Dhaka Univ. J. Sci. 61(1): 75-80, 2013 (January)

# A Generalized Computer Technique for Solving Unconstrained Non-Linear Programming Problems

**H.K. Das and M. B. Hasan**

*Department of Mathematics, University of Dhaka, Dhaka-1000, Bangladesh,*

*Email: hkdas.rohit@gmail.com*

Received on 20.01.2012. Accepted for Publication on 10. 07.2012

**Abstract**

An unconstrained problem with nonlinear objective function has many applications. This is often viewed as a discipline in and of itself. In this paper, we develop a computer technique for solving nonlinear unconstrained problems in a single framework incorporating with Golden section, Gradient Search method. For this, we first combine this algorithm and then develop a generalized computer technique using the programming language MATHEMATICA. We demonstrate our computer technique with a number of numerical examples.

**Index-Terms— Unconstrained NLPP, Optimization, Computer Algebra.**

## I. Introduction

In this paper, we review the basic algorithms for convex and concave quadratic programming (QP) that are part of the Optimization subroutine Library. Optimization might be defined as the science of determining the "best" solution to certain mathematically defined problems, which are often models of physical reality. It involves the study of optimality criteria for problems, the determination of algorithmic methods of solution, the study of the structure of such methods and computer experimentation with methods, both under trial conditions and on real life problems. There is an extremely diverse range of practical applications. These include chemical reactor design, resource allocation, scheduling, blending, data fitting and penalty functions. By nature, optimization techniques are iterative, and in most cases, usually contain a line search step. The aim of this work is to numerically evaluate the performance unconstrained Non-linear Programming Problems (NLPP).

Like linear programming (LP), NLPP is a mathematical technique for determining the optimal solutions to many business problems. But NLPP come in many different shapes and forms. Unlike the simplex method for LP, no single algorithm can solve all these different types of problems. So, we study here not only for the unconstrained NLPP optimization problem but also the constrained NLPP optimization. Finally, we successfully complete this work into a single framework. A 1-D simplex search algorithm was presented by Choo and Kim[20] and they worked only with minimization unconstrained NLP problems. However, because of difficulty of analyzing non-linear calculations, the vast majority of questions that are important to the performance of optimization algorithms in practice usually left unanswered M. J. D. Powell[21].

Actually, Ayoade[18] worked on the one dimensional simplex search only minimization type of problems for several variables using a Fortran 77 program was developed to implement his technique. So that our aim is to develop how we can solve unconstrained problems with maximization and minimization. This technique is incorporated with one dimensional Golden section method and multivariable Gradient Search method.

In this section, we discuss some basic definitions and theorems.

### Preliminaries

### Unconstrained NLPP Optimization

Unconstrained optimization problems have no constraints, so the objective is simply to

Maximize $f(x)$ over all values of $X = (x_1, x_2, \ldots)$.

The necessary conditions that a particular solution $x = x^*$ be optimal when f(x) is a differentiable function and when $f(x)$ is a concave function, this condition also is sufficient, so then solving for $x^*$ reduce to solving the system of n equations obtained by setting the n partial derivatives equal to zero. Unfortunately, for nonlinear functions $f(x)$, these equations often are going to be nonlinear as well, in which case you are unlikely to be able to solve analytically for their simultaneous solution. When a variable $x_j$ does have non negativity constraint $x_j \geq 0$, the preceding necessary and (perhaps) sufficient condition changes slightly to

$$\frac{\partial f}{\partial x_j} \begin{cases} \leq 0 \text{ at } x = x^*, & \text{if } x_j^* = 0 \\ = 0 \text{ at } x = x^*, & \text{if } x_j^* > 0 \end{cases}$$

For each such $j$.

In this paper, we develop a computer technique incorporate with Golden Section, Gradient Search. Our program can solve any kind of unconstrained NLP with faster than the above methods which we mention earlier. So, this program will save our time. And also one does not need to be confused about unconstrained type of the NLPP.

The rest of the paper is organized as follows. In the section 2 and 3 are based on one variable and multivariable unconstrained Optimization problems respectively. In

section 4, we take care on results and discussions with a number of numerical examples. In Section 5, we combine algorithm and develop a computer oriented program for solving any type of unconstrained NLPP problems using MATHEMATICA with input output procedure. In the last section, we give a comparison.

## II. One Variable Unconstrained Optimization

We now begin discussing how to solve some of the types of problems just describe by considering the simplest case unconstrained optimization with just a single variable x, where the differentiable function $f(x)$ to be maximized is concave. Thus the necessary and sufficient condition for a particular solution $x = x^*$ to be optimal (a global maximum) is

$$\frac{dy}{dx} = 0 \qquad at x = x^*$$

If this equation can be solved directly for $x^*$, we can done. However, if $f(x)$ is not a particularly simple functions, so the derivative is not just a linear or quadratic function, it may not be able to solve the equation analytically. If not, the one-dimensional search procedure provides a straightforward way of solving the problem numerically.

## III. Multivariable Unconstrained Optimization

Now consider the problem of maximizing a concave function $f(x)$ of multiple variables $X = (x_1, x_2, \ldots\ldots, x_n)$ when there are no constraints on the feasible values. Suppose again that the necessary and sufficient condition for optimality, given by the system of equations obtained by setting the respective partial derivatives equal to zero, cannot be solved analytically, so that a numerical search procedure must be used. Hence one-dimensional search procedure be extended to this multidimensional problem.

## IV. Results & Discussion

In this section, we present a number of numerical examples to show the efficiency of our technique. Also we show the complexity of the manual process of different types of problems for solving unconstrained NLP problems

**Table. 1. Examples view for Optimality**

| .Example No. | Type | Number of variables | Initial value | Optimal solution | Optimal value |
|---|---|---|---|---|---|
| 1 | Maximize | 1 | No | (4.279, 5] | 29 |
| 2 | Maximize | 2 | (0,0) | (1,1) | 1 |
| 3 | Maximize | 1 | No | 0.836 | 7.8839. |
| 4 | Maximize | 2 | (1,1) | (.3333,1.3333) | 4.66667 |
| 5 | Maximize | 1 | (-3,3) | 2.5 | 6.25 |
| 6 | Maximize | 2 | No | (55/8,9/2) | $392.81 |
| 7 | Maximize | 3 | No | (0,0,0) | 0 |

**Example 1.**

Find the optimal solution to Max $x^2 + 2x$

s /t. $-3 \leq x \leq 5$ with in an initial length of 0.8

(Wayne L. Winston[ 4] ).

**Solution**

Here $b = 5, a = -3, \varepsilon = 0.8$

$r^k * (b - a) \leq 0.8$

$\Rightarrow (0.618)^k \leq \frac{0.8}{8}$

$\Rightarrow k\ln(0.618) \leq \ln(0.1)$

$\Rightarrow k \leq 5$

**Iteration 1** $x_1 = RHP - r * (b - a)$

$= 5 - 0.618 * (5 + 3)$

$\therefore x_1 = 0.056$

$f(x_1) = 0.115735$ ,

$x_2 = LHP + r * (b - a) = -3 + 0.618 * (5 + 3)$

$\therefore x_2 = 1.9144$ , $f(x_2) = 7.4937$ and since $f(x_1) < f(x_2)$

Then the new interval of uncertainty is

$(x_1, b] = (0.056, 5)$

Now $L_1 = 5 - 0.056 = 4.944$

**Iteration 2**

$x_3 = x_2$ ,$x_4 = 0.056 + 0.618 * 4.944 = 3.111$

$f(x_3) = f(x_2) = 7.4937$ ,$f(x_4) = 15.9003$ and since

$f(x_3) < f(x_4)$

Then the new interval of uncertainty is $(x_3, b]$ and

$L_2 = 3.056$

**Iteration 3**

$x_5 = x_4 = 3.111$ , $x_6 = 1.944 + 0.618 * 3.056 = 3.832$

$f(x_5) = 15.90052$ ,$f(x_6) = 22.34822$ and since $f(x_5) < f(x_6)$

Then the new interval of uncertainty is $(3.111, 5]$ and $L_3 = 5 - 3.111 = 1.889$

**Iteration 4**

$x_7 = x_3 = 3.832$ , $x_8 = 3.111 + 0.618 * 1.889 = 4.278$

$f(x_7) = 22.34822$ ,$f(x_8) = 26.86152$ and since $f(x_7) < f(x_8)$

Then the new interval of uncertainty is $(x_7, b] = (3.832, 5]$ and $L_4 = 1.168$

**Iteration 5**

$x_9 = x_8 = 4.278$ , $x_{10} = 3.832 + 0.618 * 1.168 = 4.550$

$f(x_9) = 26.86152$ , $f(x_10) = 29.8025$and since $f(x_9) < f(x_{10})$

Then the new interval of uncertainty is $(x_9, b] = (4.279, 5]$ and $L_5 = 0.720$

$L_5 = 0.720 < 0.8$

The optimal solution lies in $(4.279, 5]$.

**Example 2**

Consider maximizing the functions ( Liberman [3]).

Max: g= $2x_1x_2 + 2x_2 - x_1^2 - 2x_2^2$

**Solution**

It is easy to see that the given problem is concave.

Now we will solve this problem by Gradient Search method.

**Iteration 1**

Let, $x^r = (0,0)$ is the initial solution.

$\nabla f(x) = (2x_2 - 2x_1, -4x_2 + 2x_1 + 2)$

$\nabla f(0,0) = (0,2)$

$x^r = x^r + t\,\nabla f(0,0) = (0,0) + t\,(0,2) = (0,2t)$

$f(x^r) = f(0,2t) = 0 + 4t - 0 - 2 * 4\,t^2 = 4t - 48t^2$

$f'(t) = 4 - 16t = 0$

This implies $t = 1/4$.

$x^{\wedge r} = (0,2 * 1/4) = (0,1/2)$.

$\nabla f(0,1/2)$=(1,0) with $|\nabla f|$=1.

**Iteration 2**

$x^r = (0,1/2)$

$x^r = x^r + t\,\nabla f(0,1/2)$=(0,1/2) + t (1,0)=(t,1/2)

f($x^r$)=f(t,1/2)= t - $t^2$-1

f'(t)=1-2t=0

This implies t=1/2.

$x^r$ =(1/2,1/2).

$\nabla f(1/2,1/2)$=(0,1) with $|\nabla f|$=0.707

Similarly after some iteration we get the iteration 6

**Iteration 6**

$x^r = (3/4,7/8)$.

$x^r = x^r + t\,\nabla f(3/4,7/8) = (\frac{3+t}{4}, 7/8)$

$f(x^r) = f\left(\frac{3+t}{4}, \frac{7}{8}\right) = 2(\frac{3+t}{4})\,7/8) + 2 \times 7/8 - 1/9(3 +$

$t)\wedge 2 - 2\left(\frac{7}{8}\right)^2$

$f'(t) = 0$

This implies t=7/8.

$x^r = (7/8,7/8)$.

$\nabla f(7/8,7/8) = (0,1/4)$ with $|\nabla f| = 0.25$

Similarly, we will get the desired results.

**Example 3**

The function to be maximized is (Liberman[3]) . $f(x) = 12x - 3x^4 - 2x^6$,

**Example 4**

Consider maximizing the function (Taha [9])

f($x_1$,$x_2$)=$4x_1 + 6x_2 - 2x_1^2 - 2x_1x_2 - 2x_2^2$

**Real life examples of Unconstrained NLP**

**Example 5**

If costs a monopolist \$5/unit to produce a product. If he produces x units of the product, then each can be sold for 10-x dollars ($0 \le x \le 10$). To maximize profit, how much should the monopolist produce?(Wayne L. Winston[4] ).

**Example 6**

A monopolist producing a single product has two types of customers. If $x_1$units are produced for customer 1, then customer 1 is willing to pay a price of 70-$4x_1$ dollars. If $x_2$Units are produced for customer 2, then customer 2 is willing to pay a price of 150-15$x_2$ dollars. For x>0, the cost of manufacturing x units is 100+15x dollars. To maximize profit, how much should the monopolist sell to each customer? ( Winston [4 ] ).

**Example 7**

Consider the following unconstrained optimization problem: (Liberman [3 ])

Maximize $f(x, y, z) = -x^2 - 6y^2 + 3xy + 3yz - z^2$

Starting with initial solution (1,1,1).

**V. A Generalized NLP Technique**

In this section, we first improve an algorithm for solving unconstrained type of NLPP. Then we develop a code using programming language *MATHEMATICA* [7].

**Algorithm**

**Step1**: Input number of variables v.

**Step 2:** If number of variables v=1, then go to the following Sub-Step.

**Sub-Step 1:**Let, $\bar{x}$ be the optimal solution to the NLP Max f(x) s/t. $a < x < b$ .

**Sub-Step 2:**Find out two points $x_1 < x_2$ and evaluate$f(x_1)$and $f(x_2)$

$x_1 = RHP - r*(b-a)$, $x_2 = LHP + r*(b-a)$

**Sub-Step 3: Case 1:**If $f(x_1) < f(x_2)$Then $\bar{x} \in (x_1, b]$

**Case2:**If $f(x_1) > f(x_2)$Then $\bar{x} \in [a, x_2)$

**Case 3:** If $f(x_1) = f(x_2)$Then $\bar{x} \in [a, x_2]$

Then the interval in which $\bar{x}$ is called the interval of uncertainty.

**Sub-Step 4:** Determine which of case 1-3 holds and find a reduced interval of uncertainty

**Sub-Step 5:** Repeat the process until the interval is sufficiently small. If not, go to step 3.

**Step 3:** If number of variables v≥2 then go to the following Sub-Step.

**Sub-Step 1:** Select $\varepsilon$ and any initial trial solution $x^1$. Max $Z = f(x), x \in \mathbb{R}^+ \cup \{0\}$.

**Sub-Step 2:** Express $f(x^1 + \nabla f(x^1))$ as a function t by setting$x_j = x_j^1 + t(\frac{\partial f}{\partial x_j})_{x=x^1}$ and

**Sub-Step 3:**Using the one dimensional search procedure find $t = t^*$ s.t.

$f(x^1 + t\nabla f(x^1))$is maximized over $t \geq 0$

**Sub-Step 4:** Calculate $\nabla f(x^1)$ at new $x^1$If$\nabla f(x^1) \leq \varepsilon$ i.e. If $\frac{\partial f}{\partial x_i} < \varepsilon$ .

**Stop.**

In the next section, we develop a computer technique for solving unconstrained type of NLP problems.

### Computer technique for Unconstrained NLP

In this section, we developed a code for solving NLP problems.

In[1]:=

```
Needs["Miscellaneous`RealOnly`"]
```

```
MA[GRADIENT_] := Module[{},
  n = Input["Enter number of variables n"];
  f = Evaluate[Input["Objective function"]];
  h[{x_, y_, z_}] :=
   Evaluate[
    Input["Objective function of gradient dominating"]];
  ξ = Input["Enter tolerence n"];
  Do[p[i] = Input["initial value"], {i, 1, n}];
  xp = Table[p[i], {i, 1, n}];
  xpp = Table[tempxp[i], {i, 1, n}];
  Do[
   q[i] = Input["v"], {i, 1, n}];
  xpp = Table[q[i], {i, 1, n}];

  Do[
   x[i] = D[f, q[i]], {i, 1, n}];
  grad = Table[x[i], {i, 1, n}];
  s[{x_, y_, z_}] = grad;
  test = Sqrt[s[xp].s[xp]];
  Print["  x'            f'         Iteration      Δf(
      x')                  x'+tDf(x')
      t*        "]
   i = 0;

  While[test ≥ ξ,
   {
    Do[
     x[i] = D[f, q[i]], {i, 1, n}];
    grad = Table[x[i], {i, 1, n}];
    s[{x_, y_, z_}] = grad;
    dfx = s[xp];
    xp = xp + t*dfx;
    i = i + 1;
    txp = xp;
    g[t_] = h[xp];
    pts = Solve[g'[t] == 0, t];
    xp = xp /. pts;
    xp = First[xp];
    Print[N[xp], "       ", h[xp] // N, "     ", i,
     "                ", dfx, "            ", txp // Simplify,
     "       ", pts];
    temp = s[xp];
    test = N[Sqrt[temp.temp]];
    tempxp = Flatten[xp];
    xpp = Table[tempxp[[i]], {i, 1, n}];
    xp = xpp;
   }
  ]
 ]
```

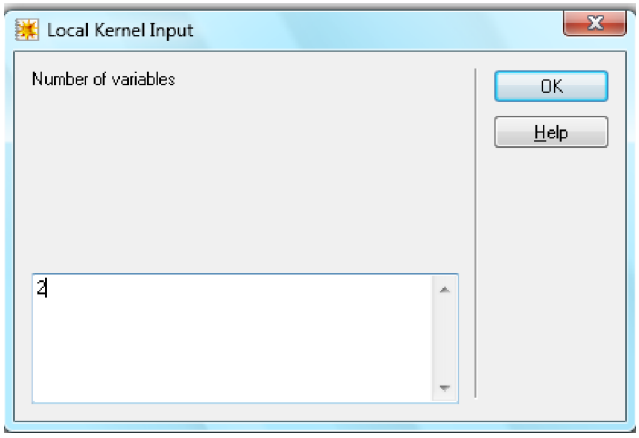
```
BA[GOLDEN_] := Module[{},
  a =
   Input["Enter  Largest Left hand point for Golden
      Section method i.e. a"];
  b =
   Input["Enter Largest Right hand point for Golden
      Section method i.e. b"];
  e = Input["Enter  Length of uncertainity"];
  g[x_] = Evaluate[Input["Objective function"]];
Print["Initial Interval=[", a, ",", b, "]"];
r = 0.618;
pts = FindRoot[(b - a) * r^k == e, {k, 1}];
m = Ceiling[k] /. pts;
Do[Label[st]; x1 = b - r * (b - a);
    Label[th]; x2 = a + r * (b - a);
    If[g[x1] < g[x2],
      Print["Iteration Number =", k,
     " and it's interval of uncertainity is:"];
    Print["[", x1, ",", b, "]"]; a = x1; Continue[th],
      Print["Iteration Number =", k,
     " and it's interval of uncertainity is:"];
    Print["[", a, ",", x2, "]"]; b = x2; Continue[st]], {k, 1, m}];
  Print["The Functional value is : ", g[(a + b)/2]]
 ]


main[unconcons_] := Module[{},
  v = Input["Number of variables"];
  If[v == 1, BA[GOLDEN], MA[GRADIENT]]
 ]
```

**Programming Input and Output systems**

In this section we have to take data of the various types of problems using the run file "Local kernel box" to get the results. In this program, we have used two module functions BA[GOLDEN_], MA[GRADIENT_]. The main module function is main [unconcons_] which call all the module functions define above. The combined programming input output is presented as follows.

When we run the above program this call the number of variables, initial values & the objective function e.t.c.. When these requirements complete then it automatically choose any of the module function between the two module functions. Similarly, it these sub module function take some input such as number of variables in the form {x1, x2, … , xn}. When we give the required statement we must press "Enter" individually for each required statement. Finally, we will get the desired results as the following way.




**Example 1**

**Input**

**main[unconcons]**

**Output**

```
Iteration Number = 5  and it's interval
of uncertainity is:
[ -3 , -2.27884 ]
The Functional value is : 1.41564
TimeUsed[]
0.114
```

**Example 2**

**Input**

**main[unconcons]**

**Output**

For limiting page we don't show the complete table by step by step.

$\{0.999992, 0.999992\}$ 1. 34

$\left\{\frac{1}{65536}, 0\right\}$ $\left\{\frac{65535+t}{65536}, \frac{131071}{131072}\right\}$ $\left\{\left\{t \to \frac{1}{2}\right\}\right\}$

$\{0.999992, 0.999996\}$ 1. 35 $\left\{0, \frac{1}{65536}\right\}$

$\left\{\frac{131071}{131072}, \frac{131071+2t}{131072}\right\}$ $\left\{\left\{t \to \frac{1}{4}\right\}\right\}$

After 35 iterations it gives the solution approximate (1,1) and the optimal value approximate to 1.

**Example 5**

**Input:**

**main[unconcons]**

**Output:**

```
Iteration Number = 15 and it's interval
of uncertainty is:
[2.49654, 2.50387]
The Functional value is 6.25
```

**Example 7**

After 21th iteration we get the approximate result is (0,0,0) with optimal value 0.

**Example 5**

**Input:**

**main[unconcons]**

**Output:**

$$\{4.42451\times10^{-6},\ 2.26898\times10^{-6},\ 4.42451\times10^{-6}\}\qquad -9.80745\times10^{-12}$$

$$\left\{\frac{47116535326076910470 49}{1144556201520286695842381824},\ -\frac{14134960597823073141147}{57227810076014334792119 0912},\ \frac{47116535326076910470 49}{1144556201520286695842381824}\right\}$$

$$\left\{\frac{47116535326076910470 49\,(1+t)}{1144556201520286695842381824},\ -\frac{47116535326076910470 49\,(-1+6t)}{1144556201520286695842381824},\ \frac{47116535326076910470 49\,(1+t)}{1144556201520286695842381824}\right\}$$

$$\left\{\left\{t\to\frac{19}{254}\right\}\right\}$$

## VI. Comparison and Discussion

In this section, we give a time comparison chart to show the efficiency of our technique. We used the computer configuration as: Processor: Intel(R) Pentium(R) Dual CPU E2180@2.00GHZ 2.00GHZ, Memory (RAM):1.00 GB and the System type: 32-bit operating system. The input-output shows that our technique can be worked on the multiple variables of NLP problems. We also present the time comparison of our technique and the build in command.

**Table 2. Comparison with Build in Command**

| No | NO Of Variable | Iteration Use | Programming Command Time | Direct Command Time |
|---|---|---|---|---|
| 1 | 1 | 5 | 0.114 | 0.121 |
| 2 | 2 | 35 | 0.21 | 0.219 |
| 3 | 1 | 21 | 0.13 | 0.14 |
| 4 | 2 | 23 | 0.11 | 0.12 |
| 5 | 1 | 15 | 0.0125 | 0.21 |
| 6 | 2 | 29 | 0.212 | 0.221 |
| 7 | 3 | 21 | 0.221 | 0.233 |

## Conclusion

In this paper, we developed a combined algorithm and its computer program incorporated with the conservatory of traditional Golden section, Gradient Search method for solving unconstrained NLP problems. We demonstrated our algorithm and program by a number of numerical examples. We observed that the result obtained by our procedure is completely identical with that of the other methods which are laborious and time consuming. We therefore, hope that our program can solve any type of unconstrained NLP and will save our time and labor.

---


1. Ravindran, Phillips & Solberg", Operation Research ", John Wiley and Sons, New York.
2. Swarup, K., P. k. Gupta and Man Mohan (2003),"Tracts in Operation Research", Revised Edition, ISBN:81-8054-059-6.
3. Hillier, F. S., and G. J. Lieberman, "Introduction to Operation Research", Mc Graw-Hill International Edition, USA, 1995.
4. Wayne L. Winston, "Application and algorithms", Duxbury press, Belmont, California, U.S.A, 1994.
5. Das H. K., T. Saha & M. B. Hasan, "Numerical Experiments by improving a numerical methods for solving Game problems thorough computer algebra", IJDS, **3,1, 23-58**, 2011.
6. Wolfe, P., "The simplex method for quadratic programming," Econometrica, 27, 3, 382-398, 1959.
7. Wolfram, S., "Mathematica," Addision-Wesly Publishing Company, Melno Park, California, New York, 2001.
8. WWW.eom.springer.de.com
9. Hamdhy.A.T., "Operation Research, 8th edition", Prentice Hall of India, PVT. LTD, New Delhi, 1862.
10. Kambo N. S., "Mathmatical Programming Techniques, Revised edition", Affiliated east-west press PVT LTD., Banglore (1984,1991).
11. Gupta P.K., D.S Hira, "Problems IN Opereation Reasearch",S.Chand & Company LTD,Ram Nagar,New Delhi-110055.
12. Don, E., "Theory and Problems of Mathmatica", Schaum's Outline Series, McGraw-Hill,Washington, D.C, 2001.
13. Van C. D. P., A.Whinstone; E.M.L.Bele, "A Comparison of Two methods for quardratic Programming", Operation Research, 14, .3, 422-443, 1966.
14. Sanders J.L., "A Nonlinear Decomposition Principle", Operation Research, 13,.2, 266-271, 1965.
15. Jenson D..L, A.J.King, "A Decomposition method for quardratic programming", IBM SYSREMS JOURNAL, 31,1,1992.
16. Thomas L. Saaty, "Mathematical Methods of Operations Research",McGraw-Hill Book Company,Inc,New York St.Louis San Francisco .
17. Sasieni M., A. Yaspan, L. Friendman, "Operations Research methods and problems", 9th edition, John Wiley & Sons,Int., 1966.
18. Kue A.., "Numerical Experiments with One Dimensional nonlinear simplex search", Computers & Operations Research, 18, 497-506, 1991.
19. Xue G., "One the convergence of one dimensional simplex search", Computer & Operations Research, 16,113-116,1989.
20. Choo E. & C. Kim, "One dimension simplex search", Computer & Operations Research 14,47-54,1987.

21. Powell M. J. D., Convergence properties of algorithms for nonlinear optimization, SIAM ,Rev, 28 487-500,1986.