

Project: 1.1 Golden Section Search for the Mode of a Function

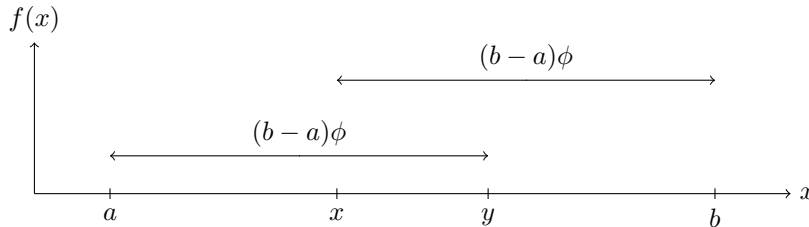
January 23, 2018

1. **Prove that the subinterval in which the mode is deduced to lie is found to be already divided in golden section from one end by the point in its interior at which we already have a function evaluation.**

Let $\phi = \frac{\sqrt{5}-1}{2}$. We know that $a < x < y < b$ and,

$$y = (b-a)\phi + a \quad (1)$$

$$x - a = b - y \quad (2)$$



We can see from the diagram (and mathematically from $x - a = b - y$) that there is symmetry, so WLOG we can assume that we find that the mode lies in the subinterval $[x, b]$. Now consider the ratio of the larger interval within the subinterval, $[y, b]$, over the whole subinterval, $[x, b]$. We can see that (using the formulae given in the question - (1) and (2)),

$$\frac{b-y}{b-x} = \frac{b - ((b-a)\phi - a)}{b - (a + b - (b-a)\phi - a)} = \frac{(b-a)(1-\phi)}{(b-a)\phi} = \phi,$$

since ϕ satisfies $\phi^2 + \phi - 1 = 0$.

Programming Questions

- (a) How does your program deal with the possibility that $f(x) = f(y)$ on one or more iterative steps?
If $f(x) = f(y)$ then, since we know it doesn't matter whichever subinterval we use (by symmetry), my program will use the same method for the case of $f(x) > f(y)$.
- (b) Is it preferable to use equation (1) or equation (2) to locate the point for the second function evaluation in each new subinterval, and why?
It is preferable to use equation (2) since the total number of operations in equation (1) is 3 compared to only 2 in equation (2). Hence the overall number of operations would be reduced when using equation (2) so the speed of the algorithm would be faster. However the complexity of both equations is the same.
- (c) How would your program function if the mode's position were at an end-point of the original interval?
My program would function correctly and end up keeping the actual mode as the end of the interval for all iterations of the golden section search. Below is an example, with the key on the next page.
Input into MATLAB: `[M, N, A] = goldensectionsearch(5 - x^2, 0, 1, 1e-2, 1)`

Iteration	xx	yy	a	x	y	b
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
9	4.99993	4.99983	0	0.00813062	0.0131556	0.0212862
10	4.99997	4.99993	0	0.005025	0.00813062	0.0131556

Table 1: Example of program working for Mode at the endpoint of the interval.

2. **As a check that you understand the method, first program it to find the position of the mode in $[0, 1]$ of the function**

$$f(x) = 1 + x + x^2 - 4x^4$$

to some appropriate accuracy. Your output should include the mode, the number of iterations performed and an indication of how accurate your result is.

Input into MATLAB: `[M, N, A] = goldensectionsearch(f, 0, 1, 1e-5, 1)`

This programme works as predicted. The actual value of the mode, m , lies in the range:

$$M - A \leq m \leq M + A$$

Key		
$f(x) = 1 + x + x^2 - 4x^4$	$xx = f(x)$	N = Iteration termination value
$yy = f(y)$	M = Mode value	A = Accuracy of Mode value obtained
goldensectionsearch(function, Lower Bound, Upper Bound, Precision, Min=0/Max=1)		

Table 2: Key for what each input/output argument represents in the function as well as understanding what the headers mean in the table below.

Iteration	xx	yy	a	x	y	b
1	1.44272	1.41641	0	0.381966	0.618034	1
2	1.27937	1.44272	0	0.236068	0.381966	0.618034
3	1.44272	1.49629	0.236068	0.381966	0.472136	0.618034
4	1.49629	1.49594	0.381966	0.472136	0.527864	0.618034
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
21	1.5	1.5	0.499954	0.49998	0.499995	0.50002
22	1.5	1.5	0.49998	0.499995	0.500005	0.50002
23	1.5	1.5	0.49998	0.499989	0.499995	0.500005
24	1.5	1.5	0.499989	0.499995	0.499999	0.500005

Table 3: Data from MATLAB for the specific values above

3. **What is likely to be the most time-consuming part of either algorithm in a real-life problem. How would the number of numerical operations required for this alternative algorithm compare with that required for the golden section search algorithm. Give quantitative estimates if possible. [Note that no additional computational work should be done to answer this question.]**

The most time consuming part of either algorithm would be calculating the value of the function for:

- a given value of x or y , in the golden search method.
- both values of x and y , in the alternative algorithm.

For any number of iterations, the alternative algorithm would require just under twice the number of calculations to match the same number of iterations as the golden search algorithm. This is because the alternative algorithm has to calculate both values of x and y within the interval, whereas the golden search algorithm can use a previous value of x or y . (However initially both must calculate two values of the function - so it is never exactly double the number of calculations)

Further to this, for the alternative algorithm, the interval size for each iteration is reduced to $\frac{2}{3}$ of the previous interval. However the golden search algorithm reduces the interval size to ϕ (0.618...) of the previous interval size. Since $\phi < \frac{2}{3}$, the golden search algorithm is more precise after each successive iteration. This means that not only does the golden search algorithm require less calculations after each iteration, it is also more precise after each iteration as the interval size for successive iterations is smaller than successive intervals for the alternative algorithm. Hence you would actually require less than half the number of computations for the golden search algorithm to obtain the same precision as the alternative algorithm.

Since in real life problems, we would want high precision and so we would be doing this for large n , where n is the number of iterations. In the limit as n is very large, the number of calculations the golden search algorithm would be doing would be $n + 1$ with complexity $O(n)$, whereas the number of calculations the alternative algorithm would be doing would be $2n$, with the same complexity of the order $O(n)$. The

Output Arguments	Values
M	0.5000
N	23
A	7.8029×10^{-6}

Table 4: Output Arguments obtained once the program terminated.

golden search algorithm would be up to twice as fast than the alternative algorithm, yet both would have the same complexity.

4. **What properties of the function $f(x)$ determine the numerical accuracy that is attainable?**

If the function is discontinuous at a discrete number of points, then the iteration methods will still locate the mode; as it will still approach the value of the mode, assuming the conditions of strictly decreasing and increasing are met on either side of the function.

At the mode, if the function is continuous and relatively flat, then the result of the algorithm can be less accurate compared to a mode that has a sharper change in gradient. Consider finding the maxima of the functions:

$$\begin{aligned} f(x) &= -x^8 \\ g(x) &= -x^2 \end{aligned}$$

When we run the program for both of these functions, using the same lower bound, upper bound and precision, the program gets to a point where it must differentiate the magnitude of numbers as small as 10^{-46} . This is beyond the precision of MATLAB and so this is inaccurate. This happens because for values of $f(x)$ for x close to 0, $f(x)$ will approximate a straight line since we have small numbers being raised to the 8th power, they will be very similar in magnitude (i.e very small). It will require a much greater precision to work out the difference in magnitude between $f(x_1)$ and $f(x_2)$, where $|x_1 - x_2| < 10^{-2}$.

Input into MATLAB, with the same key as before: [M,N,A]=goldensectionsearch($-x^8$,1,1, 10^{-5} ,1)

Iteration	xx	yy	a	x	y	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\dots
24	-7.48811×10^{-41}	-7.22247×10^{-46}	-2.15666×10^{-5}	-9.64487×10^{-6}	-2.27686×10^{-6}	\dots
25	-7.22247×10^{-46}	-7.22247×10^{-46}	-9.64487×10^{-6}	-2.27686×10^{-6}	2.27686×10^{-6}	\dots

Table 5: Last 2 iterations for $f(x) = -x^8$, showing how small $f(x)$ gets for small values of x .

Input into MATLAB, with the same key as before, however f is replaced with g : [M,N,A]=goldensectionsearch($-x^2$,1,1, 10^{-5} ,1)

Iteration	xx	yy	a	x	y	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\dots
24	-9.30236×10^{-11}	-5.18408×10^{-12}	-2.15666×10^{-5}	-9.64487×10^{-6}	-2.27686×10^{-6}	\dots
25	-5.18408×10^{-12}	-5.18408×10^{-12}	-9.64487×10^{-6}	-2.27686×10^{-6}	2.27686×10^{-6}	\dots

Table 6: Last 2 iterations for $g(x) = -x^2$, showing that the program is within a suitable range to ensure accuracy of the results.

Clearly the function that is not as flat at the mode, $-x^2$, is more accurate since it does not require doing calculations with numbers beyond the double precision of MATLAB ($\approx 2.2204 \times 10^{-16}$).

5. **If the mode was located to some accuracy, what would be the corresponding accuracy in the height of the mode? How does your answer depend on the properties of $f(x)$?**

Let's assume we have a unimodal function f with a mode, m , measured to some accuracy δ , i.e.

$$a = M - \delta \leq m \leq M + \delta = b,$$

where $[a, b]$ is the interval in which the mode lies in and: $\delta = \frac{b-a}{2}$; $M = \frac{a+b}{2}$.

We can find the average value of the function over this range. Let the average value be α , then we have the formula:

$$\alpha(b-a) = \int_a^b f(x)dx$$

We can then use α and the $\min(f(a), f(b))$ as the error in the height of the function, say

$$\epsilon = \alpha - \min(f(a), f(b)).$$

Therefore the error in the height of the function at the mode, m , is:

$$\alpha - \epsilon \leq f(m) \leq \alpha + \epsilon.$$

However, this would depend a lot on the properties of the function as to whether or not the height of the function at the mode, $f(m)$, lies in this range. If the function was sharply peaked at the mode, then ϵ may not be large enough to ensure the height of the mode is within the range above. This is because α won't necessarily lie at halfway, or higher, between the minimum of the function, $f(a)$ or $f(b)$, and the maximum of the function, $f(m)$, in the interval $[a, b]$.

6. **Taking $r = 6.5$, $R = 16$, $l = 24$ (all in cm), find the optimum values of d and θ using golden section search. Your program will need to carry out a double iteration. The inner iteration should find the variation of ϕ considered as a function of x (the distance SP); in other words, find $\Delta\phi = \max \phi(x) - \min \phi(x)$. The outer iteration then adjusts d to minimise $\Delta\phi$, and finally the optimum choice of θ can easily be made.**

- (i) Using the cosine rule, we can work out $\sin(\phi)$ in terms of x , d and l ($l = 24$ from the question).

$$\cos(90 - \phi) = \sin(\phi) = \frac{x^2 + 24^2 - d^2}{48x} \Rightarrow \phi(x, d) = \arcsin\left(\frac{x^2 + 24^2 - d^2}{48x}\right) \quad (3)$$

We know that $-1 \leq \sin(\phi) \leq 1$ and so we can find bounds on d .

$$\begin{aligned} -1 &\leq \frac{x^2 + 24^2 - d^2}{48x} & \frac{x^2 + 24^2 - d^2}{48x} &\leq 1 \\ 0 &\leq d^2 \leq (x + 24)^2 & 0 &\leq (24 - x)^2 \leq d^2 \\ 0 &\leq d \leq |x + 24| & 0 &\leq |24 - x| \leq d \end{aligned} \quad (4)$$

Since x represents the radius at which the stylus is at, x must vary from $r = 6.5$ to $R = 16$. Both $|x + 24|$ and $|24 - x|$ are linear functions of x in the range $6.5 \leq x \leq 16$, and so we need only consider both extreme values of x to see which d will satisfy all conditions.

For $x = 6.5$ and $x = 16$, we obtain the following 4 inequalities from (4).

$$\begin{aligned} \text{For } x = 6.5: & d \leq 30.5, 17.5 \leq d \\ \text{For } x = 16: & d \leq 40, 8 \leq d \\ \Rightarrow & 8 \leq 17.5 \leq d \leq 30.5 \leq 40 \\ \Rightarrow & 17.5 \leq d \leq 30.5 \end{aligned}$$

Hence $17.5 \leq d \leq 30.5$ will ensure that $\forall x \in [6.5, 16]$, $\sin(\phi)$ has real solutions. These ranges can therefore represent the intervals for which my program will run between since these are the values over which the function is valid. Using this information, I can now run my program to find the optimum value of d by applying the the golden search algorithm for both x and d .

Input into MATLAB: `goldensectionsearch2(17.5,30.5,6.5,16,10-5,10-5)`

Key
<code>goldensectionsearch2(LB of d, UB of d, LB of x, UB of x, Precision of d, Precision of x)</code>
*The d after each letter represents using the golden search algorithm while varying d (Outer Iteration)

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
1	7.55613	9.49999	17.5	22.46556	25.53444	30.5	141
2	5.86612	7.55613	17.5	20.56888	22.46556	25.53444	197
3	7.63396	5.86612	17.5	19.39667	20.56888	22.46556	253
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
27	4.694	4.69398	21.22954	21.22956	21.22957	21.22959	1597
28	4.69398	4.80602	21.22956	21.22957	21.22958	21.22959	1653
29	4.69399	4.69398	21.22956	21.22957	21.22957	21.22958	1709

Table 7: Table showing the output of my program to find the optimum value of d .

$$\text{Therefore the optimum value of } d = 21.22957. \quad (5)$$

We can now use this value of d to work out the optimum value of θ . If we substitute this value of d back into equation (3), we obtain:

$$\phi(x) = \arcsin\left(\frac{x^2 + A}{48x}\right), \text{ where } A = 125.305474775... \text{ (using } d = 21.22956724...).$$

We want the maximum absolute value of $(\phi - \theta)$ to be minimised. Therefore the value of θ will equal the average value of the maximum value and minimum value of ϕ .

Input into MATLAB: `goldensectionsearch(ϕ ,6.5,16,10-5,1)` and `goldensectionsearch(ϕ ,6.5,16,10-5,0)`

Maximum	Iteration	xx	yy	a	x	y	b
	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
	28	0.56692	0.5669198	6.5	6.500008	6.500013	6.500022
	29	0.5669202	0.56692	6.5	6.500005	6.500008	6.500013
Minimum	Iteration	xx	yy	a	x	y	b
	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
	28	-0.4852351	-0.4852351	11.19399	11.194	11.194	11.19401
	29	-0.4852351	-0.4852351	11.19399	11.19399	11.194	11.194

Table 8: Last 2 iterations of both programs which finds the x that corresponds to the maximum and minimum value of $\phi(x)$ for $x \in [6.5, 16]$.

$$\left. \begin{array}{l} x_{max} = 6.5000 \Rightarrow \phi(x_{max}) = 0.566920(078411490) \\ x_{min} = 11.1940 \Rightarrow \phi(x_{min}) = 0.485235(137267394) \end{array} \right\} \Rightarrow \theta = \frac{\phi(x_{max}) + \phi(x_{min})}{2} = 0.526078 \text{ radians}$$

This value of θ will minimise the maximum absolute value of $(\phi(x) - \theta)$.

- (ii) It is clear from the graphs below, Figure 1 and Figure 2, that the functions I was minimising/-maximising were indeed unimodal. Figure 1 shows how $\phi(x)$ behaves for different values of d and Figure 2 shows how $\Delta\phi(d)$ behaves as d varies. Hence all the graphs used in this question were indeed unimodal.

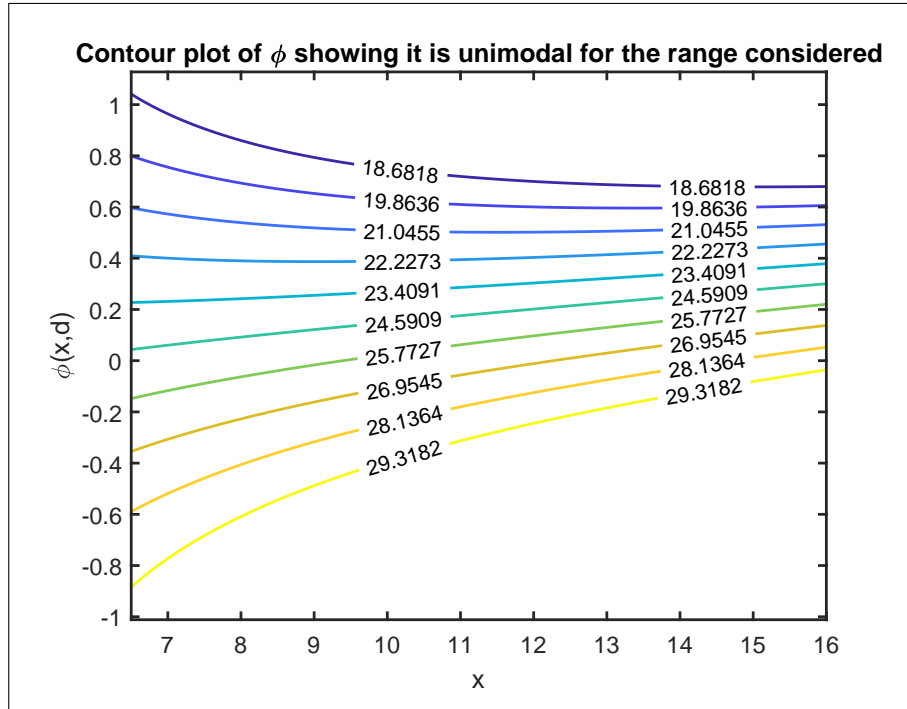


Figure 1: Graph showing a contour plot of $\phi(x, d)$ for different values of $d \in [17.5, 30.5]$

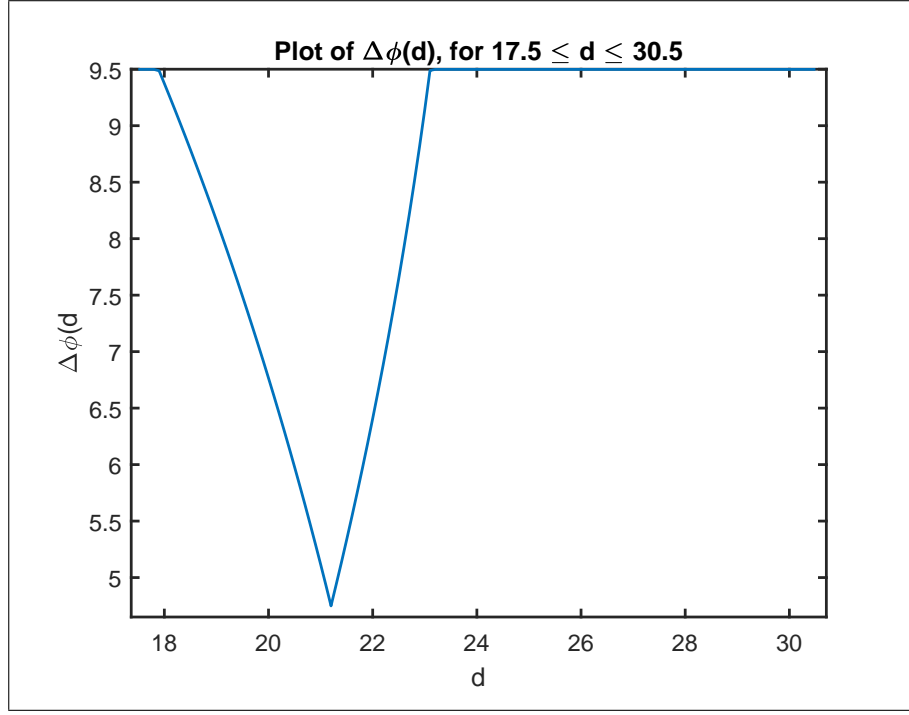


Figure 2: Graph showing a contour plot of $\phi(x, d)$ for different values of $d \in [17.5, 30.5]$

- (iii) To obtain d to an accuracy of one decimal place, we would require setting the precision of d and x to 10^{-1} . Increasing the precision of x is only useful when increasing the precision of d since the output of the inner iteration is going to tell us which, out of $\Delta\phi(xd)$ and $\Delta\phi(yd)$, is larger. The outer iteration would terminate before the inner iteration couldn't distinguish the difference in magnitude of two values of x and wrongly values that may be close to be equal.

Input into MATLAB: `goldensectionsearch2(17.5,30.5,6.5,16,10-1,10-1)`

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
1	7.45957	9.37502	17.5	22.46556	25.53444	30.5	46
2	5.82359	7.45957	17.5	20.56888	22.46556	25.53444	64
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
9	4.81249	4.68751	21.01663	21.12233	21.18765	21.29335	190
10	4.68751	4.61027	21.12233	21.18765	21.22802	21.29335	208

Table 9: Table showing the output of my program to find d correct to one decimal place.

We can see that from this table, $d = \frac{21.29335 \dots + 21.12233 \dots}{2} = 21.2078 \dots = 21.2$ (1 dp). If we compare this to our more accurate value of d , (5), we can see that our answer is indeed correct to one decimal place. The total number of inner iterations for this program is 208; so approximately 208 is required to obtain d correct to one decimal place.

However, running the program a second time with the precision of x being 0.1635, we can see we obtain the same answer of d correct to one decimal place.

Input into MATLAB: `goldensectionsearch2(17.5,30.5,6.5,16,10-1,0.1625)`

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
9	4.81249	4.68751	21.01663	21.12233	21.18765	21.29335	169
10	4.68751	4.61027	21.12233	21.18765	21.22802	21.29335	185

Table 10: Table showing the output of my program to find d correct to one decimal place with a smaller number of inner iterations.

The number of inner iterations has been reduced significantly at only 185. This implies that the maximum and minimum value of $\phi(x)$ can still be obtained even with the precision being as large as 0.1635. Therefore we can improve on the number of inner iterations required to obtain d correct to one decimal place to 185. If we used 0.164, the program does accurately compute the value of d to one decimal place and so this x precision would not be useful to us for the accuracy we would like.

The precision of the outer iteration, i.e. for working out d , has the biggest affect on the accuracy of d . As long as the precision of the inner iteration, for x , is at least as precise as the outer iteration, then the outer iteration value will determine the accuracy of d . Adjusting the inner iteration to be more precise than the outer iteration will have no affect on the accuracy of d , but will still take longer due to it carrying out more iterations to meet the specified precision. If the outer iteration is more precise than the inner iteration, then the accuracy of the value of d will suffer since the outer iteration relies on the accuracy of the inner iteration to work effectively. If the precision is too big, the inner iteration may compare numbers that are similar in magnitude and not able to distinguish which is bigger - which is the main point of the algorithm.

The tables below shows how the accuracy of d varies as the precision of the inner iteration is increased. It also shows that the value of d does not change when the precision of the inner iteration is greater than the outer.

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
14	4.48529	3.62868	21.18765	21.19718	21.20307	21.2126	125
15	3.62868	3.62868	21.19718	21.20307	21.20671	21.2126	133

Table 11: Table showing output for input into MATLAB: goldensectionsearch2(17.5,30.5,6.5,16,10⁻²,10⁰)

$$\Rightarrow d = 21.2049$$

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
14	4.61027	4.61027	21.2126	21.22213	21.22802	21.23755	280
15	4.68751	4.61027	21.2126	21.21849	21.22213	21.22802	298

Table 12: Table showing output for input into MATLAB: goldensectionsearch2(17.5,30.5,6.5,16,10⁻²,10⁻¹)

$$\Rightarrow d = 21.2203$$

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
14	4.69878	4.68751	21.2126	21.22213	21.22802	21.23755	404
15	4.68751	4.80552	21.22213	21.22802	21.23166	21.23755	430

Table 13: Table showing output for input into MATLAB: goldensectionsearch2(17.5,30.5,6.5,16,10⁻²,10⁻²)

$$\Rightarrow d = 21.2298$$

Outer Iteration No.	$\Delta\phi(xd^*)$	$\Delta\phi(yd^*)$	ad^*	xd^*	yd^*	bd^*	Inner Iteration No.
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
14	4.70739	4.69612	21.2126	21.22213	21.22802	21.23755	559
15	4.69612	4.8092	21.22213	21.22802	21.23166	21.23755	595

Table 14: Table showing output for input into MATLAB: goldensectionsearch2(17.5,30.5,6.5,16,10⁻²,10⁻³)

$$\Rightarrow d = 21.2298$$

We can clearly see the accuracy of d increases and then remains constant after the precision of the outer and inner iteration agree.

```

1  function [Mode, IterationFinal, Accuracy] = goldensectionsearch(f, Lbound, Ubound, precision,
    minmax)
2  %Program to carry out golden section search on a function f between lower
3  %bound Lbound and upper bound Ubound. The function stops when the precision
4  %of the search is exceeded.
5  %KEY: 0 is minimum, 1 is maximum
6  syms x
7  if minmax==1
8      f(x)=f;
9  elseif minmax==0
10     f(x)=-f;
11 else
12     disp('Please enter 0 for min or 1 for max appropriately')
13     return
14 end
15 f(x)=f
16 %make x a symbolic value and ensures that f(x) can be evaluated.
17 if Lbound == Ubound
18     disp('Lower bound is equal to upper bound, please choose different initial values')
19 return
20 else
21     if Lbound < Ubound
22         a=Lbound;
23         b=Ubound;
24     else
25         a=Ubound;
26         b=Lbound;
27         disp('Lower Bound entered > Upper Bound entered, this has been corrected')
28     end
29 end
30 %Ensures that the user enters boundary values that are not the same and
31 %also ensures that the bigger number is always used correctly in the
32 %program.
33 phi=((sqrt(5)-1)/2);
34 y=(b-a)*phi+a;
35 x=a+b-y;
36 %define variables from the beginning according to question
37 Iteration = 1
38 %start count at 1
39 xx=f(x);
40 yy=f(y);
41 %define the first values of f(x) and y(x)
42 fprintf(' $ %2g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\
    \hline \n', Iteration, xx, yy, a, x, y, b)
43 while b-a>=2*precision
44     if xx>yy
45         a=a;
46         b=y;
47         y=x;
48         x=a+b-y;
49         yy=xx;
50         xx=f(x);
51 %If f(x)>f(y), then we can use the implications – that we can define a new
52 %subinterval [a,y]. We use the fact that x is already evaluated at f and
53 %use that for the new y value in our refined search. We must compute the
54 %new x value evaluated at x.
55         elseif xx<yy
56             a=x;
57             b=b;
58             x=y;
59             y=a+b-x;
60             xx=yy;
61             yy=f(y);
62 %If f(x)<f(y), then we can use the implications – that we can define a new
63 %subinterval [x,b]. We use the fact that y is already evaluated at f and
64 %use that for the new x value in our refined search. We must compute the
65 %new y value evaluated at x.
66         else
67             a=a;
68             b=y;
69             y=x;
70             x=a+b-y;
71             yy=xx;
72             xx=f(x);

```

```

73 %If  $f(x)=f(y)$ , then it doesn't matter what way we evaluate it. WLOG we let
74  $a=a$  and  $b=y$  in our interval. Then we can use the implications – that we
75 can define a new subinterval  $[a,y]$ . We use the fact that  $x$  is already
76 evaluated at  $f$  and use that for the new  $y$  value in our refined search.
77 We must compute the new  $x$  value evaluated at  $x$ .
78 end
79 Iteration=Iteration+1;
80 fprintf('$ %2g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\
    \hline \n',Iteration,xx,yy,a,x,y,b)
81 %count increases by 1
82 end
83 %this while loop will keep the golden search algorithm going until the
84 specified precision is met. It considers all different values for  $f(x)$  and
85  $f(y)$  to ensure the program works for all cases.
86 Mode =((a+b)/2);
87 IterationFinal = Iteration;
88 Accuracy= ((b-a)/2);
89 %Output Arguments
90 end

```

```

1 function [Mode, IterationFinal, Accuracy] = goldensectionsearch2(dLbound, dUbound, xLbound,
2 xUbound, dprecision, xprecision)
3 %Program to carry out golden section search on a function f between lower
4 %bound Lbound and upper bound Ubound. The function stops when the precision
5 %of the search is exceeded.
6 %KEY: 0 is minimum, 1 is maximum
7 syms x
8 syms d
9 f(x,d)= asin((- d^2 + x^2 + 576)/(48*x))
10 %make x and d a symbolic value and ensures that f(x,d) can be evaluated for all x and d.
11 if dLbound == dUbound
12     disp('d Lower bound is equal to d Upper bound, please choose different intial values')
13 return
14 else
15     if dLbound < dUbound
16         ad=dLbound;
17         bd=dUbound;
18     else
19         ad=dUbound;
20         bd=dLbound;
21     disp('d Lower Bound entered > d Upper Bound entered, this has been corrected')
22 end
23 if xLbound == xUbound
24     disp('x Lower bound is equal to x Upper bound, please choose different intial values')
25 return
26 else
27     if xLbound < xUbound
28         ax=xLbound;
29         bx=xUbound;
30     else
31         ax=xUbound;
32         bx=xLbound;
33     disp('x Lower Bound entered > x Upper Bound entered, this has been corrected')
34 end
35 end
36 %Ensures that the user enters boundary values that are not the same and
37 %also ensures that the bigger number is always used correctly in the
38 %program.
39 phi=((sqrt(5)-1)/2);
40 %define variables from the beginning according to question
41 Iteration = 1;
42 count = 1;
43 %start outer iteration count at 1
44 %start inner iteration count at 1
45
46 %%%%%fprintf('Iteration = %2g,xx = %2.6g,yy = %2.6g,a = %2.6g,x = %2.6g,y = %2.6g,b = %2.6g\n
47 ', Iteration, xx, yy, a, x, y, b)
48 yd=(bd-ad)*phi+ad;
49 xd=ad+bd-yd;
50 aaxx=ax;
51 bbxx=bx;
52 while bx-ax>=2*xprecision
53
54     %FOR xd we find phimax
55     dd=xd;
56     yx=(bx-ax)*phi+ax;
57     xx=ax+bx-yx;
58     yyx=double(f(yx,dd));
59     xxx=double(f(xx,dd));
60     if xxx>yyx
61         ax=ax;
62         bx=yx;
63         yx=xx;
64         xx=ax+bx-yx;
65         yyx=xxx;
66         xxx=double(f(xx,dd));
67
68         %If f(x,d)>f(y,d), then we can use the implications – that we can
69         %define a new
70         %subinterval [a,y]. We use the fact that x is already evaluated at f
71         %and
72         %use that for the new y value in our refined search. We must compute
73         %the
74         %new x value evaluated at x.
75     elseif xxx<yyx
76         ax=xx;

```

```

72         bx=bx;
73         xx=yx;
74         yx=ax+bx-xx;
75         xxx=yyx;
76         yyx=double(f(yx,dd));
77     %If  $f(x)<f(y)$ , then we can use the implications – that we can define a
    new
78     %subinterval  $[x,b]$ . We use the fact that  $y$  is already evaluated at  $f$ 
    and
79     %use that for the new  $x$  value in our refined search. We must compute
    the
80     %new  $y$  value evaluated at  $x$ .
81     else
82         ax=ax;
83         bx=yx;
84         yx=xx;
85         xx=ax+bx-yx;
86         yyx=xxx;
87         xxx=double(f(xx,dd));
88     %If  $f(x)=f(y)$ , then it doesn't matter what way we evaluate it. WLOG we
    let
89     %a=a and b=y in our interval. Then we can use the implications – that
    we
90     %can define a new subinterval  $[a,y]$ . We use the fact that  $x$  is already
    %evaluated at  $f$  and use that for the new  $y$  value in our refined search
    .
91
92     %We must compute the new  $x$  value evaluated at  $x$ .
93     end
94     count=count+1;
95 end
96     maxphixd=(ax+bx)/2;
97     ax=aaxx;
98     bx=bbxx;
99     f(x,d)=f(x,d);
100 while bx-ax>=2*xprecision
101     %FOR  $x_d$  we find min
102     dd=xd;
103     yx=(bx-ax)*phi+ax;
104     xx=ax+bx-yx;
105     yyx=double(f(yx,dd));
106     xxx=double(f(xx,dd));
107     if xxx>yyx
108         ax=ax;
109         bx=yx;
110         yx=xx;
111         xx=ax+bx-yx;
112         yyx=xxx;
113         xxx=double(f(xx,dd));
114     %If  $f(x,d)>f(y,d)$ , then we can use the implications – that we can
    define a new
115     %subinterval  $[a,y]$ . We use the fact that  $x$  is already evaluated at  $f$ 
    and
116     %use that for the new  $y$  value in our refined search. We must compute
    the
117     %new  $x$  value evaluated at  $x$ .
118     elseif xxx<yyx
119         ax=xx;
120         bx=bx;
121         xx=yx;
122         yx=ax+bx-xx;
123         xxx=yyx;
124         yyx=double(f(yx,dd));
125     %If  $f(x)<f(y)$ , then we can use the implications – that we can define a
    new
126     %subinterval  $[x,b]$ . We use the fact that  $y$  is already evaluated at  $f$ 
    and
127     %use that for the new  $x$  value in our refined search. We must compute
    the
128     %new  $y$  value evaluated at  $x$ .
129     else
130         ax=ax;
131         bx=yx;
132         yx=xx;
133         xx=ax+bx-yx;
134         yyx=xxx;
135         xxx=double(f(xx,dd));

```

```

136         %If  $f(x)=f(y)$ , then it doesn't matter what way we evaluate it. WLOG we
137         let
138         %a=a and b=y in our interval. Then we can use the implications – that
139         we
140         %can define a new subinterval  $[a,y]$ . We use the fact that x is already
141         %evaluated at f and use that for the new y value in our refined search
142         .
143         %We must compute the new x value evaluated at x.
144         end
145         count=count+1;
146         minphixd=(ax+bx)/2;
147         f(x,d)=f(x,d);
148         ax=aaxx;
149         bx=bbxx;
150     while bx-ax>=2*xprecision
151         %FOR yd we find phimax
152         dd=yd;
153         yx=(bx-ax)*phi+ax;
154         xx=ax+bx-yx;
155         yyx=double(f(yx,dd));
156         xxx=double(f(xx,dd));
157         if xxx>yyx
158             ax=ax;
159             bx=yx;
160             yx=xx;
161             xx=ax+bx-yx;
162             yyx=xxx;
163             xxx=double(f(xx,dd));
164         %If  $f(x,d)>f(y,d)$ , then we can use the implications – that we can
165         define a new
166         %subinterval  $[a,y]$ . We use the fact that x is already evaluated at f
167         and
168         %use that for the new y value in our refined search. We must compute
169         the
170         %new x value evaluated at x.
171         elseif xxx<yyx
172             ax=xx;
173             bx=bx;
174             xx=yx;
175             yx=ax+bx-xx;
176             xxx=yyx;
177             yyx=double(f(yx,dd));
178         %If  $f(x)<f(y)$ , then we can use the implications – that we can define a
179         new
180         %subinterval  $[x,b]$ . We use the fact that y is already evaluated at f
181         and
182         %use that for the new x value in our refined search. We must compute
183         the
184         %new y value evaluated at x.
185         else
186             ax=ax;
187             bx=yx;
188             yx=xx;
189             xx=ax+bx-yx;
190             yyx=xxx;
191             xxx=double(f(xx,dd));
192         %If  $f(x)=f(y)$ , then it doesn't matter what way we evaluate it. WLOG we
193         let
194         %a=a and b=y in our interval. Then we can use the implications – that
195         we
196         %can define a new subinterval  $[a,y]$ . We use the fact that x is already
197         %evaluated at f and use that for the new y value in our refined search
198         .
199         %We must compute the new x value evaluated at x.
200         end
201         count=count+1;
202         maxphiyd=(ax+bx)/2;
203         ax=aaxx;
204         bx=bbxx;
205         f(x,d)=f(x,d);
206     while bx-ax>=2*xprecision
207         %FOR yd we find min
208         dd=yd;
209         yx=(bx-ax)*phi+ax;

```



```

266         bx=yx;
267         yx=xx;
268         xx=ax+bx-yx;
269         yyx=xxx;
270         xxx=double(f(xx,dd));
271 %If f(x,d)>f(y,d), then we can use the implications – that we can
    define a new
272 %subinterval [a,y]. We use the fact that x is already evaluated at f
    and
273 %use that for the new y value in our refined search. We must compute
    the
274 %new x value evaluated at x.
275     elseif xxx<yyx
276         ax=xx;
277         bx=bx;
278         xx=yx;
279         yx=ax+bx-xx;
280         xxx=yyx;
281         yyx=double(f(yx,dd));
282 %If f(x)<f(y), then we can use the implications – that we can define a
    new
283 %subinterval [x,b]. We use the fact that y is already evaluated at f
    and
284 %use that for the new x value in our refined search. We must compute
    the
285 %new y value evaluated at x.
286     else
287         ax=ax;
288         bx=yx;
289         yx=xx;
290         xx=ax+bx-yx;
291         yyx=xxx;
292         xxx=double(f(xx,dd));
293 %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we
    let
294 %a=a and b=y in our interval. Then we can use the implications – that
    we
295 %can define a new subinterval [a,y]. We use the fact that x is already
    evaluated at f and use that for the new y value in our refined search
296 %
    .
297 %We must compute the new x value evaluated at x.
298     end
299     count=count+1;
300 end
301 maxphixd=(ax+bx)/2;
302 ax=aaxx;
303 bx=bbxx;
304 f(x,d)=f(x,d);
305 while bx-ax>=2*xprecision
306     %FOR xd we find min
307     dd=xd;
308     yx=(bx-ax)*phi+ax;
309     xx=ax+bx-yx;
310     yyx=double(f(yx,dd));
311     xxx=double(f(xx,dd));
312     if xxx>yyx
313         ax=ax;
314         bx=yx;
315         yx=xx;
316         xx=ax+bx-yx;
317         yyx=xxx;
318         xxx=double(f(xx,dd));
319 %If f(x,d)>f(y,d), then we can use the implications – that we can
    define a new
320 %subinterval [a,y]. We use the fact that x is already evaluated at f
    and
321 %use that for the new y value in our refined search. We must compute
    the
322 %new x value evaluated at x.
323     elseif xxx<yyx
324         ax=xx;
325         bx=bx;
326         xx=yx;
327         yx=ax+bx-xx;
328         xxx=yyx;
329         yyx=double(f(yx,dd));

```



```

330         %If  $f(x) < f(y)$ , then we can use the implications – that we can define a
331         new
332         %subinterval  $[x, b]$ . We use the fact that  $y$  is already evaluated at  $f$ 
333         and
334         %use that for the new  $x$  value in our refined search. We must compute
335         the
336         %new  $y$  value evaluated at  $x$ .
337         else
338             ax=ax;
339             bx=yx;
340             yx=xx;
341             xx=ax+bx-yx;
342             yyx=xxx;
343             xxx=double(f(xx, dd));
344 %If  $f(x) = f(y)$ , then it doesn't matter what way we evaluate it. WLOG we let
345 %a=a and b=y in our interval. Then we can use the implications – that we
346 %can define a new subinterval  $[a, y]$ . We use the fact that  $x$  is already
347 %evaluated at  $f$  and use that for the new  $y$  value in our refined search.
348 %We must compute the new  $x$  value evaluated at  $x$ .
349         end
350         count=count+1;
351     end
352     minphixd=(ax+bx)/2;
353     f(x, d)=f(x, d);
354     ax=aaxx;
355     bx=bbxx;
356     xxd=abs(maxphixd-minphixd);
357     elseif xxd>yyd
358         ad=xd;
359         bd=bd;
360         xd=yd;
361         yd=ad+bd-xd;
362         xxd=yyd;
363     while bx-ax>=2*xprecision
364         %FOR yd we find phimax
365         dd=yd;
366         yx=(bx-ax)*phi+ax;
367         xx=ax+bx-yx;
368         yyx=double(f(yx, dd));
369         xxx=double(f(xx, dd));
370     if xxx>yyx
371         ax=ax;
372         bx=yx;
373         yx=xx;
374         xx=ax+bx-yx;
375         yyx=xxx;
376         xxx=double(f(xx, dd));
377 %If  $f(x, d) > f(y, d)$ , then we can use the implications – that we can
378 define a new
379 %subinterval  $[a, y]$ . We use the fact that  $x$  is already evaluated at  $f$ 
380 and
381 %use that for the new  $y$  value in our refined search. We must compute
382 the
383 %new  $x$  value evaluated at  $x$ .
384     elseif xxx<yyx
385         ax=xx;
386         bx=bx;
387         xx=yx;
388         yx=ax+bx-xx;
389         xxx=yyx;
390         yyx=double(f(yx, dd));
391 %If  $f(x) < f(y)$ , then we can use the implications – that we can define a
392 new
393 %subinterval  $[x, b]$ . We use the fact that  $y$  is already evaluated at  $f$ 
394 and
395 %use that for the new  $x$  value in our refined search. We must compute
396 the
397 %new  $y$  value evaluated at  $x$ .
398     else
399         ax=ax;
400         bx=yx;
401         yx=xx;
402         xx=ax+bx-yx;
403         yyx=xxx;
404         xxx=double(f(xx, dd));
405 %If  $f(x) = f(y)$ , then it doesn't matter what way we evaluate it. WLOG we

```

```

397         let
398         %a=a and b=y in our interval. Then we can use the implications – that
399         we
400         %can define a new subinterval [a,y]. We use the fact that x is already
401         %evaluated at f and use that for the new y value in our refined search
402         .
403         %We must compute the new x value evaluated at x.
404         end
405         count=count+1;
406         end
407         maxphiyd=(ax+bx)/2;
408         ax=aaxx;
409         bx=bbxx;
410         f(x,d)=f(x,d);
411         while bx-ax>=2*xprecision
412             %FOR yd we find min
413             dd=yd;
414             yx=(bx-ax)*phi+ax;
415             xx=ax+bx-yx;
416             yyx=double(f(yx,dd));
417             xxx=double(f(xx,dd));
418             if xxx>yyx
419                 ax=ax;
420                 bx=yx;
421                 yx=xx;
422                 xx=ax+bx-yx;
423                 yyx=xxx;
424                 xxx=double(f(xx,dd));
425             %If f(x,d)>f(y,d), then we can use the implications – that we can
426             define a new
427             %subinterval [a,y]. We use the fact that x is already evaluated at f
428             and
429             %use that for the new y value in our refined search. We must compute
430             the
431             %new x value evaluated at x.
432             elseif xxx<yyx
433                 ax=xx;
434                 bx=bx;
435                 xx=yx;
436                 yx=ax+bx-xx;
437                 xxx=yyx;
438                 yyx=double(f(yx,dd));
439             %If f(x)<f(y), then we can use the implications – that we can define a
440             new
441             %subinterval [x,b]. We use the fact that y is already evaluated at f
442             and
443             %use that for the new x value in our refined search. We must compute
444             the
445             %new y value evaluated at x.
446             else
447                 ax=ax;
448                 bx=yx;
449                 yx=xx;
450                 xx=ax+bx-yx;
451                 yyx=xxx;
452                 xxx=double(f(xx,dd));
453             %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we
454             let
455             %a=a and b=y in our interval. Then we can use the implications – that
456             we
457             %can define a new subinterval [a,y]. We use the fact that x is already
458             %evaluated at f and use that for the new y value in our refined search
459             .
460             %We must compute the new x value evaluated at x.
461             end
462             count=count+1;
463             end
464             minphiyd=(ax+bx)/2;
465             f(x,d)=f(x,d);
466             ax=aaxx;
467             bx=bbxx;
468             yd=abs(maxphiyd-minphiyd);
469             %If f(x)<f(y), then we can use the implications – that we can define a new
470             %subinterval [x,b]. We use the fact that y is already evaluated at f and
471             %use that for the new x value in our refined search. We must compute the
472             %new y value evaluated at x.

```

```

461 else xxd=yyd;
462       ad=ad;
463       bd=yd;
464       yd=xd;
465       xd=ad+bd-yd;
466       yyd=xxd;
467
468       while bx-ax>=2*xprecision
469           %FOR xd we find phimax
470           dd=xd;
471           yx=(bx-ax)*phi+ax;
472           xx=ax+bx-yx;
473           yyx=double(f(yx,dd));
474           xxx=double(f(xx,dd));
475           if xxx>yyx
476               ax=ax;
477               bx=yx;
478               yx=xx;
479               xx=ax+bx-yx;
480               yyx=xxx;
481               xxx=double(f(xx,dd));
482           %If f(x,d)>f(y,d), then we can use the implications – that we can
483           %define a new
484           %subinterval [a,y]. We use the fact that x is already evaluated at f
485           %and
486           %use that for the new y value in our refined search. We must compute
487           %the
488           %new x value evaluated at x.
489           elseif xxx<yyx
490               ax=xx;
491               bx=bx;
492               xx=yx;
493               yx=ax+bx-xx;
494               xxx=yyx;
495               yyx=double(f(yx,dd));
496           %If f(x)<f(y), then we can use the implications – that we can define a
497           %new
498           %subinterval [x,b]. We use the fact that y is already evaluated at f
499           %and
500           %use that for the new x value in our refined search. We must compute
501           %the
502           %new y value evaluated at x.
503           else
504               ax=ax;
505               bx=yx;
506               yx=xx;
507               xx=ax+bx-yx;
508               yyx=xxx;
509               xxx=double(f(xx,dd));
510           end
511       count=count+1;
512   end
513   maxphixd=(ax+bx)/2;
514   ax=aaxx;
515   bx=bbxx;
516   f(x,d)=f(x,d);
517   while bx-ax>=2*xprecision
518       %FOR xd we find min
519       dd=xd;
520       yx=(bx-ax)*phi+ax;
521       xx=ax+bx-yx;
522       yyx=double(f(yx,dd));
523       xxx=double(f(xx,dd));
524       if xxx>yyx
525           ax=ax;
526           bx=yx;
527           yx=xx;
528           xx=ax+bx-yx;
529           yyx=xxx;
530           xxx=double(f(xx,dd));
531       %If f(x,d)>f(y,d), then we can use the implications – that we can
532       %define a new
533       %subinterval [a,y]. We use the fact that x is already evaluated at f
534       %and
535       %use that for the new y value in our refined search. We must compute
536       %the

```



```

1  function [Delta , Dxaxis] = graphfordeltaphi(dLbound,dUbound,xLbound,xUbound,increment ,
    xprecision)
2  %Program to carry out golden section search on a function f between lower
3  %bound Lbound and upper bound Ubound. The function stops when the precision
4  %of the search is exceeded.
5  %KEY: 0 is minimum, 1 is maximum
6  syms x
7  syms d
8  f(x,d)= asin((- d^2 + x^2 + 576)/(48*x))
9  %make x and d a symbolic value and ensures that f(x,d) can be evaluated for all x and d.
10 if dLbound == dUbound
11     disp('d Lower bound is equal to d Upper bound, please choose different intial values')
12 return
13 else
14     if dLbound < dUbound
15         ad=dLbound;
16         bd=dUbound;
17     else
18         ad=dUbound;
19         bd=dLbound;
20     disp('d Lower Bound entered > d Upper Bound entered, this has been corrected')
21 end
22 end
23 if xLbound == xUbound
24     disp('x Lower bound is equal to x Upper bound, please choose different intial values')
25 return
26 else
27     if xLbound < xUbound
28         ax=xLbound;
29         bx=xUbound;
30     else
31         ax=xUbound;
32         bx=xLbound;
33     disp('x Lower Bound entered > x Upper Bound entered, this has been corrected')
34 end
35 end
36 %Ensures that the user enters boundary values that are not the same and
37 %also ensures that the bigger number is always used correctly in the
38 %program.
39 phi=((sqrt(5)-1)/2);
40 %define variables from the beginning according to question
41 Iteration = 1;
42 count=1;
43 %start outer iteration count at 1
44 %start inner iteration count at 1
45
46 %%%%%fprintf('Iteration = %2g,xx = %2.6g,yy = %2.6g,a = %2.6g,x = %2.6g,y = %2.6g,b = %2.6g\n
    ',Iteration,xx,yy,a,x,y,b)
47 yd=(bd-ad)*phi+ad;
48 xd=ad+bd-yd;
49 aaxx=ax;
50 bbxx=bx;
51 m=17.5;
52 Delta=zeros(1,floor((bd-ad)/increment));
53 Dxaxis=zeros(1,floor((bd-ad)/increment));
54 for m=ad:increment:bd
55     xd=m;
56     yd=m;
57 while bx-ax>=2*xprecision
58
59     %FOR xd we find phimax
60     dd=xd;
61     yx=(bx-ax)*phi+ax;
62     xx=ax+bx-yx;
63     yyx=double(f(yx,dd));
64     xxx=double(f(xx,dd));
65     if xxx>yyx
66         ax=ax;
67         bx=yx;
68         yx=xx;
69         xx=ax+bx-yx;
70         yyx=xxx;
71         xxx=double(f(xx,dd));
72 %If f(x,d)>f(y,d), then we can use the implications - that we can
        define a new
        %subinterval [a,y]. We use the fact that x is already evaluated at f
        and

```

```

73         %use that for the new y value in our refined search. We must compute
74         the
75         %new x value evaluated at x.
76         elseif xxx<yyx
77             ax=xx;
78             bx=bx;
79             xx=yx;
80             yx=ax+bx-xx;
81             xxx=yyx;
82             yyx=double(f(yx,dd));
83         %If f(x)<f(y), then we can use the implications – that we can define a
84         new
85         %subinterval [x,b]. We use the fact that y is already evaluated at f
86         and
87         %use that for the new x value in our refined search. We must compute
88         the
89         %new y value evaluated at x.
90         else
91             ax=ax;
92             bx=yx;
93             yx=xx;
94             xx=ax+bx-yx;
95             yyx=xxx;
96             xxx=double(f(xx,dd));
97         %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we
98         let
99         %a=a and b=y in our interval. Then we can use the implications – that
100         we
101         %can define a new subinterval [a,y]. We use the fact that x is already
102         %evaluated at f and use that for the new y value in our refined search
103         .
104         %We must compute the new x value evaluated at x.
105         end
106         count=count+1;
107
108         maxphixd=(ax+bx)/2;
109         ax=aaxx;
110         bx=bbxx;
111         f(x,d)=f(x,d);
112
113     while bx-ax>=2*xprecision
114         %FOR xd we find min
115         dd=xd;
116         yx=(bx-ax)*phi+ax;
117         xx=ax+bx-yx;
118         yyx=double(f(yx,dd));
119         xxx=double(f(xx,dd));
120         if xxx>yyx
121             ax=ax;
122             bx=yx;
123             yx=xx;
124             xx=ax+bx-yx;
125             yyx=xxx;
126             xxx=double(f(xx,dd));
127         %If f(x,d)>f(y,d), then we can use the implications – that we can
128         define a new
129         %subinterval [a,y]. We use the fact that x is already evaluated at f
130         and
131         %use that for the new y value in our refined search. We must compute
132         the
133         %new x value evaluated at x.
134         elseif xxx<yyx
135             ax=xx;
136             bx=bx;
137             xx=yx;
138             yx=ax+bx-xx;
139             xxx=yyx;
140             yyx=double(f(yx,dd));
141         %If f(x)<f(y), then we can use the implications – that we can define a
142         new
143         %subinterval [x,b]. We use the fact that y is already evaluated at f
144         and
145         %use that for the new x value in our refined search. We must compute
146         the
147         %new y value evaluated at x.
148         else
149             ax=ax;

```

