## Golden Section Search

### Theory

Golden section search is one of the many methods for 1D optimization. It is a modified version of Dichotomous search and an exact search method.

Given a unimodal function and an interval where is known to have a minimum in, golden section method tries to find an interval where the minimum lies and finally the minimum point with a pre-defined error by reducing the search interval at every iteration.

Main usage of golden section search is to solve the following problem to find a suitable step size for multivariable optimization:

Where is the current vector and is the search direction. However, one can use golden section search for any 1D minimization problem given that a suitable interval where the cost function is uniimodal. If the cost function is not unimodal in the given interval, the algorithm may converge to a local minimum.

### Usage

* is the cost function passed as a function argument.
* is an instance of class. Changeable properties are:

1. *exactLineSearchLowerBound*: Defines the lower bound of search interval (default ).
2. *exactLineSearchUpperBound*: Defines the upper bound of search interval (default ).
3. *exactLineSearchIntervalTolerance*: Defines the smallest possible search interval range. Once the search interval range is lower than *exactLineSearchIntervalTolerance,* golden section search returns the minimizer and minimums (default ).

## Steepest Descent

### Theory

The steepest descent method is one of the most well-known, easy to implement, and straightforward multivariable unconstrained optimization methods. It has fast convergence since the descent direction is identical with the gradient of the cost function, which is the direction of fastest change for the cost function at a given point.

At each iteration, suppose the current point is given by . Then the descent direction is directly found as:

Where is the cost function to be minimized.

Step size is found as follows:

The step size which minimizes is found via utilizing *gsSearch().*

Finally, minimum point vector is updated as:

After each iteration, is calculated. If it is less than a predefined toleranc, or reached iteration is greater than the maximum iterations allowed, iterations stop with current as the minimizer.

### Usage

* *init* is the initial point to start iterating.
* is the cost function passed as a function argument.
* is the gradient of the cost function passed as a function argument.
* is an instance of class. Changeable properties are:

1. *exactLineSearchLowerBound*: Defines the lower bound of search interval for the *gsSearch()* that is called inside the *steepestDescent()* to find the optimal step size (default ).
2. *exactLineSearchUpperBound*: Defines the uppper bound of search interval for the *gsSearch()* that is called inside the *steepestDescent()* to find the optimal step size (default ).
3. *exactLineSearchIntervalTolerance*: Defines the smallest possible search interval range for the *gsSearch()* that is called inside the *steepestDescent()* to find the optimal step size (default) .
4. *gradTolerance*: Defines the tolerance for the sqare of the norm of the gradient (default).
5. *maxIter*: Maximum allowed number of iterations (default).

## Conjugate Gradient Methods

Conjugate gradient methods is a family of optimization algorithms where descent directions are calculated by not only utilizing only the gradient, but also previous search directions. At each iteration, search direction bear a strict relationship with previous search directions. Efficiency and accuracy of these algorithms made them the most-widely used multivariable unconstrained optimization algorithms.

Conjugate gradient descent algorithms implemented here differs from each other only by means of descent directions. Therefore, they are introduced together.

### Theory

Let denote the iteration number, starting from . Descent direction is found as:

What differs different conjugate gradient algorithms is the calculation of . Algorithms implemented here are the following:

* Fletcher-Reeves (FR)
* Polak-Ribiere-Polyak (PRP)
* Hestenes-Stiefel (HS)
* Conjugate descent (CD)

Let us denote as . Then calculation of for different algorithms can be given as follows:

* FR:
* PRP:
* HS:
* CD:

Where denotes transpose.

Once descent direction is calculated, minimizer candidate is updated as follows:

Where is again the step size. Although it is possible to calculate by an exact line search, such as in the case of steepest descent, it is shown that Armijo-type line searches are generally sufficient. Armijo-type line search takes its roots from Armijo condition. Armijo condition states that for a minimizer candidate , a search direction   and a constant , if the following holds, is a sufficient step size:

Selection of can be accomplished as follows:

* Initialize (i.e. )
* Run the following loop:

**while**  **do:**

There are 3 parameters to determine in this Armijo-type line search. Initial value of to begin to descent from, and . determines how fast to descent. If is large, it takes a larger step to satisfy Armijo condition. determines how fast the step size diminishes at each of line search. As default values, , and . Note that both and should be in the range .

### Usage

* *init* is the initial point to start iterating.
* is the cost function passed as a function argument.
* is the gradient of the cost function passed as a function argument.
* is an instance of class. Changeable properties are:

1. *armijoLineSearchAlpha*: Defines for Armijo-type line search. Should be positive. (default = 1)
2. *armijoLineSearchBeta:* Defines for Armijo-type line search. Should be in the range . (default = 0.6)
3. *armijoLineSearchDelta:* Defines for Armijo-type line search. Should be in the range . (default = 0.05)
4. *gradTolerance*: Defines the tolerance for the sqare of the norm of the gradient (default).
5. *maxIter*: Maximum allowed number of iterations (default ).

## Spectral Gradient

### Theory

Spectral gradient algrotihm is essentially a steepest descent method without any line search. Elimination of line search also eliminates line search related parameters such as boundaries of the golden section line search, therefore making the algorithm less dependent on the problem.

As in the steepest descent, descent direction is equal to the negative of the gradient of the function at a given point:

However, step size at each iteration is calculated as follows:

Where and .

If we are at the first iteration such that and is the initial value that is assigned by user, and .

### Usage

* *init* is the initial point to start iterating.
* is the cost function passed as a function argument.
* is the gradient of the cost function passed as a function argument.
* is an instance of class. Changeable properties are:

1. *gradTolerance*: Defines the tolerance for the sqare of the norm of the gradient (default)
2. *maxIter*: Maximum allowed number of iterations (default).

## Newton Methods

### Theory

Newton methods are second order methods. They utilize a quadratic approximation of the Taylor series. In other words, not only the gradient but also the Hessian of the cost function is utilized. Newton methods in general generate search directions at each iteration as follows:

Where is the Hessian of the cost function. Rest of the algorithm is identical with gradient descent method:

Where

One of the main drawback of Newton methods is the case when Hessian is singular. In order to deal with the singularity of Hessian, various methods have been developed. One approach is developed by Goldfeld, Quandt and Trotter and it is referred as *gtqNewton()*. Another approach is developed by Matthews and Davies and referred as *mdNewton()*. These methods modify Hessian such that it contains direction information and not singular. Notice that an exact line search is needed. Therefore, as in the steepest descent method, golden-section line search is utilized.

For the *gtqNewton()* method, the Hessian is modified as follows:

Here, is the original Hessian and is identity matrix. Determination of is as follows:

* If is nonpositive definite, is large.
* Otherwise, is small.

Therefore, there are 2 parameters to be defined by the user. These are small and large values.

Matthews and Davies modification method does not require any parameter definitions.

### Usage

* *init* is the initial point to start iterating.
* is the cost function passed as a function argument.
* is the gradient of the cost function passed as a function argument.
* *hfunc* is the Hessian of the cosst function passed as a function argument.
* is an instance of class. Changeable properties for the golden section search are:

1. *exactLineSearchLowerBound*: Defines the lower bound of search interval for the *gsSearch()* to find the optimal step size (default ).
2. *exactLineSearchUpperBound*: Defines the uppper bound of search interval for the *gsSearch* to find the optimal step size (default ).
3. *GTQposdefBeta:* For the Goldfeld, Quandt and Trotter method, modification constant in the case Hessian is positive definite (default ).
4. *GTQnonposdefBeta:* For the Goldfeld, Quandt and Trotter method, modification constant in the case Hessian is nonpositive definite (default ).
5. *gradTolerance*: Defines the tolerance for the sqare of the norm of the gradient (default).
6. *maxIter*: Maximum allowed number of iterations (default ).

## Example 1

In this example, a very simple cost function is minimized with all the methods presented. The cost function is a two-variable function as follows:

Gradient of this function is as follows:

Finally, Hessian is as follows:

As initial point, zero vector is selected such that

As the *optimOptions,* an instance is created with default options:

*defaultOptions = minunc.optimOptions()*

The cost function is minimized with all the presented methods and results are printed to the secreen. One can analyze the results and compare them by means of accuracy.