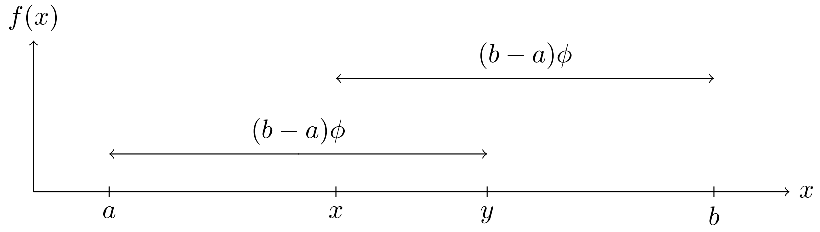
Project: 1.1 Golden Section Search for the Mode of a Function

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,

$$y = (b-a)\phi + a \tag{1}$$
$$x - a = b - y \tag{2}$$



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($5 - x^2$,0,1,1×10^{-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($f, 0, 1, 1 \times 10^{-5}$,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
⋮	⋮	⋮	⋮	⋮	⋮	⋮
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 uses 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 is 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	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$
24	-7.48811×10^{-41}	-7.22247×10^{-46}	-2.15666×10^{-5}	-9.64487×10^{-6}	-2.27686×10^{-6}	$\cdots$
25	-7.22247×10^{-46}	-7.22247×10^{-46}	-9.64487×10^{-6}	-2.27686×10^{-6}	2.27686×10^{-6}	$\cdots$

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	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$
24	-9.30236×10^{-11}	-5.18408×10^{-12}	-2.15666×10^{-5}	-9.64487×10^{-6}	-2.27686×10^{-6}	$\cdots$
25	-5.18408×10^{-12}	-5.18408×10^{-12}	-9.64487×10^{-6}	-2.27686×10^{-6}	2.27686×10^{-6}	$\cdots$

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).

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

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

$$\begin{aligned} -1 &\leq \frac{x^2 + 24^2 - d^2}{48x} & \qquad \frac{x^2 + 24^2 - d^2}{48x} &\leq 1 \\ 0 \leq d^2 &\leq (x + 24)^2 & 0 \leq (24 - x)^2 &\leq d^2 \\ 0 \leq d &\leq \lvert x + 24 \rvert & 0 \leq \lvert 24 - x \rvert &\leq d \end{aligned} \tag{4}$$

   Since x represents the radius at which the stylus is at, x must vary from $r = 6.5$ to $R = 16$. Both $\lvert x+24 \rvert$ and $\lvert 24-x \rvert$ 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).

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

   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
goldensectionsearch2(LB of d, UB of d, LB of x, UB of x, Precision of d, Precision of x)
*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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
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.

$$\text{Therefore the optimum value of } d = 21.22957. \tag{5}$$

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 minmised. 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)

	Iteration	xx	yy	a	x	y	b
Maximum	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	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 $x \in [6.5, 16]$.

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

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.

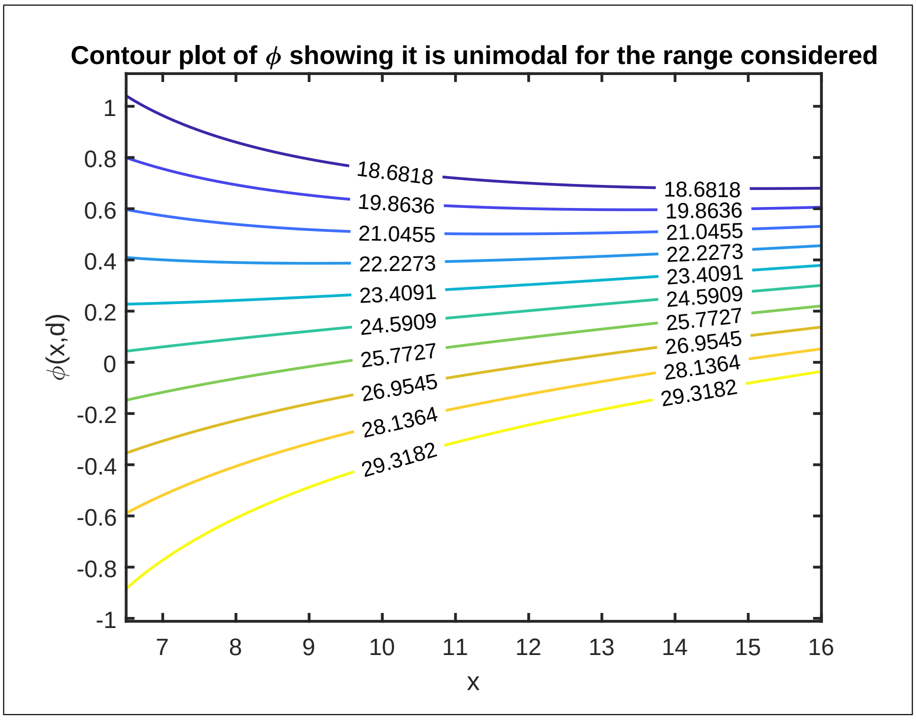


Figure 1: Graph showing a contour plot of $\phi(x, d)$ for different values of $d \in [17.5, 30.5]$

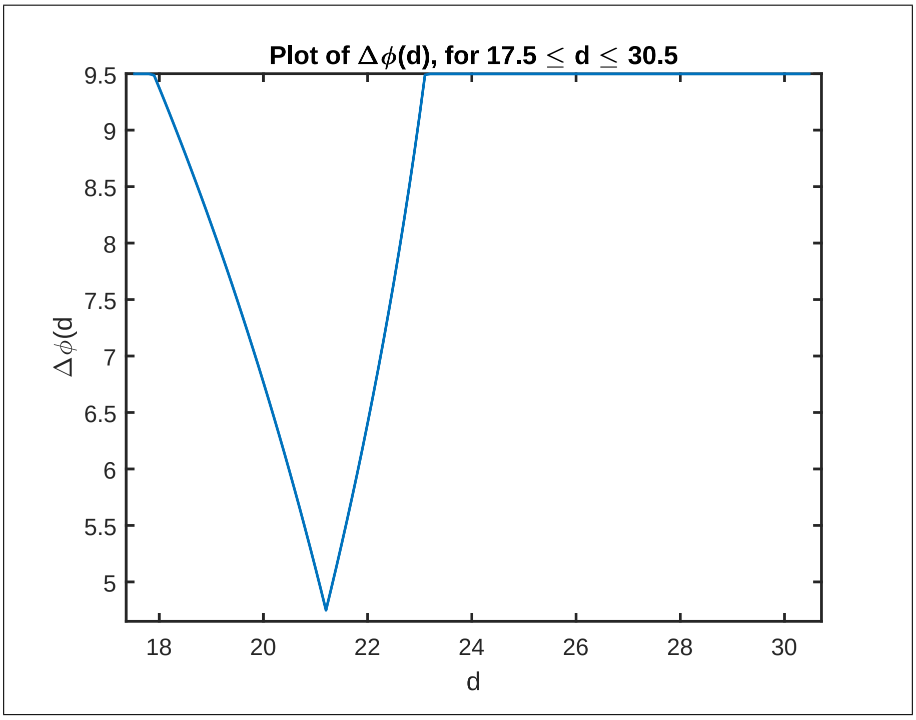


Figure 2: Graph showing a contour plot of $\phi(x, d)$ for different values of $d \in [17.5, 30.5]$

(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, $d = \dfrac{21.29335\ldots + 21.12233\ldots}{2} = 21.2078\ldots = 21.2$ (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},10^{0})

$$\Rightarrow d = 21.2049$$

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.

```
function [Mode, IterationFinal,Accuracy] = goldensectionsearch(f,Lbound,Ubound,precision,
    minmax)
%Program to carry out golden section search on a function f between lower
%bound Lbound and upper bound Ubound. The function stops when the precision
%of the search is exceeded.
%KEY: 0 is minimum, 1 is maximum
syms x
if minmax==1
    f(x)=f;
elseif minmax==0
    f(x)=-f;
else
    disp('Please enter 0 for min or 1 for max appropriately')
    return
end
f(x)=f
%make x a symbolic value and ensures that f(x) can be evaluated.
if Lbound == Ubound
    disp('Lower bound is equal to upper bound, please choose different intial values')
return
else
    if Lbound < Ubound
        a=Lbound;
        b=Ubound;
    else
        a=Ubound;
        b=Lbound;
        disp('Lower Bound entered > Upper Bound entered, this has been corrected')
    end
end
%Ensures that the user enters boundary values that are not the same and
%also ensures that the bigger number is always used correctly in the
%program.
phi=((sqrt(5)-1)/2);
y=(b-a)*phi+a;
x=a+b-y;
%define variables from the beginning according to question
Iteration = 1
%start count at 1
xx=f(x);
yy=f(y);
%define the first values of f(x) and y(x)
fprintf('$ %2g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\
    hline \n',Iteration,xx,yy,a,x,y,b)
while b-a>=2*precision
    if xx>yy
        a=a;
        b=y;
        y=x;
        x=a+b-y;
        yy=xx;
        xx=f(x);
%If f(x)>f(y), 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
%use that for the new y value in our refined search. We must compute the
%new x value evaluated at x.
    elseif xx<yy
        a=x;
        b=b;
        x=y;
        y=a+b-x;
        xx=yy;
        yy=f(y);
%If f(x)<f(y), then we can use the implications - that we can define a new
%subinterval [x,b]. We use the fact that y is already evaluated at f and
%use that for the new x value in our refined search. We must compute the
%new y value evaluated at x.
    else
        a=a;
        b=y;
        y=x;
        x=a+b-y;
        yy=xx;
        xx=f(x);
```

```
%If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
%a=a and b=y in our interval. 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 use that for the new y value in our refined search.
%We must compute the new x value evaluated at x.
    end
    Iteration=Iteration+1;
fprintf('$ %2g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\
    hline \n',Iteration,xx,yy,a,x,y,b)
%count increases by 1
end
%this while loop will keep the golden search algorithm going until the
%specified precision is met. It considers all different values for f(x) and
%f(y) to ensure the program works for all cases.
Mode =((a+b)/2);
IterationFinal = Iteration;
Accuracy= ((b-a)/2);
%Output Arguments
end
```

```
function [Mode, IterationFinal,Accuracy] = goldensectionsearch2(dLbound,dUbound,xLbound,xUbound,dprecision,xprecision)
%Program to carry out golden section search on a function f between lower
%bound Lbound and upper bound Ubound. The function stops when the precision
%of the search is exceeded.
%KEY: 0 is minimum, 1 is maximum
syms x
syms d
f(x,d)= asin((- d^2 + x^2 + 576)/(48*x))
%make x and d a symbolic value and ensures that f(x,d) can be evaluated for all x and d.
if dLbound == dUbound
    disp('d Lower bound is equal to d Upper bound, please choose different intial values')
return
else
    if dLbound < dUbound
        ad=dLbound;
        bd=dUbound;
    else
        ad=dUbound;
        bd=dLbound;
        disp('d Lower Bound entered > d Upper Bound entered, this has been corrected')
    end
end
if xLbound == xUbound
    disp('x Lower bound is equal to x Upper bound, please choose different intial values')
return
else
    if xLbound < xUbound
        ax=xLbound;
        bx=xUbound;
    else
        ax=xUbound;
        bx=xLbound;
        disp('x Lower Bound entered > x Upper Bound entered, this has been corrected')
    end
end
%Ensures that the user enters boundary values that are not the same and
%also ensures that the bigger number is always used correctly in the
%program.
phi=((sqrt(5)-1)/2);
%define variables from the beginning according to question
Iteration = 1;
count = 1;
%start outer iteration count at 1
%start inner iteration count at 1

%%%%%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)
    yd=(bd-ad)*phi+ad;
    xd=ad+bd-yd;
aaxx=ax;
bbxx=bx;
while bx-ax>=2*xprecision

                                %FOR xd we find phimax
                                dd=xd;
                                yx=(bx-ax)*phi+ax;
                                xx=ax+bx-yx;
                                yyx=double(f(yx,dd));
                                xxx=double(f(xx,dd));
                            if xxx>yyx
                                 ax=ax;
                                 bx=yx;
                                 yx=xx;
                                 xx=ax+bx-yx;
                                 yyx=xxx;
                                 xxx=double(f(xx,dd));
                        %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
                        %use that for the new y value in our refined search. We must compute the
                        %new x value evaluated at x.
                            elseif xxx<yyx
                                 ax=xx;
```

```
                    bx=bx;
                    xx=yx;
                    yx=ax+bx-xx;
                    xxx=yyx;
                    yyx=double(f(yx,dd));
            %If f(x)<f(y), then we can use the implications - that we can define a new
            %subinterval [x,b]. We use the fact that y is already evaluated at f and
            %use that for the new x value in our refined search. We must compute the
            %new y value evaluated at x.
                else
                    ax=ax;
                    bx=yx;
                    yx=xx;
                    xx=ax+bx-yx;
                    yyx=xxx;
                    xxx=double(f(xx,dd));
            %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
            %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
            %We must compute the new x value evaluated at x.
                end
            count=count+1;
end
              maxphixd=(ax+bx)/2;
              ax=aaxx;
              bx=bbxx;
              f(x,d)=-f(x,d);
while bx-ax>=2*xprecision
               %FOR xd we find min
                  dd=xd;
                  yx=(bx-ax)*phi+ax;
                  xx=ax+bx-yx;
                  yyx=double(f(yx,dd));
                  xxx=double(f(xx,dd));
              if xxx>yyx
                    ax=ax;
                    bx=yx;
                    yx=xx;
                    xx=ax+bx-yx;
                    yyx=xxx;
                    xxx=double(f(xx,dd));
            %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
            %use that for the new y value in our refined search. We must compute the
            %new x value evaluated at x.
                elseif xxx<yyx
                    ax=xx;
                    bx=bx;
                    xx=yx;
                    yx=ax+bx-xx;
                    xxx=yyx;
                    yyx=double(f(yx,dd));
            %If f(x)<f(y), then we can use the implications - that we can define a new
            %subinterval [x,b]. We use the fact that y is already evaluated at f and
            %use that for the new x value in our refined search. We must compute the
            %new y value evaluated at x.
                else
                    ax=ax;
                    bx=yx;
                    yx=xx;
                    xx=ax+bx-yx;
                    yyx=xxx;
                    xxx=double(f(xx,dd));
```

```
                        %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
                        %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                        %We must compute the new x value evaluated at x.
                           end
                            count=count+1;
end
                        minphixd=(ax+bx)/2;
                        f(x,d)=-f(x,d);
                           ax=aaxx;
                           bx=bbxx;
while bx-ax>=2*xprecision
                               %FOR yd we find phimax
                               dd=yd;
                               yx=(bx-ax)*phi+ax;
                               xx=ax+bx-yx;
                               yyx=double(f(yx,dd));
                               xxx=double(f(xx,dd));
                            if xxx>yyx
                                ax=ax;
                                bx=yx;
                                yx=xx;
                                xx=ax+bx-yx;
                                yyx=xxx;
                                xxx=double(f(xx,dd));
                        %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
                        %use that for the new y value in our refined search. We must compute the
                        %new x value evaluated at x.
                            elseif xxx<yyx
                                ax=xx;
                                bx=bx;
                                xx=yx;
                                yx=ax+bx-xx;
                                xxx=yyx;
                                yyx=double(f(yx,dd));
                        %If f(x)<f(y), then we can use the implications - that we can define a new
                        %subinterval [x,b]. We use the fact that y is already evaluated at f and
                        %use that for the new x value in our refined search. We must compute the
                        %new y value evaluated at x.
                            else
                                ax=ax;
                                bx=yx;
                                yx=xx;
                                xx=ax+bx-yx;
                                yyx=xxx;
                                xxx=double(f(xx,dd));
                        %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
                        %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                        %We must compute the new x value evaluated at x.
                            end
                            count=count+1;
end
                           maxphiyd=(ax+bx)/2;
                           ax=aaxx;
                           bx=bbxx;
                           f(x,d)=-f(x,d);
while bx-ax>=2*xprecision
                            %FOR yd we find min
                               dd=yd;
                               yx=(bx-ax)*phi+ax;
```

```
                                        xx=ax+bx-yx;
                                        yyx=double(f(yx,dd));
                                        xxx=double(f(xx,dd));
                                    if xxx>yyx
                                         ax=ax;
                                         bx=yx;
                                         yx=xx;
                                         xx=ax+bx-yx;
                                         yyx=xxx;
                                         xxx=double(f(xx,dd));
                                         count=count+1;
                                %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
                                %use that for the new y value in our refined search. We must compute the
                                %new x value evaluated at x.
                                     elseif xxx<yyx
                                         ax=xx;
                                         bx=bx;
                                         xx=yx;
                                         yx=ax+bx-xx;
                                         xxx=yyx;
                                         yyx=double(f(yx,dd));
                                %If f(x)<f(y), then we can use the implications - that we can define a new
                                %subinterval [x,b]. We use the fact that y is already evaluated at f and
                                %use that for the new x value in our refined search. We must compute the
                                %new y value evaluated at x.
                                     else
                                         ax=ax;
                                         bx=yx;
                                         yx=xx;
                                         xx=ax+bx-yx;
                                         yyx=xxx;
                                         xxx=double(f(xx,dd));
                                %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
                                %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                                %We must compute the new x value evaluated at x.
                                    end
                                     count=count+1;
end
                                minphiyd=(ax+bx)/2;
                                f(x,d)=-f(x,d);
                                ax=aaxx;
                                bx=bbxx;
xxd=abs(maxphixd-minphixd);
yyd=abs(maxphiyd-minphiyd);
fprintf('$ %2g $ & $ %2.6g $ & $ %2.6g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\ hline \n',Iteration,xxd,yyd,ad,xd,yd,bd,count)
while bd-ad>=2*dprecision

   if xxd<yyd
        ad=ad;
        bd=yd;
        yd=xd;
        xd=ad+bd-yd;
        yyd=xxd;

                                     while bx-ax>=2*xprecision
                                        %FOR xd we find phimax
                                        dd=xd;
                                        yx=(bx-ax)*phi+ax;
                                        xx=ax+bx-yx;
                                        yyx=double(f(yx,dd));
                                        xxx=double(f(xx,dd));
                                     if xxx>yyx
                                         ax=ax;
```

```
                bx=yx;
                yx=xx;
                xx=ax+bx-yx;
                yyx=xxx;
                xxx=double(f(xx,dd));
        %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
        %use that for the new y value in our refined search. We must compute the
        %new x value evaluated at x.
            elseif xxx<yyx
                ax=xx;
                bx=bx;
                xx=yx;
                yx=ax+bx-xx;
                xxx=yyx;
                yyx=double(f(yx,dd));
        %If f(x)<f(y), then we can use the implications - that we can define a new
        %subinterval [x,b]. We use the fact that y is already evaluated at f and
        %use that for the new x value in our refined search. We must compute the
        %new y value evaluated at x.
            else
                ax=ax;
                bx=yx;
                yx=xx;
                xx=ax+bx-yx;
                yyx=xxx;
                xxx=double(f(xx,dd));
        %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
        %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
        %We must compute the new x value evaluated at x.
            end
            count=count+1;
        end
       maxphixd=(ax+bx)/2;
        ax=aaxx;
        bx=bbxx;
        f(x,d)=-f(x,d);
        while bx-ax>=2*xprecision
            %FOR xd we find min
              dd=xd;
              yx=(bx-ax)*phi+ax;
              xx=ax+bx-yx;
              yyx=double(f(yx,dd));
              xxx=double(f(xx,dd));
            if xxx>yyx
                ax=ax;
                bx=yx;
                yx=xx;
                xx=ax+bx-yx;
                yyx=xxx;
                xxx=double(f(xx,dd));
        %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
        %use that for the new y value in our refined search. We must compute the
        %new x value evaluated at x.
            elseif xxx<yyx
                ax=xx;
                bx=bx;
                xx=yx;
                yx=ax+bx-xx;
                xxx=yyx;
                yyx=double(f(yx,dd));
```

```
                        %If f(x)<f(y), then we can use the implications - that we can define a new
                        %subinterval [x,b]. We use the fact that y is already evaluated at f and
                        %use that for the new x value in our refined search. We must compute the
                        %new y value evaluated at x.
                            else
                                ax=ax;
                                bx=yx;
                                yx=xx;
                                xx=ax+bx-yx;
                                yyx=xxx;
                                xxx=double(f(xx,dd));
%If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
%a=a and b=y in our interval. 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 use that for the new y value in our refined search.
%We must compute the new x value evaluated at x.
                           end
                            count=count+1;
                        end
                        minphixd=(ax+bx)/2;
                        f(x,d)=-f(x,d);
                        ax=aaxx;
                        bx=bbxx;
xxd=abs(maxphixd-minphixd);
    elseif xxd>yyd
        ad=xd;
        bd=bd;
        xd=yd;
        yd=ad+bd-xd;
        xxd=yyd;
                            while bx-ax>=2*xprecision
                               %FOR yd we find phimax
                               dd=yd;
                               yx=(bx-ax)*phi+ax;
                               xx=ax+bx-yx;
                               yyx=double(f(yx,dd));
                               xxx=double(f(xx,dd));
                            if xxx>yyx
                                ax=ax;
                                bx=yx;
                                yx=xx;
                                xx=ax+bx-yx;
                                yyx=xxx;
                                xxx=double(f(xx,dd));
                        %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
                        %use that for the new y value in our refined search. We must compute the
                        %new x value evaluated at x.
                            elseif xxx<yyx
                                ax=xx;
                                bx=bx;
                                xx=yx;
                                yx=ax+bx-xx;
                                xxx=yyx;
                                yyx=double(f(yx,dd));
                        %If f(x)<f(y), then we can use the implications - that we can define a new
                        %subinterval [x,b]. We use the fact that y is already evaluated at f and
                        %use that for the new x value in our refined search. We must compute the
                        %new y value evaluated at x.
                            else
                                ax=ax;
                                bx=yx;
                                yx=xx;
                                xx=ax+bx-yx;
                                yyx=xxx;
                                xxx=double(f(xx,dd));
                        %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we
```

```
let
                %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                %We must compute the new x value evaluated at x.
                    end
                    count=count+1;
                end
                maxphiyd=(ax+bx)/2;
                ax=aaxx;
                bx=bbxx;
                f(x,d)=-f(x,d);
            while bx-ax>=2*xprecision
                %FOR yd we find min
                    dd=yd;
                    yx=(bx-ax)*phi+ax;
                    xx=ax+bx-yx;
                    yyx=double(f(yx,dd));
                    xxx=double(f(xx,dd));
                if xxx>yyx
                        ax=ax;
                        bx=yx;
                        yx=xx;
                        xx=ax+bx-yx;
                        yyx=xxx;
                        xxx=double(f(xx,dd));
                %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
                %use that for the new y value in our refined search. We must compute the
                %new x value evaluated at x.
                    elseif xxx<yyx
                        ax=xx;
                        bx=bx;
                        xx=yx;
                        yx=ax+bx-xx;
                        xxx=yyx;
                        yyx=double(f(yx,dd));
                %If f(x)<f(y), then we can use the implications - that we can define a new
                %subinterval [x,b]. We use the fact that y is already evaluated at f and
                %use that for the new x value in our refined search. We must compute the
                %new y value evaluated at x.
                    else
                        ax=ax;
                        bx=yx;
                        yx=xx;
                        xx=ax+bx-yx;
                        yyx=xxx;
                        xxx=double(f(xx,dd));
                %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
                %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                %We must compute the new x value evaluated at x.
                end
                    count=count+1;
            end
            minphiyd=(ax+bx)/2;
            f(x,d)=-f(x,d);
            ax=aaxx;
            bx=bbxx;
yyd=abs(maxphiyd-minphiyd);
%If f(x)<f(y), then we can use the implications - that we can define a new
%subinterval [x,b]. We use the fact that y is already evaluated at f and
%use that for the new x value in our refined search. We must compute the
%new y value evaluated at x.
```

```
    else xxd=yyd;
         ad=ad;
         bd=yd;
         yd=xd;
         xd=ad+bd-yd;
         yyd=xxd;
                          while bx-ax>=2*xprecision
                             %FOR xd we find phimax
                             dd=xd;
                             yx=(bx-ax)*phi+ax;
                             xx=ax+bx-yx;
                             yyx=double(f(yx,dd));
                             xxx=double(f(xx,dd));
                          if xxx>yyx
                              ax=ax;
                              bx=yx;
                              yx=xx;
                              xx=ax+bx-yx;
                              yyx=xxx;
                              xxx=double(f(xx,dd));
                      %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
                      %use that for the new y value in our refined search. We must compute
                          the
                      %new x value evaluated at x.
                          elseif xxx<yyx
                              ax=xx;
                              bx=bx;
                              xx=yx;
                              yx=ax+bx-xx;
                              xxx=yyx;
                              yyx=double(f(yx,dd));
                      %If f(x)<f(y), then we can use the implications - that we can define a
                           new
                      %subinterval [x,b]. We use the fact that y is already evaluated at f
                          and
                      %use that for the new x value in our refined search. We must compute
                          the
                      %new y value evaluated at x.
                          else
                              ax=ax;
                              bx=yx;
                              yx=xx;
                              xx=ax+bx-yx;
                              yyx=xxx;
                              xxx=double(f(xx,dd));
                          end
                          count=count+1;
                      end
                         maxphixd=(ax+bx)/2;
                          ax=aaxx;
                          bx=bbxx;
                          f(x,d)=-f(x,d);
                      while bx-ax>=2*xprecision
                          %FOR xd we find min

                             dd=xd;
                             yx=(bx-ax)*phi+ax;
                             xx=ax+bx-yx;
                             yyx=double(f(yx,dd));
                             xxx=double(f(xx,dd));
                         if xxx>yyx
                              ax=ax;
                              bx=yx;
                              yx=xx;
                              xx=ax+bx-yx;
                              yyx=xxx;
                              xxx=double(f(xx,dd));
                      %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
                      %use that for the new y value in our refined search. We must compute
                          the
```

```
                %new x value evaluated at x.
                    elseif xxx<yyx
                        ax=xx;
                        bx=bx;
                        xx=yx;
                        yx=ax+bx-xx;
                        xxx=yyx;
                        yyx=double(f(yx,dd));
                %If f(x)<f(y), then we can use the implications - that we can define a new
                %subinterval [x,b]. We use the fact that y is already evaluated at f and
                %use that for the new x value in our refined search. We must compute the
                %new y value evaluated at x.
                    else
                        ax=ax;
                        bx=yx;
                        yx=xx;
                        xx=ax+bx-yx;
                        yyx=xxx;
                        xxx=double(f(xx,dd));
%If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
 %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
%We must compute the new x value evaluated at x.
                   end
                    count=count+1;
                end
                minphixd=(ax+bx)/2;
                f(x,d)=-f(x,d);
                ax=aaxx;
                bx=bbxx;
xxd=abs(maxphixd-minphixd);
end
    Iteration=Iteration+1;
fprintf('$ %2g $ & $ %2.6g $ & $ %2.6g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\ hline \n',Iteration,xxd,yyd,ad,xd,yd,bd,count)
%count increases by 1
end
%this while loop will keep the golden search algorithm going until the
%specified precision is met. It considers all different values for f(x) and
%f(y) to ensure the program works for all cases.
Mode =((ad+bd)/2)
IterationFinal = Iteration;
Accuracy= ((bd-ad)/2);
%Output Arguments
end
```

```
function [Delta, Dxaxis] = graphfordeltaphi(dLbound,dUbound,xLbound,xUbound,increment,xprecision)
%Program to carry out golden section search on a function f between lower
%bound Lbound and upper bound Ubound. The function stops when the precision
%of the search is exceeded.
%KEY: 0 is minimum, 1 is maximum
syms x
syms d
f(x,d)= asin((- d^2 + x^2 + 576)/(48*x))
%make x and d a symbolic value and ensures that f(x,d) can be evaluated for all x and d.
if dLbound == dUbound
    disp('d Lower bound is equal to d Upper bound, please choose different intial values')
return
else
    if dLbound < dUbound
        ad=dLbound;
        bd=dUbound;
    else
        ad=dUbound;
        bd=dLbound;
        disp('d Lower Bound entered > d Upper Bound entered, this has been corrected')
    end
end
if xLbound == xUbound
    disp('x Lower bound is equal to x Upper bound, please choose different intial values')
return
else
    if xLbound < xUbound
        ax=xLbound;
        bx=xUbound;
    else
        ax=xUbound;
        bx=xLbound;
        disp('x Lower Bound entered > x Upper Bound entered, this has been corrected')
    end
end
%Ensures that the user enters boundary values that are not the same and
%also ensures that the bigger number is always used correctly in the
%program.
phi=((sqrt(5)-1)/2);
%define variables from the beginning according to question
Iteration = 1;
count=1;
%start outer iteration count at 1
%start inner iteration count at 1

%%%%%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)
    yd=(bd-ad)*phi+ad;
    xd=ad+bd-yd;
aaxx=ax;
bbxx=bx;
m=17.5;
Delta=zeros(1,floor((bd-ad)/increment));
Dxaxis=zeros(1,floor((bd-ad)/increment));
for m=ad:increment:bd
    xd=m;
    yd=m;
while bx-ax>=2*xprecision
                                %FOR xd we find phimax
                                dd=xd;
                                yx=(bx-ax)*phi+ax;
                                xx=ax+bx-yx;
                                yyx=double(f(yx,dd));
                                xxx=double(f(xx,dd));
                             if xxx>yyx
                                 ax=ax;
                                 bx=yx;
                                 yx=xx;
                                 xx=ax+bx-yx;
                                 yyx=xxx;
                                 xxx=double(f(xx,dd));
                        %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
```

```
                    %use that for the new y value in our refined search. We must compute the
                    %new x value evaluated at x.
                        elseif xxx<yyx
                            ax=xx;
                            bx=bx;
                            xx=yx;
                            yx=ax+bx-xx;
                            xxx=yyx;
                            yyx=double(f(yx,dd));
                    %If f(x)<f(y), then we can use the implications - that we can define a new
                    %subinterval [x,b]. We use the fact that y is already evaluated at f and
                    %use that for the new x value in our refined search. We must compute the
                    %new y value evaluated at x.
                        else
                            ax=ax;
                            bx=yx;
                            yx=xx;
                            xx=ax+bx-yx;
                            yyx=xxx;
                            xxx=double(f(xx,dd));
                    %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
                    %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                    %We must compute the new x value evaluated at x.
                        end
                    count=count+1;
end
                        maxphixd=(ax+bx)/2;
                        ax=aaxx;
                        bx=bbxx;
                        f(x,d)=-f(x,d);
while bx-ax>=2*xprecision
                        %FOR xd we find min
                            dd=xd;
                            yx=(bx-ax)*phi+ax;
                            xx=ax+bx-yx;
                            yyx=double(f(yx,dd));
                            xxx=double(f(xx,dd));
                        if xxx>yyx
                            ax=ax;
                            bx=yx;
                            yx=xx;
                            xx=ax+bx-yx;
                            yyx=xxx;
                            xxx=double(f(xx,dd));
                    %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
                    %use that for the new y value in our refined search. We must compute the
                    %new x value evaluated at x.
                        elseif xxx<yyx
                            ax=xx;
                            bx=bx;
                            xx=yx;
                            yx=ax+bx-xx;
                            xxx=yyx;
                            yyx=double(f(yx,dd));
                    %If f(x)<f(y), then we can use the implications - that we can define a new
                    %subinterval [x,b]. We use the fact that y is already evaluated at f and
                    %use that for the new x value in our refined search. We must compute the
                    %new y value evaluated at x.
                        else
                            ax=ax;
```

```
                                        bx=yx;
                                        yx=xx;
                                        xx=ax+bx-yx;
                                        yyx=xxx;
                                        xxx=double(f(xx,dd));
                                %If f(x)=f(y), then it doesn't matter what way we evaluate it. WLOG we let
                                %a=a and b=y in our interval. 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 use that for the new y value in our refined search.
                                %We must compute the new x value evaluated at x.
                                    end
                                    count=count+1;
end
                                minphixd=(ax+bx)/2;
                                f(x,d)=-f(x,d);
                                ax=aaxx;
                                bx=bbxx;
xxd=abs(maxphixd-minphixd);

Delta(1,Iteration)=xxd;
Dxaxis(1,Iteration)=m;
m=m+increment;
Iteration=Iteration+1;
end
%fprintf('$ %2g $ & $ %2.6g $ & $ %2.6g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ & $ %2.7g $ \\ hline \n',Iteration,xxd,yyd,ad,xd,yd,bd,count)
%count increases by 1
%this while loop will keep the golden search algorithm going until the
%specified precision is met. It considers all different values for f(x) and
%f(y) to ensure the program works for all cases.
Delta
Dxaxis
%Output Arguments
end
```