An extension of golden section algorithm for n-variable functions with MATLAB code

G Sandhya Rani, Sarada Jayan and K V Nagaraja
Department of Mathematics, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India
sandhya.reddyj@gmail.com, j_sarada@blr.amrita.edu, kv_nagaraja@blr.amrita.edu

Abstract: Golden section search method is one of the fastest direct search algorithms to solve single variable optimization problems, in which the search space is reduced from \([a, b]\\) to \([0, 1]\). This paper describes an extended golden section search method in order to find the minimum of an n-variable function by transforming its n-dimensional cubic search space to the zero-one n-dimensional cube. The paper also provides a MATLAB code for two-dimensional and three-dimensional golden section search algorithms for a zero-one n-dimensional cube. Numerical results for some benchmark functions up to five dimensions and a comparison of the proposed algorithm with the Neldor Mead Simplex Algorithm is also provided.

1. Introduction:
Optimization is imperative in almost all fields of engineering, sciences and finance. Researchers in each of these areas have to hinge on numerical techniques to solve optimization problems as analytical ways of solving may be impractical in most of the real world application problems. The numerical optimization problems are mainly classified as gradient-based (derivative based) methods and gradient-free methods. A wide variety of derivative-based optimization methods are available to find the minimum of a real valued multivariable function \(f(x)\), if \(f(x)\) is differentiable and it’s gradient, \(\nabla f(x)\) can be estimated precisely using finite differences. But in most of the engineering application problems in optimization, the objective function is discontinuous, non-linear and non-convex for which the derivative either cannot be directly found or the computationally obtained derivative may be unreliable. Such problems are often solved using direct search methods. Direct search methods are referred to as unconstrained optimization techniques which do not require the information about the derivatives of the objective function. Many direct search methods were established and used since 1960s [1, 2, 3], but in the last two decades there is extensive research happening in this field[4, 5, 6, 7] due to the development and interest in parallel and distributed computing.

Among the direct search single variable optimization problems, the Golden Section Search(GSS) is one of the renowned algorithms. Initially Kiefer [8] introduced the golden number, and the Golden section search algorithm, according to [9] is one of the best direct search algorithms used to find the minimum of a single variable optimization problem. It is a region-elimination method in which the search interval \([a, b]\) is first transformed into a unit interval \([0, 1]\), using a linear transformation. After that, two points at \(\tau\) distance from both the end of the search space is taken in such a manner that at every iteration the eliminated region is \(1 - \tau\) to that in the previous iteration. Such a \(\tau\) is obtained by
solving $r^2 = 1 - r$ and the solution of this equation, $r = 0.618$ is termed as the golden number. In GSS, the search interval is reduced to $(0.618)^{n-1}$ after $n$ function evaluations.

According to [9], GSS needs only one function evaluation in each iteration and also the effective region elimination per function evaluation is exactly 38.2%, which is higher than many other methods, which makes this ideal for many application problems.

The authors of [10, 11] have developed and used a two-dimensional GSS for object tracking using Gabor Wavelet Transform. But in this paper they have not mentioned the 2D-GSS algorithm in detail. Later Yen Ching Chang[12] proposes an $n$-dimensional GSS algorithm and its variants. Recently Chakraborthy and Panda [13] proposed a GSS algorithm over hyper-rectangle.

In this paper, like in [13] we choose an $n$-dimensional cubic search space and provide an extension of GSS to find the minimum of $n$-variable functions, by initially transforming the $n$-dimensional cubic search space to an $n$-dimensional zero-one search space using the transformation in [16]. The region elimination rules for $n$-variable functions over an $n$-dimensional cubic search space are defined initially in section 2 and the proposed algorithm is explained in detail in section 3. The percentage of search space eliminated in each iteration is also derived. Numerical results for some benchmark functions given in [14,15] up to 5 dimensions and a comparison of the proposed algorithm with the Neldor-Mead Simplex Algorithm is provided in section 4. Section 5 displays a MATLAB code for the proposed method and the conclusions are presented in section 6.

2. Region elimination rules defined over a mesh for multivariable functions:

In this section we define region elimination rules for a unimodal $n$-variable function defined in an $n$-dimensional cubic search space, using the function values at $2^n$ mesh points inside the search space.

2.1. Region elimination rules for two variable functions over a rectangle

Consider $2^2=4$ mesh points $A(x_1, y_1), B(x_2, y_1), C(x_1, y_2), D(x_2, y_2)$ in the rectangular search space $a_1 < x < b_1$, $a_2 < y < b_2$ as in Fig.1. For unimodal functions for minimization the following can be concluded:

- If $f(A) = \min \{f(A), f(B), f(C), f(D)\}$, then the minimum will not lie in $x_2 < x < b_1$ and $y_2 < y < b_2$ (shaded region in Fig.1(a)).
- If $f(B) = \min \{f(A), f(B), f(C), f(D)\}$, then the minimum will not lie in $a_1 < x < x_1$ and $y_2 < y < b_2$ (shaded region in Fig.1(b)).
- If $f(C) = \min \{f(A), f(B), f(C), f(D)\}$, then the minimum will not lie in $x_2 < x < b_1$ and $y_2 < y < y_1$ (shaded region in Fig.1(c)).
- If $f(D) = \min \{f(A), f(B), f(C), f(D)\}$, then the minimum will not lie in $a_1 < x < x_1$ and $a_2 < y < y_1$ (shaded region in Fig.1(d)).

If the minimum function value is equal at two or three different points, then the region to be eliminated would be the intersection of the corresponding regions in Fig.1. For example, if $f(A) = f(B) = \min \{f(A), f(B), f(C), f(D)\}$, then the minimum will not lie in the region $y_2 < y < b_2$, which is the intersection of shaded regions in Fig.1(a) and Fig.1(b) and thus that region can be eliminated. If the function value is equal at all the points, which is a very rare situation, especially when computations are performed numerically, then we can conclude that the minimum lies inside the rectangle $ABCD$, i.e., we can eliminate $a_1 < x < x_1$, $x_2 < x < b_1$, $a_2 < y < y_1$, and $y_2 < y < b_2$.

2.2. Region elimination rules for three variable functions over a cuboid

Let us assume the search space of a three-dimensional function to be a cuboid, given by $a_1 < x < b_1$, $a_2 < y < b_2$ and $a_3 < y < b_3$. The region elimination rules are defined using the function evaluations at $8(=2^3)$ mesh points inside the search space. Let these points be $A(x_1, y_1, z_1), B(x_1, y_2, z_1), C(x_1, y_1, z_2), D(x_1, y_2, z_2)$ in the $x = x_1$ plane and $E(x_2, y_1, z_1), F(x_2, y_2, z_1), G(x_2, y_1, z_2), H(x_2, y_2, z_2)$ in the $x = x_2$ plane.

For unimodal functions for minimization of $f$ the following can be concluded:
• If $f(A) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $x_2 < x < b_1$, $y_2 < y < b_2$ and $z_2 < z < b_3$.

• If $f(B) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $x_2 < x < b_1$, $a_2 < y < y_1$ and $z_2 < z < b_3$.

• If $f(C) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $x_2 < x < b_1$, $y_2 < y < b_2$ and $a_3 < z < z_1$.

• If $f(D) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $x_2 < x < b_1$, $a_2 < y < y_1$ and $a_3 < z < z_1$.

• If $f(E) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $a_1 < x < x_1$, $y_2 < y < b_2$ and $z_2 < z < b_3$.

• If $f(F) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $a_1 < x < x_1$, $a_2 < y < y_1$ and $z_2 < z < b_3$.

• If $f(G) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $a_1 < x < x_1$, $y_2 < y < b_2$ and $a_3 < z < z_1$.

• If $f(H) = \min \{ f(A), f(B), f(C), f(D), f(E), f(F), f(G), f(H) \}$, then eliminate $a_1 < x < x_1$, $a_2 < y < y_1$ and $a_3 < z < z_1$.

Like in two-variable case if the minimum function values are at more than one point then the region corresponding to the intersection of these regions can be eliminated and if all the function values are equal, it can be concluded that the minimum lies in the cuboid with $A, B, C, D, E, F, G, H$ as corners.

In a similar manner region elimination rules can be described for n-variable functions too over an n-dimensional cubic search space.

![Region Elimination Pattern for two-dimensional problem](image-url)
3. Golden section search algorithm for n-variable functions:

**Step 1**
Choose a lower bound \( a_i \) and an upper bound \( b_i \) for each variable \( x_i \) in the minimization problem. Also choose a termination parameter \( \varepsilon \).

**Step 2**
Transform the n-dimensional cubic search space,
\[
\{ (x_1, x_2, ..., x_n) \mid a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2, ..., a_n \leq x_n \leq b_n \}
\]
to an n-dimensional zero-one cube:
\[
\{ (u_1, u_2, ..., u_n) \mid 0 \leq u_i \leq 1, i = 1, 2, ..., n \}
\]
using the linear transformation
\[
x_1 = (b_1 - a_1)u_1 + a_1 \\
x_2 = (b_2 - a_2)u_2 + a_2 \\
... \\
x_n = (b_n - a_n)u_n + a_n
\]
This would transform the function from \( f(x_1, x_2, ..., x_n) \) to \( g(u_1, u_2, ..., u_n) \) where each variable \( u_i \) will have a lower bound \( a_{u_i} = 0 \) and an upper bound \( b_{u_i} = 1 \) for \( i = 1, 2, ..., n \).

**Step 3**
Evaluate the \( 2^n \) mesh points,
\[
\mathbf{x}_{i_1i_2...i_n} = \begin{pmatrix} a_{u_1} \\ a_{u_2} \\ \vdots \\ a_{u_n} \end{pmatrix} + \begin{pmatrix} (1 - \tau)^{1-i_1} \tau^{i_1}(b_{u_i} - a_{u_i}) \\ (1 - \tau)^{1-i_2} \tau^{i_2}(b_{u_2} - a_{u_2}) \\ \vdots \\ (1 - \tau)^{1-i_n} \tau^{i_n}(b_{u_n} - a_{u_n}) \end{pmatrix},
\]
where \( \tau \) is the golden number.

For two variable problems, the four initial points would be:
\[
x_{00} = (0.382, 0.382) \quad x_{01} = (0.382, 0.618) \quad x_{10} = (0.618, 0.382) \quad x_{11} = (0.618, 0.618).
\]
For three variable problems there will be eight points, which are:
\[
x_{000} = (0.382, 0.382, 0.382) \quad x_{001} = (0.382, 0.382, 0.618) \quad x_{010} = (0.382, 0.618, 0.382) \quad x_{011} = (0.382, 0.618, 0.618) \quad x_{100} = (0.618, 0.382, 0.382) \quad x_{101} = (0.618, 0.382, 0.618) \quad x_{110} = (0.618, 0.618, 0.382) \quad x_{111} = (0.618, 0.618, 0.618).
\]

**Step 4**
Compute the function value at each of the points generated in Step 4, \( g(\mathbf{x}_{i_1i_2...i_n}) \). Depending on at which point the minimum function value is there, apply the region elimination rule defined in Section 3 to eliminate a region. Set new \( a_{u_1}, a_{u_2}, ..., a_{u_n} \) and \( b_{u_1}, b_{u_2}, ..., b_{u_n} \).

**Step 5**
Evaluate \( \mathbf{L}_u = \begin{pmatrix} b_{u_1} \\ b_{u_2} \\ \vdots \\ b_{u_n} \end{pmatrix} - \begin{pmatrix} a_{u_1} \\ a_{u_2} \\ \vdots \\ a_{u_n} \end{pmatrix} \). Is \( \| \mathbf{L}_u \| \leq \varepsilon \)?: If no, go to Step 3; Else Terminate.

The point at which the function value is the least in the final iteration is taken as the minimum point, \( (u_1^*, u_2^*, ..., u_n^*) \) for the function \( g(u_1, u_2, ..., u_n) \) and substituting this point in the transformation given in Step 2, we get the minimum point \( (x_1^*, x_2^*, ..., x_n^*) \) of the given function \( f(x_1, x_2, ..., x_n) \).

The region eliminated in one-dimensional GSS in each iteration is 38.2%, which is \((1 - \tau)\%\). While applying two-dimensional GSS, the region eliminated would be \((1 - \tau)(1 + \tau)\%\}(= 61.8\%). For three-dimensional GSS, it is \((1 - \tau)(1 + \tau + \tau^2)\%\}(= 76.39\%) and similarly we can generalize the region eliminated in each iteration while applying the proposed algorithm would be:
(1 + \tau)(1 + \tau^2 + \cdots + \tau^{n-1})\%

This makes the proposed algorithm fast.

4. Numerical outcomes:
In order to authenticate the performance of the algorithm, we applied the algorithm in some benchmark functions given in [14,15], few of which are presented in this section. All the results are obtained in a 6 GB RAM Intel i5 laptop. Table 1 gives the minimum obtained for some standard functions using the proposed algorithm up to five dimensions. Table 2 provides numerical results for some fixed dimensional benchmark functions.

The proposed algorithm is also compared with the Neldor-Mead simplex Algorithm (NMSA) which is incorporated in MATLAB to find a minimum of a function with the command fminsearch. The comparison of number of iterations and time taken to evaluate the minimum for four functions using both methods tabulated in Table 3, attests the achievement of the extended GSS that is proposed in this paper.

Table 1: Minimum of few benchmark functions obtained using the proposed algorithm for up to 5 dimensions

| Function      | Search Space                     | Minimum obtained |
|---------------|----------------------------------|------------------|
| Sphere        | 2D $u_1^2 + u_2^2$               | $u_1, u_2 \in [-5.125.12]$ (0,0) |
|               | 3D $u_1^2 + u_2^2 + u_3^2$       | $u_1, u_2, u_3 \in [-5.125.12]$ (0,0) |
|               | 4D $u_1^2 + u_2^2 + u_3^2 + u_4^2$ | $u_1, u_2, u_3, u_4 \in [-5.125.12]$ (0,0) |
|               | 5D $u_1^2 + u_2^2 + u_3^2 + u_4^2 + u_5^2$ | $u_1, u_2, u_3, u_4, u_5 \in [-5.125.12]$ (0,0) |
| Ridge         | 2D $u_1^2 + (u_1^2 + u_2^2)$     | $u_1, u_2 \in [-100.100]$ (0,0) |
|               | 3D $u_1^2 + (u_1^2 + u_2^2) + (u_1^2 + u_2^2 + u_3^2)$ | $u_1, u_2, u_3 \in [-100.100]$ (0,0) |
|               | 4D $u_1^2 + (u_1^2 + u_2^2) + (u_1^2 + u_2^2 + u_3^2) + (u_1^2 + u_2^2 + u_3^2 + u_4^2)$ | $u_1, u_2, u_3, u_4 \in [-100.100]$ (0,0) |
| Sum squares function | 2D $u_1^2 + 2u_2^2$ | $u_1, u_2 \in [-10,10]$ (0,0) |
|               | 3D $u_1^2 + 2u_2^2 + 3u_3^2$    | $u_1, u_2, u_3 \in [-10,10]$ (0,0) |
|               | 4D $u_1^2 + 2u_2^2 + 3u_3^2 + 4u_4^2$ | $u_1, u_2, u_3, u_4 \in [-10,10]$ (0,0) |
|               | 5D $u_1^2 + 2u_2^2 + 3u_3^2 + 4u_4^2 + 5u_5^2$ | $u_1, u_2, u_3, u_4, u_5 \in [-10,10]$ (0,0) |
| Schwefel 2.22 | 2D $|u_1| + |u_2| + |u_3|$ | $u_1, u_2 \in [-10,10]$ (0,0) |
|               | 3D $|u_1| + |u_2| + |u_3| + |u_1||u_2||u_3|$ | $u_1, u_2, u_3 \in [-10,10]$ (0,0) |
|               | 4D $|u_1| + |u_2| + |u_3| + |u_4| + |u_1||u_2||u_3||u_4|$ | $u_1, u_2, u_3, u_4 \in [-10,10]$ (0,0) |
|               | 5D $|u_1| + |u_2| + |u_3| + |u_4| + |u_5| + |u_1||u_2||u_3||u_4||u_5|$ | $u_1, u_2, u_3, u_4, u_5 \in [-10,10]$ (0,0) |
| Brent         | 2D $(u_1 + 10)^2 + (u_2 + 10)^2 + e^{-u_1^2-u_2^2}$ | $u_1, u_2 \in [-10,10]$ $(-10, -10)$ |
|               | 3D $(u_1 + 10)^2 + (u_2 + 10)^2 + (u_3 + 10)^2 + e^{-u_1^2-u_2^2-u_3^2}$ | $u_1, u_2, u_3 \in [-10,10]$ $(-10, -10, -10)$ |
|               | 4D $(u_1 + 10)^2 + (u_2 + 10)^2 + (u_3 + 10)^2 + (u_4 + 10)^2 + e^{-u_1^2-u_2^2-u_3^2-u_4^2}$ | $u_1, u_2, u_3, u_4 \in [-10,10]$ $(-10, -10, -10, -10)$ |
|               | 5D $(u_1 + 10)^2 + (u_2 + 10)^2 + (u_3 + 10)^2 + (u_4 + 10)^2 + (u_5 + 10)^2 + e^{-u_1^2-u_2^2-u_3^2-u_4^2-u_5^2}$ | $u_1, u_2, u_3, u_4, u_5 \in [-10,10]$ $(-10, -10, -10, -10, -10)$ |
Table 2: Minimum of few benchmark functions obtained using the proposed algorithm

| Function          | Search Space                        | Minimum obtained |
|-------------------|-------------------------------------|------------------|
| Hyper-ellipsoid   | $u_1^2 + 2u_2^2$                    | $u_1, u_2 \in [-100,100]$ |
| Ackley 2          | $-200e^{-0.2\sqrt{u_1^2 + u_2^2}}$ | $u_1, u_2 \in [-32,32]$ |
| Matyas            | $0.26(u_1^2 + u_2^2) - 0.48u_1u_2$  | $u_1, u_2 \in [-10,10]$ |
| Powell            | $((u_1 + 10u_2)^2 + 5(u_3 - u_4)^2 + (u_2 - 3u_3)^2 + 10(u_1 - u_4)^2$ | $u_1, u_2, u_3, w \in [-10,10]$ |
| Rotated Ellipse   | $7u_1^2 - 6\sqrt{3}u_1u_2 + 13u_2^2$ | $u_1, u_2 \in [-500,500]$ |

Table 3: Comparison of the proposed algorithm with Nelder-Mead Simplex Algorithm

| Function | Results using proposed algorithm | Using Nelder-Mead Simplex Algorithm |
|----------|----------------------------------|------------------------------------|
|          | Initial domain | Iterations | Min Point | Time (in sec) | Initial guess | Iterations | Min Point | Time (in sec) |
| $f_3$    | $-3 \leq u_1 \leq 4,$ | 39 | $(1.0000,1.0000)$ | 0.001 | $(-3.2)$ | 47 | $(1.0000,1.0000)$ | 0.01 |
|          | $-5 \leq u_2 \leq 2$ | | | | $(-3,-5)$ | 49 | | 0.006 |
| $f_2$    | $0 \leq u_1 \leq 3,$ | 39 | $(0,0)$ | 0.002 | $(0,4)$ | 200 | $(0,0)$ | 0.009 |
|          | $0 \leq u_2 \leq 4$ | | | | $(3,0)$ | 200 | | 0.008 |
| $f_3$    | $-5.12 \leq u_1 \leq 5.12,$ | 39 | $(0,0)$ | 0.001 | $(-5.12,-5.12)$ | 46 | $(0,0)$ | 0.01 |
|          | $-5.12 \leq u_2 \leq 5.12,$ | | | | $(-5.12,5.12)$ | 46 | | 0.006 |
| $f_4$    | $1 \leq u_1 \leq 5,$ | 42 | $(1.8122,2.5000)$ | 0.001 | $(1,1)$ | 41 | $(1.8122,2.5000)$ | 0.006 |
|          | $1 \leq u_2 \leq 3$ | | | | $(5,1)$ | 50 | | 0.008 |

$f_1 = \sqrt{(u_1 - 1)^2 + u_2^2}$
$f_2 = \sqrt{u_1u_2}$
$f_3 = u_1^2 + u_2^2$
$f_4 = -|u_1 + u_2 - (u_2 - 2)^2| \sin(x)$

5. MATLAB code for proposed algorithm:

5.1. MATLAB code for finding the minimum of a two-variable function

```matlab
f=@(x,y) sqrt(3*x*4*y); % enter the function
a1 = 0; % lower bound for variable x
b1 = 1; % upper bound for variable x
a2 = 0; % lower bound for variable y
b2 = 1; % upper bound for variable y
epsilon = 0.00000001; % termination criteria
tau=double((sqrt(5)-1)/2); % golden number
```
k=0;  % number of iterations
x1= a1+(1-tau)*( b1-a1); x2= a1+(tau)*( b1-a1);
y1= a2+(1-tau)*( b2-a2); y2= a2+(tau)*( b2-a2);
ck=[x1,y1];  % Point A
fk=[x1,y2];  % Point C
hk=[x2,y1];  % Point B
gk=[x2,y2];  % Point D
fek=f(x1,y1);ffk=f(x1,y2);fhk=f(x2,y1);fgk=f(x2,y2);  % function values at points A, B, C and D.
while sqrt((b1-a1)^2+( b2-a2 )^2) > epsilon  % termination condition
  k=k+1;
  min1=min([fek,fhk,ffk,fgk]);
  if min1==fek
    b1=x2;  b2=y2;
  elseif min1==ffk
    b1=x2;  a2=y1;
  elseif min1==fhk
    a1=x1;  a2=y1;
  elseif min1==fgk
    a1=x1;  b2=y2;
  end
  x1= a1+(1-tau)*( b1-a1); x2= a1+(tau)*( b1-a1);
y1= a2+(1-tau)*( b2-a2); y2= a2+(tau)*( b2-a2);
fek=f(x1,y1);ffk=f(x1,y2);fhk=f(x2,y1);fgk=f(x2,y2);
  min1=min([fek,fhk,ffk,fgk]);
end
if min1==fek
  fprintf('minimum at the point %f', x1,y1);
else if min1==ffk
  fprintf('minimum at the point %f', x1,y2);
else if min1==fhk
  fprintf('minimum at the point %f', x2,y1);
else if min1==fgk
  fprintf('minimum at the point %f', x2,y2);
end
fprintf('minimum value %f', min1);
fprintf('number of iterations %f', k);

5.2. MATLAB code for finding the minimum of a three-variable function

```
g=@(x,y,z) x^2+y^2+z^2;  % enter the function
a1=0;  % lower bound for variable x
b1=1;  % upper bound for variable x
a2=0;  % lower bound for variable y
b2=1;  % upper bound for variable y
a3=0;  % lower bound for variable z
b3=1;  % upper bound for variable z
epsilon=0.00000001;  % termination criteria
tau=double((sqrt(5)-1)/2);  % golden number
k=0;  % number of iterations
x1= a1+(1-tau)*( b1-a1); x2= a1+(tau)*( b1-a1);
```
y1= a1+(1-tau)*( b1-a1); y2= a2+(1-tau)*( b2-a2);
z1= a1+(1-tau)*( b1-a1); z2= a2+(1-tau)*( b2-a2);

ek=[x1,y1,z1]; % Point A
fk=[x1,y2,z1]; % Point B
gk=[x1,y1,z2]; % Point C
hk=[x1,y2,z2]; % Point D
ik=[x2,y1,z1]; % Point E
jk=[x2,y2,z1]; % Point F
kk=[x2,y2,z2]; % Point G
lk=[x2,y2,z2]; % Point H

gek=g(x1,y1,z1); gfk=g(x1,y2,z1); ggk=g(x1,y1,z2); ghk=g(x1,y2,z2);
gik=g(x2,y1,z1); gjk=g(x2,y2,z1); gkk=g(x2,y1,z2); glk=g(x2,y2,z2);

while sqrt((b1-a1)^2+( b2-a2)^2+( b3-a3)^2)>epsilon % termination condition
k=k+1;
min1=min([gek,ghk,gfk,ggk,gik,gjk,gkk,glk]);
if min1==gek
  b1=x1; b2=y1; b3=z2;
elseif min1==gfk
  b1=x2; a2=y1; b3=z2;
elseif min1==ggk
  b1=x2; b2=y2; a3=z1;
elseif min1==ghk
  b1=x2; a2=y1; a3=z1;
elseif min1==gik
  a1=x1; b2=y2; b3=z2;
elseif min1==gjk
  a1=x1; a2=y1; b3=z2;
elseif min1==gkk
  a1=x1; b2=y2; a3=z1;
elseif min1==glk
  a1=x1; a2=y1; a3=z1;
end

if min1==gek
  fprintf('minimum at the point %f, %f, %f', x1,y1,z1);
elseif min1==gfk
  fprintf('minimum at the point %f, %f, %f', x1,y2,z1);
elseif min1==ggk
  fprintf('minimum at the point %f, %f, %f', x1,y1,z2);
elseif min1==ghk
  fprintf('minimum at the point %f, %f, %f', x1,y2,z2);
elseif min1==gik
  fprintf('minimum at the point %f, %f, %f', x2,y1,z1);
elseif min1==gjk
  fprintf('minimum at the point %f, %f, %f', x2,y2,z1);
else
  fprintf('minimum at the point %f, %f, %f', x2,y1,z1);
end
fprintf('minimum at the point %f', x2,y2,z1);
elseif min1==gkk
    fprintf('minimum at the point %f', x2,y1,z2);
elseif min1==glk
    fprintf('minimum at the point %f', x2,y2,z2);
end
fprintf('minimum %f', min1);
fprintf('number of iterations %f', k);

6. Conclusions
The current work describes an extended golden section search algorithm for finding minimum of multidimensional unconstrained optimization problems and establishes the benefit of using it. We need not have the information of the derivative of the function in order to apply this algorithm. Also a global minimum can be obtained using this algorithm for functions that are convex or quasi-convex. As the size of the eliminated region in each iteration is more in the proposed method compared to other search techniques, the number of iterations required to reach the minimum is less and so the time taken to obtain the minimum is also less. This rapidity of the proposed algorithm makes it practical for many engineers and researchers working in optimization. The transformation of the search space to a zero-one cube and the MATLAB code over the zero-one square/cube would enable the user to easily make use of the algorithm for their work.

7. References
[1] Hooke R and Jeeves T A 1961 Direct search solution of numerical and statistical problems Journal of the ACM (JACM) 8 212–229.
[2] Spendley W, Hext G R and Himsworth F 1962 Sequential application of simplex designs in optimisation and evolutionary operation Technometrics 4 441–461.
[3] Nelder J A and Mead R 1965 A simplex method for function minimization The Computer Journal 7 308–313.
[4] Kolda T G, Lewis R M and Torczon V 2003 Optimization by direct search: new perspectives on some classical and modern methods SIAM Review 45 385–482.
[5] Conn A R, Scheinberg K and Vicente L N 2009 Introduction to Derivative-Free Optimization (Philadelphia: SIAM).
[6] Rios L M and Sahinidis N V 2013 Derivative-free optimization: a review of algorithms and comparison of software implementations Journal of Global Optimization 56 1247–1293.
[7] Sanket M Bhat, Sarraf Nikhil Vineetha K V, Sarada Jayan, Dhanesh G Kurup A parallelized method for global single variable optimization (Accepted for publication in IEEE Explore)
[8] Kiefer J 1953 Sequential minimax search for a maximum Proceedings of the American Mathematical Society 4 502–506.
[9] Deb K 1995 Optimization for Engineering Design: Algorithms and Examples (New Delhi: Prentice Hall).
[10] He C, Dong J, Zheng Y F and Ahalt S C 2001 Object Tracking Using the Gabor Wavelet Transform and the Golden Section Algorithm Proc. 2001 IEEE Int. Conf. Robot. Autom. 2 1671-1676.
[11] He C, Dong J, Zheng Y F and Ahalt S C 2002 Object Tracking Using the Gabor Wavelet Transform and the Golden Section Algorithm IEEE Multimedia 4 528-538.
[12] Chang Y C 2009 N-dimension golden section search: its variants and limitations Proceedings of the IEEE 2nd International Conference on Biomedical Engineering and Informatics BMEI'09 1–6.
[13] Chakraborty S K and Panda G 2016 Golden Section Search over a hyper-rectangle: a direct
search method *International J. Mathematics in Operational Research* 8 279-292.

[14] Koupaei J A, Hosseini S M M and Ghaini F M 2016 A new optimization algorithm based on chaotic maps and golden section search method *Engineering Applications of Artificial Intelligence* 50 201-214.

[15] Jamil M and Yang X S 2013 A literature survey of benchmark functions for global optimisation problems *International Journal of Mathematical Modelling and Numerical Optimisation* 4 150-194.

[16] Sarada Jayan and Nagaraja K V 2014 Numerical integration over n-dimensional cubes using generalized Gaussian quadrature *Proceedings of the Jangjeon Mathematical Society* 17 63-69.