SWAP OVER MIN - ;

: d(z)   ( z -- d )
       g(z) G-g(z) ;

: z^     ( zmin T dmin dmax -- z^ )
       OVER + -ROT UM* ROT UM/MOD NIP
              1 MAX + ;

: amin  ( dmin zmin dmax zmax -- )
       DUP 3 PICK - DUP 1 > IF
       2 PICK 5 PICK
       UM* ROT MU/MOD ROT DROP
       putlist ELSE DROP 2DROP 2DROP THEN ;
```

```
Scr # 31
\ Global search                                    reh 20Dec88

A test function is stored in the table func.table.

getval   Produces a saw-tooth like function with 2 local maxima
         for each maximum.


fillup   Fills the table with 1000 values


g(z)     evaluates the function g from the lookup table
         ( z is the location in the table 0 - 999)

G is the maximum value of the function.

```

```
Scr # 32
\ Global search                                 v   02reh 11Sep88

Head points to first node in list.
Each node contains 14 bytes,
| ptr. to next node| amin HI | amin LO | zmin | zmax |
| dmin | dmax | .

Node is node address.
Ptr is address of previous node.

@node   fetches the values dmin,zmin and dmax,zmax from the node

!node   stores the values dmin,zmin, dmax,zmax and damin in node

insert.node   insert node with address "node" just after
                node with address "ptr".
```

```
Scr # 33
\ Global search                                    reh 11Sep88
Nodes are stored in increasing order of amin.

findloc   Finds the address "ptr" of the node after which
          amin is to be inserted.





putlist   Creats a new node, inserts it in the proper place
          in the list, and stores damin, dmin, zmin,
          dmax and zmax in the node.

showall   Prints all nodes in the list in the order
          dmin zmin dmax zmax

```

```
Scr # 34
\ Global search                                    reh 20Dec88


d(z)     Calculates    d = G - g(z).


z^       Calculates    z^ = zmin + ( dmin * T )/ ( dmin + dmax )
                       T  = zmax - zmin


amin     Calculates   amin = (dmin * dmax) / T
         as a double number and puts it in the list.
         If zmax - zmin = 1, there is no new segment so
         don't bother putting anything on the list.


```

```
Scr # 15
 \                                                     reh 11Sep88
 VARIABLE maxg           \ max. value so far
 VARIABLE maxz           \  at z = maxz
 VARIABLE cnt            \ # of iterations

 : maxupdate     ( z g -- z g)
         2DUP maxg @ OVER <
         IF maxg ! maxz ! ELSE 2DROP THEN ;

 : globinit      ( -- )
        0 cnt !  0 head ! 0 maxg ! HERE mark.dp ! ;

 : nextseg  ( -- dmin zmin dmax zmax ff|list 0 0 0 tf)
         head DUP @ DUP IF DUP @ ROT ! @node FALSE
                        ELSE 0 0 TRUE THEN ;

Scr # 16
 \                                                     reh 20Dec88
 : once  ( dmin zmin dmax zmax -- dmin zmin dmax zmax f )
         DUP 3 PICK TUCK - 5 PICK 4 PICK
         z^ DUP g(z) maxupdate DUP G-g(z) -ROT G < NOT
         IF 2DROP TRUE EXIT THEN
         2SWAP 2OVER 2SWAP amin amin
         nextseg ;

 : find.max    ( zmin zmax # -- )
         globinit 2 PICK d(z) DUP
         IF 2 PICK d(z) DUP
            IF 4 ROLL SWAP 4 ROLL 4 ROLL 1+ 1 DO once
                   IF I cnt ! LEAVE THEN LOOP 2DROP 2DROP
            ELSE 2DROP DROP NIP maxz ! G maxg ! THEN
         ELSE 2DROP DROP maxz ! G maxg ! THEN
         mark.dp @ DP ! ;

Scr # 17
 \                                                     reh 20May88

 : .title
         CR 6 SPACES ." Range" 9 SPACES ." #iter" 3 SPACES
         ." Max." 2 SPACES  ." at z ="  3 SPACES ." %" CR ;

 : .line  ." ---------------------------------------------" ;

 : .num  4 .R 3 SPACES ;         \ print number justified








Scr # 18
 \ Sample test                                         reh 20Dec88

 : test1
         .title 0 val !
         func.table fillup .line
         0 341 85 DO CR I
         DUP .num ." -> "
         DUP 255 + DUP .num 3 SPACES
         100 find.max
         cnt @ .num maxg @ .num maxz @ .num
         cnt @ 100 256 */ DUP .num
         + 26 +LOOP
         10 / CR ." Avg. % iterations = " . ;
```

```
Scr # 35
 \ Global search                                           reh 11Sep88
 maxg          Contains max value of g found so far.
 maxz          Contains value of z corresponding to maxg
 cnt           Contains the number of iterations

 maxupdate     checks g for maxg
               if g > maxg then replace maxg with g & update maxz

 globinit      initialization     cnt = 0   head = 0
                                  maxg = 0  mark.dp = HERE

 nextseg  Removes the first node in the list ( containing the
         smallest amin ) and puts the four values dmin, zmin, dmax
         and zmax on the stack under a true flag.  If the list is
         empty four arbitrary (dummy) values are left on the
         stack under a false flag.

Scr # 36
 \ Global search                                           reh 20Dec88
 once  Does the following for a segment from zmin to zmax.
       Finds z^ and g^ and update maxg.
       If g^ = G, quit
       else divide T  into two segments
       T1 = z^ - zmin
       T2 = zmax - z^
       Calculate amin for each segment and add
       two new segments to segment list.
       Choose segment with the lowest amin and remove from
       segment list.
 find.max  Finds the location of the global maximum G.
           ( # is maximum number of iterations )
           If not found in # iterations, maxg contains the
           maximum value found so far.  maxz contains the value of
           z corresponding to maxg.  cnt is the no. of iterations.

Scr # 37
                                                           reh 24Mar88
 .title print out
                   Range       #iter    Max.  at z =    %

 .line prints a line

 .num  prints a number right justified in a field of width 4
         then leaves 3 spaces.









Scr # 38
                                                           reh 20Dec88
 test1    calls find.max 10 times with the range zmin-zmax
          varying from 85-340 to 319-574.  The maximum always
          occurs at 340.  Thus the initial width T is always
          255 and the maximum value at 340 moves through
          values of z starting at zmax and moving to zmin
          in steps of 26.

          For each call of globmax the number of iteration
          needed to find G is printed out together with the
          percent (cnt*100/256).  At the end the average percent
          for all 10 trials is computed and printed out.
```