DirectSearch[Search] - numerically compute minimum or maximum of a function

Calling Sequence

Search(**f**, **constr**, **options**)

Parameters

`f`	–	*algebraic, procedure*; objective function
`constr`	–	(optional) *list(relation), set(relation)*; constraints
`options`	–	(optional) equation(s) of the form **option=value** where **option** is one of **assume**, **checkexit**, **checksolution**, **evaluationlimit**, **initialpoint**, **maximize**, **searchpath**, **step**, **tolerances**, **usewarning**, or **variables**; specify options for the **Search** command.

Description

- The **Search** command numerically computes the minimum (maximum) of a real-valued nonlinear multivariate function **f** with (without) constraints **constr**. The **Search** is derivative-free direct searching method, i.e. it do not require the objective function f(x1,x2,…,xn) to be differentiable and continuous. Generally, a local minimum is returned unless the problem is convex. For global nonlinear optimization of multivariable function see GlobalSearch.
- The first parameter **f** is the objective function, which must be an algebraic expression or name of procedure that accepts *n* floating-point parameters representing the problem variables and returns a float.
- If the option **variables** is not specified the problem variables are the indeterminates of type name found in the algebraic expression **f**. If **f** is a name of procedure the problem variables are the names of procedure formal parameters which can be returned by **op(1, eval(f))** command. When the option **variables** is specified the names of procedure formal parameters are ignored and problem variables are the names in **variables** list.
- The second parameter **constr** is optional and is a set or list of relations (of type `<`, `<=`, `< >` or `=`) involving the problem variables or (and) names of procedures. The any procedure in **constr** must accepts *n* floating-point parameters representing the problem variables in the same order as in the procedure **f** and returns a float. If objective function **f** is not a name of procedure and **constr** includes names of procedures then the option **variables** must be specified in order to give the order of problem variables in these procedures explicitly.
- **Search** returns the solution as a list containing the final minimum (or maximum) value, the extremum point and a number of objective function evaluations. If the function is a procedure or(and) **variables** option is provided the extremum point is a Vector with problem variable values, otherwise the extremum point is a list of equations *varname* = *value.*

Options

The **options** argument can contain the following equations.

- **assume = positive, negative, nonpositive, nonnegative** -- Assume that all variables are positive, negative, non-positive, or non-negative.
- **checkexit** = *posint* -- Set the number of exit condition checking before search stoping. The default value is **2**. The more is **checkexit** the more is number of objective function evaluations but also the more are both reliability and accuracy. When **checkexit=1** the number of function evaluation of quadratic and near quadrartic objective functions is very small.
- **checksolution** = *nonnegint* -- Set the number of randomly distributed new initial points near obtained extremum point for the additional searches, starting from the new initial points, in order to verify solution. The default value is **0**. The additional search is very time consumed. The number of objective function evaluations is increased approximately in **checksolution** times. The more is **checksolution** the more are both reliability and accuracy.
- **evaluationlimit** = *posint* -- Set the maximum number of objective function evaluations. The default value is **10 000**.
- **initialpoint** = *Vector(equation)*, *Array(equation)*, *list(equation)*, *set(equation)*, or *Vector(realcons)*, *Array(realcons)*, *list(realcons)* -- Use the provided initial point, which is a Vector, Array, list or set of equations *varname* = *value* (for any form input of the objective function and constraints) or a Vector, Array, list of exactly *n* values (for procedure form input of objective function and constraints or for any form input of the objective function and constraints when option **variables** is specified). *Varnames* are names of the problem variables. The default value of initial point are [0.9,..., 0.9]. If initial point does not satisfy some inequality constraints or a complex value is encountered a new feasible point is automatically and randomly searched.
- **maximize** or **maximize** = *truefalse* -- Maximize the objective function when equal to **true** and minimize when equal to **false**. The option **'maximize'** is equivalent to **maximize** = **true**. The default is **maximize** = **false**.
- **searchpath**=*'name'* -- If this option is provided **name** return Matrix of all search points; each column of Matrix correspond to one search point.
- **step** = *positive* -- Set the initial searching step. The default value is **1.0**.
- **tolerances =[positive, positive]**, **[positive]**, or **positive** -- Set the absolute tolerances for the extremum point (tolerances[1]) and minimum (or maximum) value (tolerances[2]). The search is stoped when the following exit conditions are met: ||Xmin-Xold ||<=tolerances[1] and |fmin-fold |<=tolerances[2]. The default value is $[10^{-6}, 10^{-6}]$. When tolerances contains one value this value corresponds to both the extremum point and minimum (or maximum) value.
- **usewarning=truefalse** -- Allow or do not allow to use Warnings. The default value is **true**.
- **variables**=*list(name)* -- Specify the names of problem variables and its order explicitly.

Notes

- The **Search** solvers are iterative in nature and require an initial point. The quality of the solution can depend greatly on the point chosen, so it is recommended that you provide a point using the **initialpoint** option.
- The computation is performed in floating-point. Therefore, all data provided must have type realcons and all returned solutions are floating-point, even if the problem is specified with exact values.
- When a complex value is encountered during the extremum searching, the **Search** tries to find a new feasible real point.
- The **Search** command will choose either the hardware floating-point environment or the arbitrary-precision software floating-point environment to

perform the computations. To maximize efficiency, the solvers attempt to use hardware floats whenever possible. Software floats are used only when the environment variable UseHardwareFloats is set to 'false', or when UseHardwareFloats is 'deduced' and Digits is greater than the value of *evalhf*(*Digits*).

- By default, **Search** evaluates the procedure calls using the evalhf command. Procedures that contain any Maple constructs not supported by **evalhf** are evaluated using the slower evalf command. For more information on **evalhf** construct support, see the evalhf and evalhf/procedure help pages.
- Although the *assume* option is accepted, general assumptions are not supported.
- It is recommended that you try different initial points with each problem to verify that the solution is indeed an extremum.

Method description

- The **Search** optimization algorithm is an original derivative-free direct search algorithm which has the following features. It velocity almost matches one of the fastest direct search algorithm, i.e. conjugate direction Powell's algorithm, and in terms of reliability it outperforms one of the most reliable direct search algorithm, Nelder-Meed simplex algorithm. By velocity we mean the number of objective function evaluations.
- The **Search** optimization algorithm was optimized so to minimize the number of function evaluation given high reliability and accuracy. The **Search** constructs a set of conjugate directions with orthogonal shift from previous set of conjugate directions. The optimization algorithm is not quite greedy, that is, moving on directions that do not contain current extremum point is allowed. If inequality constraints are specified the **Search** never computes the function values that do not satisfies the inequality constraints, instead it search feasible point. For equation constraints it use a penalty function method.
- The method implemented in **Search** can be referred to as conjugate direction method with orthogonal shift.
- **Search** as well as Powell's algorithm uses conjugate search directions. Therefore the extremum point of the n-dimensional quadratic function is achieved when n conjugate search directions are constructed for the first time.
- In constructing of conjugate directions an orthogonal shift from the subset of already constructed conjugate directions is used. The orthogonal shift make the algorithm non greedy but greatly increases its reliability.
- The algorithm consists of three stages. At stage one the first search direction is defined as being the opposite of quasi-gradient. At stage two the first n conjugate directions using the orthogonal shift are constructed. At the third stage the main iteration cycle is carried out. This cycle covers the iterative update of the constructed conjugate directions using the orthogonal shift. Below we consider these three stages of minimum search for the n-dimensional nonlinear function $f(x_1, x_2, ..., x_n)$ at $n > 1$ in more detail.

- **Stage I.** Initial n search directions correspond to coordinate axes: $\boldsymbol{u_1} = (1, 0, 0, ...)$, $\boldsymbol{u_2} = (0, 1, 0, ...)$, ..., $\boldsymbol{u_n} = (0, 0, ..., n)$. One step is made in each direction. The initial step value L is determined by a user. By default it is equal to 1. Function increments for each direction $df_1, ..., df_n$ are calculated. As the first conjugate direction we select the anti-gradient direction which corresponds to the normalized vector $\boldsymbol{u}^* = Normalize(-df_1, ..., -df_n)$. The first search direction is replaced by the anti-gradient direction $\boldsymbol{u}_1 = \boldsymbol{u}^*$ and we carry out the one-dimension minimum search for this direction. Let $\boldsymbol{x}_{\mathbf{min}}^{(\mathbf{1})}$ denote the point of obtained minimum.

- **Stage II.** Initial n search directions at this stage are: $\boldsymbol{u_1} = \boldsymbol{u}^*$, $\boldsymbol{u_2} = (0, 1, 0, ...)$, ..., $\boldsymbol{u_n} = (0, 0, ..., n)$. The step value L at this stage is the same and equal to the initial step value. The orthogonal shift value is $L_s = 0.62 \cdot L$ and is also the same. We shall describe the constructing of the rest of $n - 1$ conjugate directions as the following pseudo-code.

```
for i from 2 to n    # the cycle of creating n-1 conjugate directions

    We make the orthogonal shift from the already available $\boldsymbol{u_1}, ..., \boldsymbol{u_{i-1}}$ conjugate directions from the current minimum point $\boldsymbol{x}_{\mathbf{min}}^{(i-1)}$. The orthogonal shift direction can be obtained by means of the Gram-Schmidt orthogonalization process. As a result of the $GramSchmidt(\boldsymbol{u_1}, ..., \boldsymbol{u_i})$ procedure we have $\boldsymbol{u_1^*}, ..., \boldsymbol{u_i^*}$ orthonormalized vectors. Vectors $\boldsymbol{u_1^*}, ..., \boldsymbol{u_{i-1}^*}$ are not used. The orthogonal shift direction is defined by vector $\boldsymbol{u_i^*}$. In this way we have a point corresponding to the orthogonal shift $\boldsymbol{y}_{\mathbf{min}}^{(i-1)} = \boldsymbol{x}_{\mathbf{min}}^{(i-1)} + L_s \cdot \boldsymbol{u_i^*}$.

    for j from 1 to i-1    # repeated movement in i-1 conjugate directions

        Starting from point $\boldsymbol{y}_{\mathbf{min}}^{(i-1)}$ we perform a one-dimensional search for minimum in $\boldsymbol{u_j^*}$ direction.
        If the function value in the obtained point is less than the function value in point $\boldsymbol{y}_{\mathbf{min}}^{(i-1)}$, we update point $\boldsymbol{y}_{\mathbf{min}}^{(i-1)}$.

    end for j.

    Direction $\boldsymbol{u_i}$ is replaced by a direction connecting points $\boldsymbol{x}_{\mathbf{min}}^{(i-1)}$ and $\boldsymbol{y}_{\mathbf{min}}^{(i-1)}$ as follows: $\boldsymbol{u_i} = Normalize(\boldsymbol{x}_{\mathbf{min}}^{(i-1)} - \boldsymbol{y}_{\mathbf{min}}^{(i-1)})$, if $f(\boldsymbol{x}_{\mathbf{min}}^{(i-1)}) < f(\boldsymbol{y}_{\mathbf{min}}^{(i-1)})$, or $\boldsymbol{u_i} = Normalize(\boldsymbol{y}_{\mathbf{min}}^{(i-1)} - \boldsymbol{x}_{\mathbf{min}}^{(i-1)})$ otherwise. This direction will be conjugate to all $\boldsymbol{u_1}, ..., \boldsymbol{u_{i-1}}$ previous conjugate directions. A one-dimensional search for minimum in $\boldsymbol{u_i}$ direction is performed starting from point $\boldsymbol{x}_{\mathbf{min}}^{(i-1)}$, if $f(\boldsymbol{x}_{\mathbf{min}}^{(i-1)}) < f(\boldsymbol{y}_{\mathbf{min}}^{(i-1)})$, or from point $\boldsymbol{y}_{\mathbf{min}}^{(i-1)}$ otherwise. The resulting minimum point becomes the current minimum point $\boldsymbol{x}_{\mathbf{min}}^{(i)}$ for the following cycle iteration.

end for i.
```

If the target function is the n-dimensional quadratic function, its minimum is achieved already at the end of stage II.

- **Stage III.** Initial n search directions at this stage correspond to mutual conjugate directions found at stages II: $\boldsymbol{u_1}, ..., \boldsymbol{u_n}$. The initial step value $L = 0.32 \cdot \left\| \boldsymbol{x}_{\mathbf{min}}^{(n)} - \boldsymbol{x}_{\mathbf{min}}^{(n-1)} \right\|$. If $L = 0$, then $L = tolerances_1$, where $tolerances_1$ is a confidence interval for the extremum point. Initial orthogonal shift value is $L_s = 0.62 \cdot L$. If $L_s = 0$, then $L_s = L$. Let $\boldsymbol{x}_{\mathbf{min}}^{(\mathbf{1})}$ denote the current minimum point. We shall describe the main iteration cycle as the following pseudo-code.

```
for i from 1 to infinity    # main iteration cycle

    We perform the orthogonal shift from the already available $\boldsymbol{u_n}, ..., \boldsymbol{u_2}$ conjugate directions from the current minimum point
```

$\boldsymbol{x}^{(i)}_{\mathbf{min}}$. The orthogonal shift direction can be obtained by means of the Gram-Schmidt orthogonalization process. As a result of the $GramSchmidt(\boldsymbol{u}_n, ..., \boldsymbol{u}_1)$ procedure we have $\boldsymbol{u}^*_n, ..., \boldsymbol{u}^*_1$ orthonormalized vectors. Vectors $\boldsymbol{u}^*_n, ..., \boldsymbol{u}^*_2$ are not used. The orthogonal shift direction is defined by vector $\boldsymbol{u}^*_1$. In this way we have a point corresponding to the orthogonal shift $\boldsymbol{y}^{(i)}_{\mathbf{min}} = \boldsymbol{x}^{(i)}_{\mathbf{min}} + L_s \cdot \boldsymbol{u}^*_1$.
We circularly shift the whole set of conjugate directions $\boldsymbol{u}_1, ..., \boldsymbol{u}_n$ to the left. In other words we re-denote the set of conjugate directions as follows: let us denote $\boldsymbol{u}_2$ as $\boldsymbol{u}_1$, $\boldsymbol{u}_3$ as $\boldsymbol{u}_2, \ldots, \boldsymbol{u}_n$ as $\boldsymbol{u}_{n-1}$ and $\boldsymbol{u}_1$ as $\boldsymbol{u}_n$.

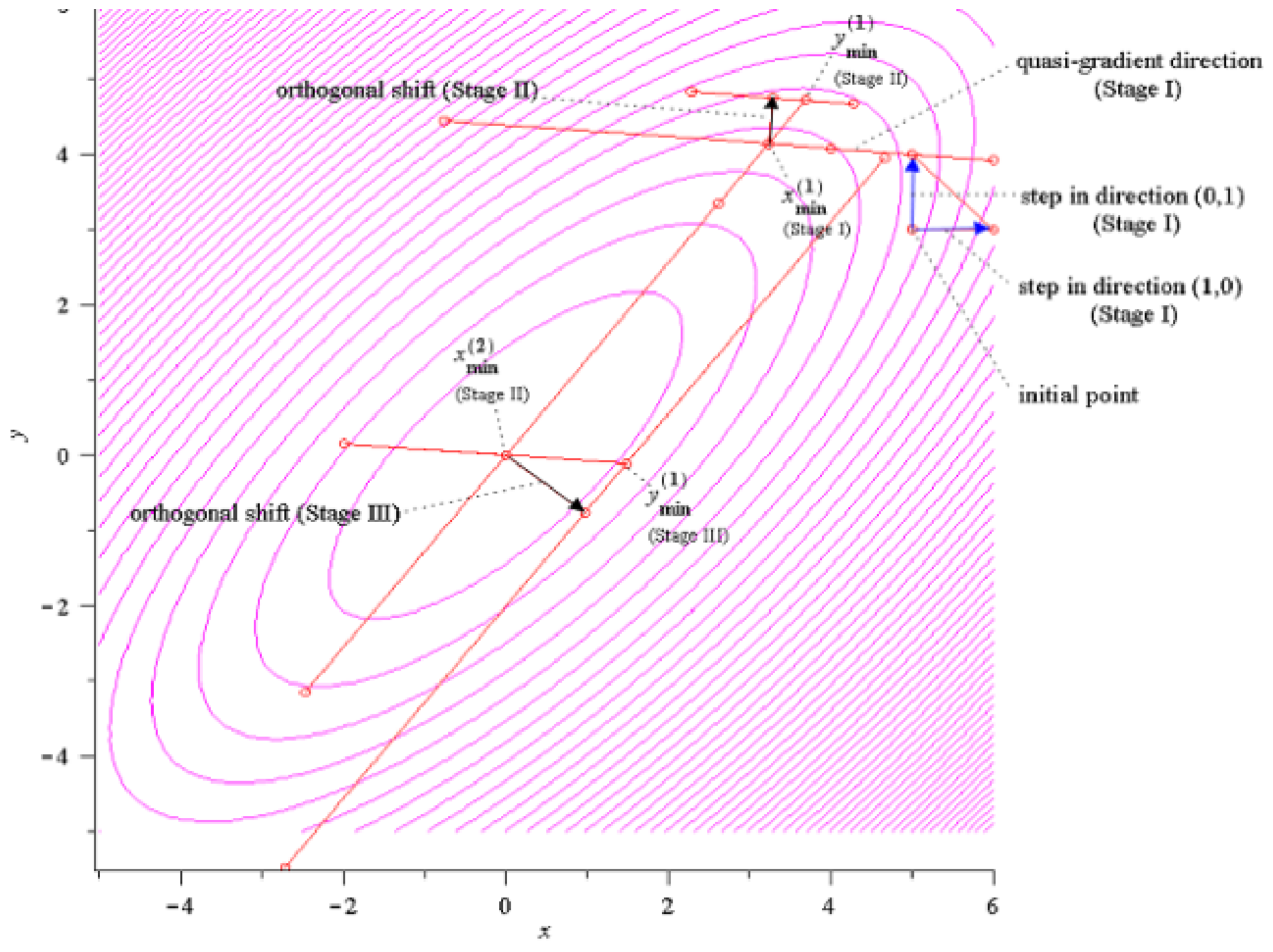
```
for j from 1 to n-1    # repeated movement in n-1 conjugate directions
    The single-dimensional search for minimum within this cycle shall be performed with a $3 \cdot L$ step.
    This is done because we have deviated from the current minimum.
    Therefore the typical scale of function variation has increased.

    Starting from point $\boldsymbol{y}^{(i)}_{\mathbf{min}}$ we perform a one-dimensional search for minimum in $\boldsymbol{u}^*_j$ direction.
    If the function value in the obtained point is less than the function value in point $\boldsymbol{y}^{(i)}_{\mathbf{min}}$, we update point $\boldsymbol{y}^{(i)}_{\mathbf{min}}$.

end for j.
```

Direction $\boldsymbol{u}_n$ is replaced by a direction connecting points $\boldsymbol{x}^{(i)}_{\mathbf{min}}$ and $\boldsymbol{y}^{(i)}_{\mathbf{min}}$ as follows: $\boldsymbol{u}_n = Normalize\left(\boldsymbol{x}^{(i)}_{\mathbf{min}} - \boldsymbol{y}^{(i)}_{\mathbf{min}}\right)$, if $f\left(\boldsymbol{x}^{(i)}_{\mathbf{min}}\right) < f\left(\boldsymbol{y}^{(i)}_{\mathbf{min}}\right)$, or $\boldsymbol{u}_n = Normalize\left(\boldsymbol{y}^{(i)}_{\mathbf{min}} - \boldsymbol{x}^{(i)}_{\mathbf{min}}\right)$ otherwise. This direction will be conjugate to all $\boldsymbol{u}_1, ..., \boldsymbol{u}_{n-1}$ previous conjugate directions. A one-dimensional search for minimum in $\boldsymbol{u}_n$ direction with step L is performed starting from point $\boldsymbol{x}^{(i)}_{\mathbf{min}}$, if $f\left(\boldsymbol{x}^{(i)}_{\mathbf{min}}\right) < f\left(\boldsymbol{y}^{(i)}_{\mathbf{min}}\right)$, or from point $\boldsymbol{y}^{(i)}_{\mathbf{min}}$ otherwise. The resulting minimum point becomes the current minimum point $\boldsymbol{x}^{(i+1)}_{\mathbf{min}}$ for the following cycle iteration.
The search step and orthogonal shift values are adjusted for the following iteration as follows:
$L = 0.32 \cdot \left\| \boldsymbol{x}^{(i+1)}_{\mathbf{min}} - \boldsymbol{x}^{(i)}_{\mathbf{min}} \right\| + 0.091 \cdot L$.
If $L = 0$, then $L = tolerances_1$, where $tolerances_1$ is a confidence interval for the extremum point. The orthogonal shift value is $L_s = 0.62 \cdot L$. If $L_s = 0$, then $L_s = L$.
We shall now check the following condition of search termination. If the following inequalities are satisfied twice in a row: $L \le tolerances_1$ and $f\left(\boldsymbol{x}^{(i)}_{\mathbf{min}}\right) - f\left(\boldsymbol{x}^{(i+1)}_{\mathbf{min}}\right) \le tolerances_2$, the search is stopped. Here $tolerances_2$ is confidence interval value for the extremum value. As a minimum estimate we use $f\left(\boldsymbol{x}^{(i+1)}_{\mathbf{min}}\right)$, and as a minimum point we use $\boldsymbol{x}^{(i+1)}_{\mathbf{min}}$. Otherwise we move on to the next search iteration.

- **end for i**.
- A one-dimensional search in case n>1 is carried out as follows. A step is made from the initial point in the search direction. If the step is successful, the extremum point is updated, the step is doubled and the search is continued in the same direction until we have the first unsuccessful step. Then a single parabolic approximation is performed for the last three points. As a minimum point we take a point where the minimum function value is achieved. If the first step is unsuccessful, we change the search direction and repeat this procedure.
- The plot below shows the real path of the following function minimum search $f(x, y) = x^2 + y^2 - 1.5 \cdot x \cdot y$ starting with initial point [x=5, y=3]. An option checkexit=1 was set because the function is quadratic and its minimum is achieved by the end of stage II.



- The above plot shows that the quadratic function minimum is achieved by the end of stage II. One iteration of stage III leads to the same minimum point obtained at stage II.
- **Optimization with constraints.** Non-greedy Search algorithm is well-adapted for search of extremum along the edges of constraints such as inequalities. When an unfeasible point is found during the one-dimensional search in some direction, the search step is decreased according to a certain rule until a feasible point meeting all constraints is obtained. Therefore the Search algorithm never calculates the function value in the unfeasible point. Moreover, if the point meets all the constraints but the function value in this point is not a real value, such point is also considered an unfeasible point.
- For equality constraints the Search algorithm uses the penalty function method with a quadratic penalty function.

Examples

> $with(DirectSearch):$

When a complex value is encountered, the **Search** tries to find a new feasible real point

> $f := x \to \sqrt{x}$

$$f := x \to \sqrt{x} \qquad (7.1)$$

> $Search(f)$

```
Warning, complex value encountered; trying to find a feasible point
```

$$\left[4.224890044617\ 10^{-8}, \left[\ 1.78496958891038702\ 10^{-15}\ \right], 144\right] \qquad (7.2)$$

but with constraints given explicitly the number of function evaluations is less

> $Search(f, [0 \le x])$

$$\left[4.224890044617\ 10^{-8}, \left[\ 1.78496958891038702\ 10^{-15}\ \right], 19\right] \qquad (7.3)$$

To increase reliability and accuracy one can increase **checkexit** or (and) **tolerances**

> $f := \sqrt{x+y} + x^2 + y^2;\ constr := [0 \le x + y]$

$$f := \sqrt{x+y} + x^2 + y^2$$
$$constr := [0 \le x + y] \qquad (7.4)$$

> $Search(f, constr); Search(f, constr, checkexit = 10); Search\left(f, constr, tolerances = 10^{-14}\right);$
$sol := Search\left(f, constr, checkexit = 10, tolerances = 10^{-14}\right)$

$$[0.0000481852771596812, [x = -0.00490360808372191162, y = 0.00490360808373084805], 124]$$
$$\left[8.32874378229308\ 10^{-9}, [x = -0.0000645319447339574692, y = 0.0000645319447339574692], 377\right]$$
$$\left[8.32874378218459\ 10^{-9}, [x = -0.0000645319447335371782, y = 0.0000645319447335371782], 390\right]$$

$$sol := [1.85460307534371\ 10^{-66}, [x = 9.62964972193617927\ 10^{-34}, y = -9.62964972193617927\ 10^{-34}], 714] \quad (7.5)$$

To increase reliability and accuracy one can also repeat search from the last extremum point estimation

> *sol* := *Search*(*f*, *constr*, *initialpoint* = sol_2)

$$sol := [1.77025887161799\ 10^{-96}, [x = 9.40813177952453019\ 10^{-49}, y = -9.40813177952453019\ 10^{-49}], 29] \quad (7.6)$$

To verify the solution and increase reliability and accuracy one can also increase **checksolution**, but this option is very time consumed

> *Search*(*f*, *constr*, *checksolution* = 10)

$$[1.48095176804396\ 10^{-7}, [x = 0.000089140217573204 3027, y = -0.0000891402175557266113], 4262] \quad (7.7)$$

Minimize Rosenbrock function for different initial points and initial steps and show search pathes

> $f := (x, y) \rightarrow 100\ (x^2 - y)^2 + (1 - x)^2$

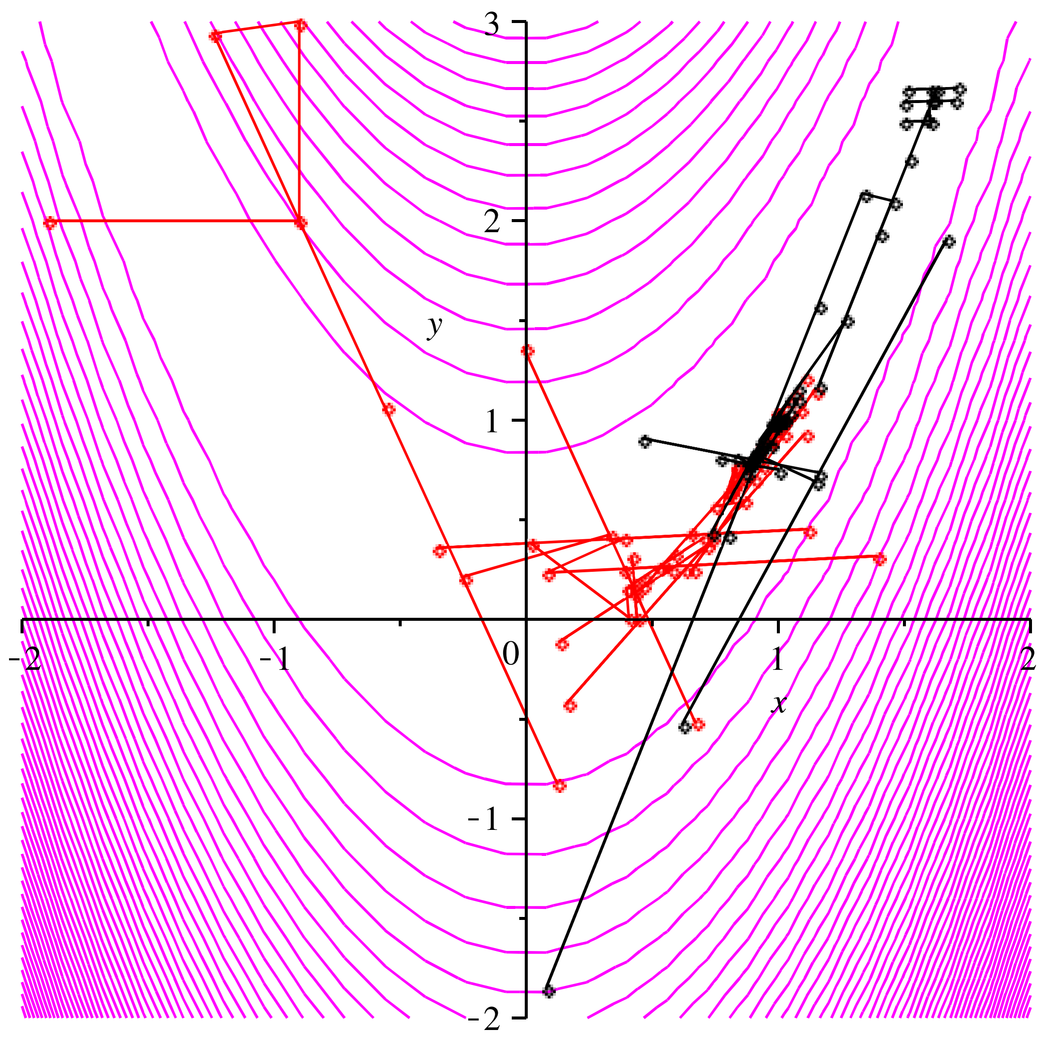
$$f := (x, y) \rightarrow 100\ (x^2 - y)^2 + (1 - x)^2 \quad (7.8)$$

> *Search*(*f*, *initialpoint* = [-1.9, 2], *step* = 1, *searchpath* = '*Path1*'); *Search*(*f*, *initialpoint* = [1.5, 2.5], *step* = 0.1, *searchpath* = '*Path2*')

$$\left[2.85062813591897\ 10^{-19}, \begin{bmatrix} 0.999999999973711794 \\ 0.999999999952070694 \end{bmatrix}, 133\right]$$

$$\left[2.341755563227\ 10^{-23}, \begin{bmatrix} 0.999999999999524702 \\ 0.999999999999058498 \end{bmatrix}, 104\right] \quad (7.9)$$

> *with*(*plots*) : *with*(*LinearAlgebra*) : *p0* := *contourplot*(*f* (*x*, *y*), *x* = -2 ..2, *y* = -2 ..3, *color* = *magenta*, *contours* = 50) :
p1 := *plot*(*Transpose* (*Path1*), *style* = *point*, *symbolsize* = 8, *color* = *red*) : *p11* := *plot*(*Transpose* (*Path1*), *color* = *red*) : *p2* :=
plot(*Transpose* (*Path2*), *style* = *point*, *symbolsize* = 8, *color* = *black*) : *p22* := *plot*(*Transpose* (*Path2*), *color* = *black*) : *display*(*p0*, *p1*,
p11, *p2*, *p22*)



Increase reliability and accuracy by increasing **checkexit**

> *Search*(*f*, *initialpoint* = [-1.2, 1], *checkexit* = 10)

$$\left[0., \begin{bmatrix} 1. \\ 1. \end{bmatrix}, 253\right] \quad (7.10)$$

When objective function **f** is a name of procedure the names in constraints must coincide with the procedure formal parameter names

> $f := (x, y) \rightarrow \frac{1}{\sqrt{x} - 1} + \ln(\sqrt{x} - 1) + \Gamma(x + y^2)$; $constr := [0 < \sqrt{x} - 1, 0 < x + y^2]$

$$f := (x, y) \rightarrow \frac{1}{\sqrt{x} - 1} + \ln(\sqrt{x} - 1) + \Gamma(x + y^2)$$

$$constr := [0 < \sqrt{x} - 1, 0 < x + y^2] \quad (7.11)$$

> *Search*(*f*, *constr*, *initialpoint* = [4, 4])

$$\left[2.43979073793139, \begin{bmatrix} 2.25726463557748992 \\ -8.83149941533719970\ 10^{-10} \end{bmatrix}, 66\right] \quad (7.12)$$

When objective function **f** is an expression the initial point must contain the equations *varname* = *value*

> $Search(f(x, y), constr, initialpoint = [x = 4, y = 4])$

$$[2.43979073793139, [x = 2.25726463557748992, y = -8.83149941533719970\ 10^{-10}], 66] \quad (7.13)$$

or one must provides option **variables** with the names of problem variables and its order

> $Search(f(x, y), constr, initialpoint = [4, 4], variables = [x, y])$

$$\left[2.43979073793139, \begin{bmatrix} 2.25726463557748992 \\ -8.83149941533719970\ 10^{-10} \end{bmatrix}, 66\right] \quad (7.14)$$

The constraints can include names of procedures that must accepts *n* floating-point parameters representing the problem variables

> $c1 := (x, y) \to \sqrt{x} - 1;\ c2 := (x, y) \to x + y^2$

$$c1 := (x, y) \to \sqrt{x} - 1$$
$$c2 := (x, y) \to x + y^2 \quad (7.15)$$

> $Search(f, [0 < c1, 0 < c2], initialpoint = [4, 4])$

$$\left[2.43979073793139, \begin{bmatrix} 2.25726463557748992 \\ -8.83149941533719970\ 10^{-10} \end{bmatrix}, 66\right] \quad (7.16)$$

The constraits can include both names of procedures and expressions involving the problem variables

> $c1 := (x, y) \to x;\ c2 := (x, y) \to y^2$

$$c1 := (x, y) \to x$$
$$c2 := (x, y) \to y^2 \quad (7.17)$$

> $Search\left(f, \left[0 < \sqrt{c1} - 1, 0 < x + c2\right], initialpoint = [4, 4]\right)$

$$\left[2.43979073793139, \begin{bmatrix} 2.25726463557748992 \\ -8.83149941533719970\ 10^{-10} \end{bmatrix}, 66\right] \quad (7.18)$$

See Also

DirectSearch, GlobalSearch, Optimization, NLPSolve