1 Example of a Golden Section Search

The sequencing of the Golden Section Search is illustrated by showing the first four evaluations of the merit function for an actual example. We choose to minimize a very simple merit function whose minimum is known in advance for our illustration. This enables us to confirm that the Golden Section procedure is correctly reducing the interval of uncertainty to include the known minimum. We can then have confidence in the procedure to seek the minimum of more realistic functions whose minimums are not known in advance.

The example equation is:

\[ M = (x - 0.7)^2 \]

The initial interval of uncertainty is specified to be:

\( -1 \leq x \leq 2 \)

Clearly, this function will have a minimum at:

\( x = 0.7 \)

Two evaluations of the merit function are required to reduce the interval of uncertainty on the first pass\(^1\). We define \( z_1 \) to be the width of the interval of uncertainty. We define \( z_2 \) to be the width of the interval of uncertainty following the first pass. Thus:

\[ z_1 = 3 \]
\[ z_2 = 0.618 \times 3 = 1.854 \]

We place the first and second values of the design variable at a distance of \( z_2 \) from each of the bounds of the initial interval of uncertainty, as shown in Fig. 1. Thus, the first two values of the design variable are:

\( x_1 = x_{r \text{init}} - z_2 = 0.146 \)
\( x_2 = x_{l \text{init}} + z_2 = 0.854 \)

The merit function is then evaluated for these two values of the design variable:

\[ M(x_1 = 0.146) = (0.146 - 0.7)^2 = 0.307 \]
\[ M(0.854) = 0.024 \]

Since the value of the merit function is higher at \( x_1 \) than it is at \( x_2 \), we reset the left limit of the interval of uncertainty to \( x_1 \), as illustrated in Fig. 2. We also know that the width of the interval of uncertainty can be reduced to:

\[ z_3 = 0.618 \times z_2 = 1.146 \]

after the third evaluation of the merit function. We place the third value of the design variable at a position of \( z_3 \) to the right of the new left bound, as shown in Fig. 2:

\( x_3 = 0.146 + 1.146 = 1.292 \)
Note that the Golden Section procedure guarantees that \( x_2 \) is already placed at a distance of \( z_3 \) to the left of the current right bound. Thus, that value of the design variable, and the value of the merit function at that value, need not be re-computed.

All following passes follow a very similar pattern. We will illustrate one more pass to clarify the procedure.

First, the merit function is evaluated for the new value of the design variable:

\[
M(1.292) = 0.350
\]

Since this value is substantially higher than the value at \( x_2 \), the right bound is reset to coincide with \( x_3 \). This is illustrated in Fig. 3.

The width of the interval of uncertainty after the fourth evaluation of the merit function can be reduced to:

\[
z_4 = 0.618 \times z_3 = 0.708
\]

Thus, the next value of the design variable is placed 0.708 to the left of the new right bound, as shown in Fig. 3.

The value of the merit function at the new point is:

\[
M(0.584) = 0.013
\]

This is the lowest value yet. Thus, the right bound of the interval of uncertainty can be reset to \( x_2 \), as illustrated in Fig. 4.

This sequence can be continued until the interval of uncertainty is reduced to the size requested by the user. Note that the actual minimum value of the design variable, \( x = 0.7 \), is correctly included in the last interval of uncertainty illustrated in Fig. 4.

2 Suggestions for Implementing Golden Section Search

We highly recommend that you make a sketch of the graph of the value of the merit function versus the design variable to obtain a clear picture of what transpires in a golden section search. Estimate the positions of the design variable and
2.2 Minimizing the Number of Variables

You do not need to store arrays of previous values of design variables or values of the merit function in your Golden Section function. To do so would be inefficient. Your Golden Section function should need no arrays.

The numerical error associated with using \( \sqrt{5} - 1 \) in your Golden Section search loop is only parameters that you need at any cycle of the golden section method. Their definitions are:

- \( m_l \): Value of the merit function at \( x_l \) (defined below).
- \( m_r \): Value of the merit function at \( x_r \) (defined below).
- \( l_bnd \): Left bound of the design variable for the current interval of uncertainty.
- \( r_bnd \): Right bound of the design variable for the current interval of uncertainty.
- \( x_l \): Value of the design variable within the current interval of uncertainty that may constitute the next left bound.
- \( x_r \): Value of the design variable within the current interval of uncertainty that may constitute the next right bound.
- \( z \): The most recent interval of uncertainty.

Consider what must be done at any cycle of the Golden Section method after the first. For illustrative purposes, we will assume that \( x_r \) becomes the new right bound at the current cycle. Figure 6 illustrates this case. Everything stated
1. The former value of $x_l$ becomes the new value of $x_r$. Note that the new value of $x_r$ does not need to be explicitly computed by adding $0.618 \times z$ to the left bound.

2. The former value of $m_l$ becomes the new value of $m_r$. The new value of $m_r$ should not be computed by calling the merit function with the new value of $x_r$. Doing so would double the number of times the merit function is evaluated and halve the performance of your Golden Section function! Rather, just re-use the available value of $m_l$.

3. The new value of $x_l$ is computed by moving $0.618 \times z$ to the left of the new right bound.

4. The merit function need be evaluated only once per cycle, in this case at the newly computed value of $x_l$.

2.4 Pre-Computing the Number of Evaluations of the Merit Function

As derived in lecture, the number of evaluations of the merit function required to meet a specified fractional reduction in the interval of uncertainty, $f$, is:

$$N = 1 - 2.078 \ln f$$

However, this is probably not exactly the number of times that you want to execute the loop in your Golden Section function. Rather, you will need to adjust this value to account for the following:

1. You probably need to evaluate the merit function twice before starting your Golden Section loop. These evaluations should be counted in the total.

2. You clearly can not evaluate the merit function a fractional number of times to obtain an exact and arbitrary value of $f$. Thus, you may need to adjust your number of evaluations to provide a value of $f$ slightly better than that requested by the user.

For example, consider the case where the user requests a fractional reduction of $f = 0.01$, which corresponds to 10.57 evaluations of the merit function. Realistically, you would want to evaluate the merit function 11 times, giving an actual $f$ of .008. (10 evaluations would give an $f$ of 0.013, which does not quite satisfy the fractional reduction requested by the user.)

2.5 Completing the Golden Section algorithm

The simplest and most efficient code for the Golden Section method is obtained by making one last reduction of the interval of uncertainty after completing the golden section loop.

An example of the design variable space upon completing the Golden Section loop is shown enlarged in Fig. 7. Two values of the merit function are still available at $x_l$ and $x_r$. Therefore, $m_l$ can be compared to $m_r$, and the interval of uncertainty can be reduced one last time. In the case illustrated in Fig. 7, $x_{l \text{ final}}$ would be set to $l_{\text{bnd}}$ and $x_{r \text{ final}}$ would be set to $x_r$.

However, note that the procedure described in Section 2.3 need not be applied for this last cut. In fact, computing a new $x_l$ and a new $m_l$ would be wasteful, as they would never be used. Thus, the code to make the final reduction in the interval of uncertainty is best implemented
as a short addendum following completion of the golden section loop.

2.6 Checking the Final Interval of Uncertainty

Note that the final interval of uncertainty illustrated in Fig. 7 still contains one value of the design variable, $x_l$. This is because the absolute minimum of the function shown could occur either to the left or to the right of $x_l$. We cannot reduce the interval of uncertainty any further than that shown without evaluating the merit function again.

In particular, note that specifying the final interval of uncertainty to fall between $x_l$ and $x_r$ in Fig. 7 would be incorrect. Be sure that your function does not do this!

2.7 What do I do if I obtain two exactly equal values of the merit function?

The diligent programmer may attempt to code their golden function algorithm to handle the unlikely but possible special case where $m_l$ exactly equals $m_r$. However, this is not necessary.

The only way that a unimodal function can produce exactly equal values for $m_l$ and $m_r$ is if the minimum value falls between $x_l$ and $x_r$. Thus, either the left bound can be re-set to $x_l$ or the right bound can be re-set to $x_r$. The resulting interval of uncertainty would be guaranteed to contain the overall minimum of the function regardless of which choice you make. Thus, your algorithm would continue to reduce the interval of uncertainty in later cycles with no special provisions.

One could write special code to drastically reduce the interval of uncertainty from $x_l$ to $x_r$ in the case that $m_l$ exactly equals $m_r$. However, the likelihood of obtaining exactly equal values of the merit function for two values of the design variable is very small for real merit functions. Thus, we do not recommend that you clutter your code by designing the complex logic for this special case.

2.8 Input and Output

Please do not read in any data from function “gold1D”. All the data that you need should be available from the input calling arguments, as defined on the input/output chart. Likewise, please do not print any results from “gold1D”. Rather, return the results in the output calling arguments defined in the input/output chart.

In general, try to never read in data or print results from a “computational” function such as “gold1D”. To do so unnecessarily limits its versatility. For example, assume that “gold1D” was utilized as part of a solid modeler. You would not want to print the results in this case; rather, you would internally apply the results toward another step in computing your solid model.

If you do wish to print your results, write a simple function that does nothing but take the output from “gold1D” and print it. A test program might simply print the results from the mainline itself.

One possible exception to this rule is error messages. You may print error messages from “computational” routines if your code detects that something has gone wrong. Note that professional code usually does this with special “error handling” procedures instead, but we will not have time to discuss these in this class.