

SCINET MATH **OPTIMIZATION**

Whitepaper Series



1. Introduction

Optimization involves finding the values of variables where the dependent function has a minimum or a maximum. Scinet Math supports optimization of one-variable functions and n-ary functions.

Following optimization methods are implemented within Scinet Math:

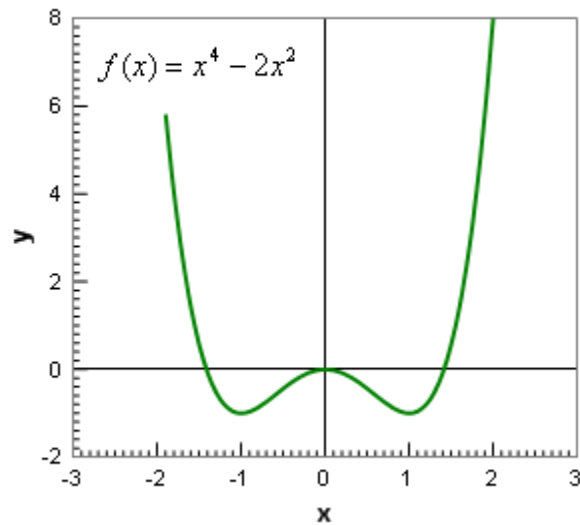
- The Bisection Method
- The Golden Section Search Method
- Newton's Method
- Quadratic Interpolation Method
- Cubic Interpolation Method
- Simplex Method



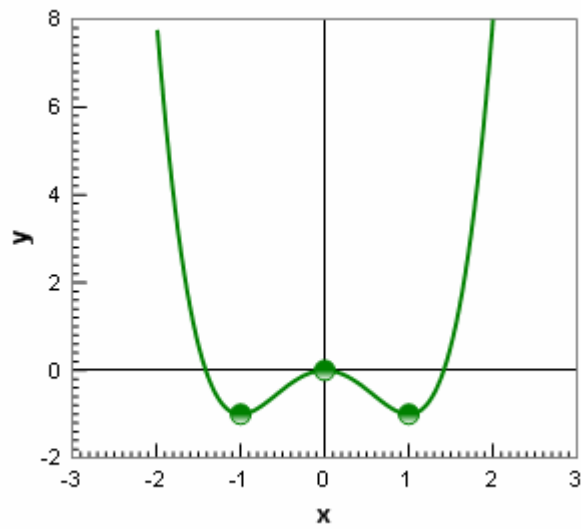
2. Optimization Methods

Scinet Math provides the `OptimaFinder` class in the `Scinet.Math.Optimization` namespace to compute the value of optima for the specified function and interval.

Throughout this technical article, we will use the following one-variable mathematical function to demonstrate the use of Scinet Math to compute the local minima and maxima:



As you can see the function has a local minima around $x = -1$ and $x = 1$ and a local maxima around $x = 0$.



Now, let's use Scinet Math to confirm this and compute the values of local minima and maxima.

The Bisection Method

The Bisection method for optimization starts with an interval that contains the optima and then halves that interval to zoom in on the optima.

The class `OptimaFinder` provides the following two methods to support the Bisection method.

```
static double BisectionMethod(Function f, double a, double b)
static double BisectionMethod(Function f, double a, double b, double tolerance)
```

where the parameters are:

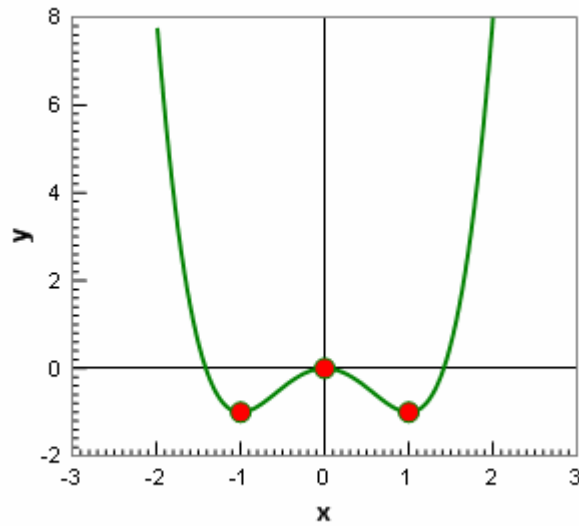
<i>f</i>	One-variable function to search for an optima.
<i>a</i>	Lower limit of the search interval.
<i>b</i>	Upper limit of the search interval.
<i>tolerance</i>	The tolerance factor to use for determining if the answer is converged close enough.

Example:

```
Console.WriteLine("Minima = {0}", OptimaFinder.BisectionMethod(f, -1.5, -0.5));  
Console.WriteLine("Minima = {0}", OptimaFinder.BisectionMethod(f, 0.5, 1.5));  
Console.WriteLine("Maxima = {0}", OptimaFinder.BisectionMethod(f, -0.5, 0.5));
```

The output is:

```
Minima = -0.99993896484375  
Minima = 0.99993896484375  
Maxima = -6.103515625E-05
```



The Golden Section Search Method

The Golden Section Search method for optimization is similar to the Bisection method. Instead of choosing the interval mean value, this method uses an interval reduction factor based on the Fibonacci numbers. This method is especially successful for finding the local minima. Not recommended for finding the local maxima.

The class `OptimaFinder` provides the following four methods to support the Golden Section Search method.

```
static double GoldenSearchMethod(Function f, double a, double b)
static double GoldenSearchMethod(Function f, double a, double b,
                                double tolerance)
static double GoldenSearchMethod(Function f, double a, double b,
                                double tolerance, int maxIteration)
static double GoldenSearchMethod(Function f, double a, double b,
                                int maxIteration)
```

where the parameters are:

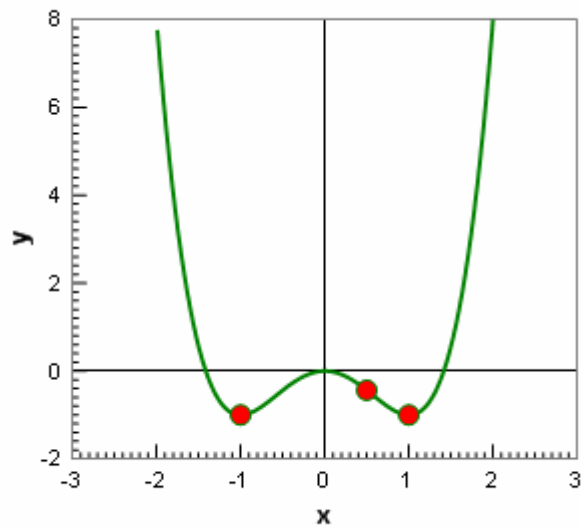
- f* One-variable function to search for an optima.
- a* Lower limit of the search interval.
- b* Upper limit of the search interval.
- tolerance* The tolerance factor to use for determining if the answer is converged close enough.
- maxIteration* Number of iterations to use.

Example:

```
Console.WriteLine("Minima = {0}", OptimaFinder.GoldenSearchMethod(f, -1.5, -0.5));  
Console.WriteLine("Minima = {0}", OptimaFinder.GoldenSearchMethod(f, 0.5, 1.5));  
Console.WriteLine("Maxima = {0}", OptimaFinder.GoldenSearchMethod(f, -0.5, 0.5));
```

The output is:

```
Minima = -0.999987374693828  
Minima = 0.999987374693828  
Maxima = 0.499966946519324
```



Newton's Method

Newton's method uses the fact that the derivative of the function at the local minima and/or maxima is zero.

The class `OptimaFinder` provides the following two methods to support the Newton's method.

```
static double NewtonMethod(Function f, double x)
static double NewtonMethod(Function f, double x, double tolerance)
static double NewtonMethod(Function f, double x, int maxIteration)
static double NewtonMethod(Function f, double x, double tolerance,
                           int maxIteration)
```

where the parameters are:

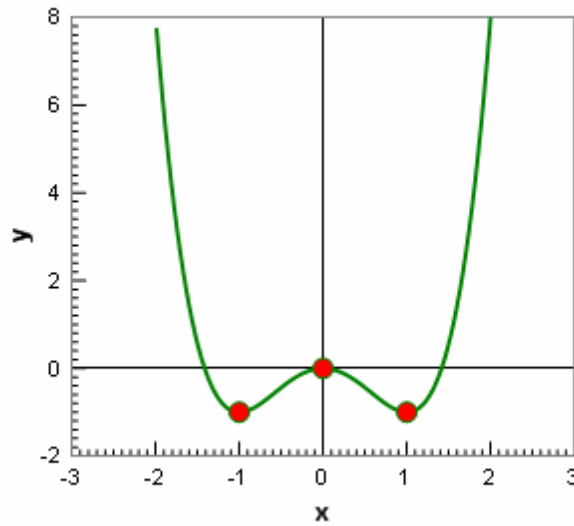
- f* One-variable function to search for an optima.
- a* Lower limit of the search interval.
- b* Upper limit of the search interval.
- tolerance* The tolerance factor to use for determining if the answer is converged close enough.
- maxIteration* Number of iterations to use.

Example:

```
Console.WriteLine("Minima = {0}", OptimaFinder.NewtonMethod (f, -1.5));  
Console.WriteLine("Minima = {0}", OptimaFinder.NewtonMethod (f, 1.5));  
Console.WriteLine("Maxima = {0}", OptimaFinder.NewtonMethod (f, 0.2));
```

The output is:

```
Minima = -1.00000000035018  
Minima = 1.00000000035018  
Maxima = -3.15761176396517E-09
```



The Quadratic Interpolation Method

The Quadratic Interpolation method zooms in on the optima to locate it, instead of reducing the interval that contains the optima. The method requires three initial guesses to compute the location of the local optima.

The class `OptimaFinder` provides the following four methods to support the Quadratic Interpolation method.

```
static double QuadraticInterpolationMethod(Function f, double a, double b,  
                                           double c)  
static double QuadraticInterpolationMethod(Function f, double a, double b,  
                                           double c, double tolerance)  
static double QuadraticInterpolationMethod(Function f, double a, double b,  
                                           double c, int maxIteration)  
static double QuadraticInterpolationMethod(Function f, double a, double b,  
double c, double tolerance, int maxIteration)
```

where the parameters are:

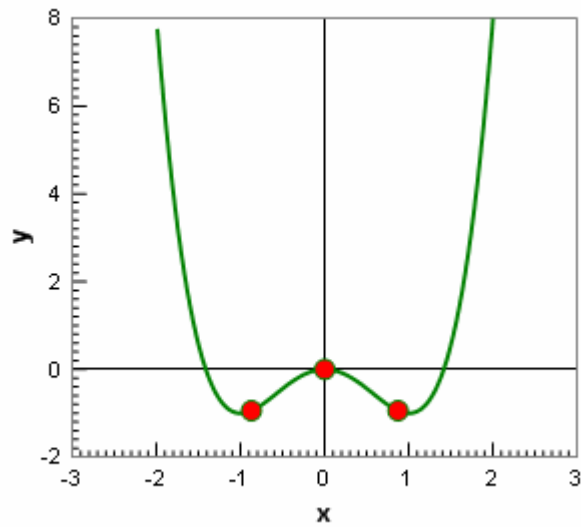
- f* One-variable function to search for an optima.
- a* Lower limit of the search interval.
- b* Upper limit of the search interval.
- tolerance* The tolerance factor to use for determining if the answer is converged
close enough.
- maxIteration* Number of iterations to use.

Example:

```
Console.WriteLine("Minima = {0}",  
    OptimaFinder.QuadraticInterpolationMethod(f, -1.5, -1.2, -0.5));  
Console.WriteLine("Minima = {0}",  
    OptimaFinder.QuadraticInterpolationMethod(f, 0.5, 1.2, 2));  
Console.WriteLine("Maxima = {0}",  
    OptimaFinder.QuadraticInterpolationMethod(f, -0.5, 0.2, 0.5));
```

The output is:

```
Minima = -0.870691511755034  
Minima = 0.799465722836153  
Maxima = 0
```



The Cubic Interpolation Method

The Cubic Interpolation method uses two initial guess to locate the minima. The method computes the slopes at the initial guesses to perform the interpolation. This method is designed to compute the local minima only.

The class `OptimaFinder` provides the following four methods to support the Cubic Interpolation method.

```
static double CubicInterpolationMethod(Function f, double a, double b)
static double CubicInterpolationMethod(Function f, double a, double b,
                                       double tolerance)
static double CubicInterpolationMethod(Function f, double a, double b,
                                       int maxIteration)
static double CubicInterpolationMethod(Function f, double a, double b,
                                       double tolerance,
                                       int maxIteration)
```

where the parameters are:

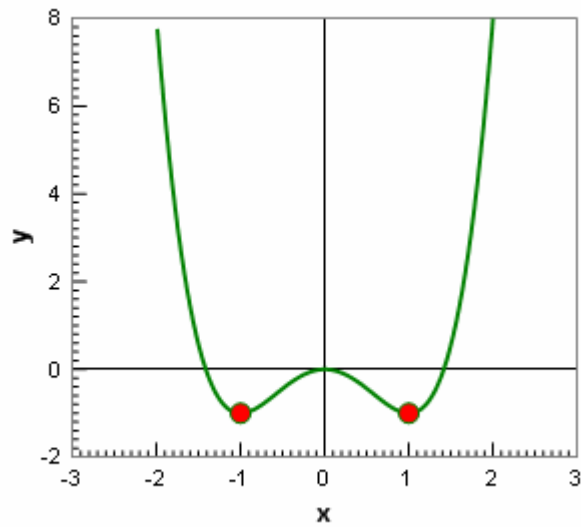
- f* One-variable function to search for an optima.
- a* Lower limit of the search interval.
- b* Upper limit of the search interval.
- tolerance* The tolerance factor to use for determining if the answer is converged close enough.
- maxIteration* Number of iterations to use.

Example:

```
Console.WriteLine("Minima = {0}",  
    OptimaFinder.CubicInterpolationMethod(f, -1.5, -0.5));  
Console.WriteLine("Minima = {0}",  
    OptimaFinder.CubicInterpolationMethod(f, 0.5, 1.5));
```

The output is:

```
Minima = -0.999999999999997  
Minima = 1
```



The Simplex Method

The Simplex method allows finding the local optima of n-ary (n-variable) functions. This method uses the function values at $n + 1$ points and locates the points with the best and the worst values. Then it replaces the worst value point with a better point by expanding and contracting the simplex near the worst-value point. The process continues until the multidimensional simplex shrinks around a minima or maxima point.

The class `OptimaFinder` provides the following four methods to support the Simplex method.

```
static void SimplexMethod( FunctionND f, ref double points)
static void SimplexMethod( FunctionND f, ref double points, double tolerance)
static void SimplexMethod( FunctionND f, ref double points, int maxIteration)
static void SimplexMethod( FunctionND f, ref double points, double tolerance,
                           int maxIteration)
```

where the parameters are:

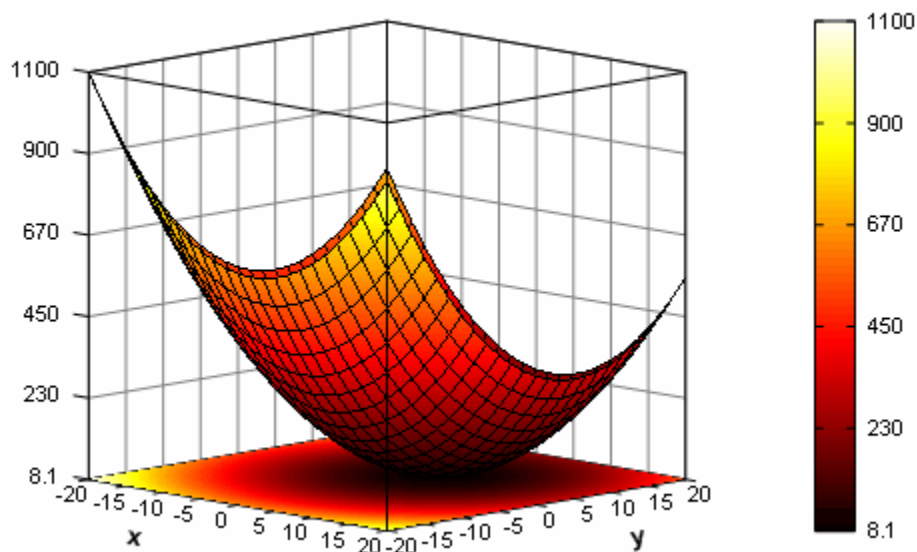
f N-variable function to search for an optima.
points Initial guesses
tolerance The tolerance factor to use for determining if the answer is converged close enough.
maxIteration Number of iterations to use.

Example:

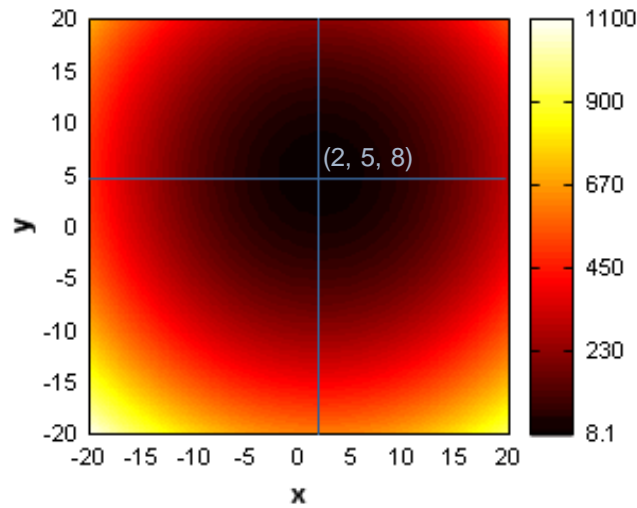
As an example, let's use the following two-variable function:

$$f(x, y) = (x - 2)^2 + (y - 5)^2 + 8$$

The following graph shows the function in x [-20, 20] and y [-20, 20] intervals.



Let's take a closer look at the density graph of this function:



As you can see the function has a minimum point at (2, 5, 8). Now, let's verify our findings with Scinet Math:

Let's use three initial guesses for the saddle point. We only need to guess the x and y values, such as (0.5, -0.5), (0.5, 1) and (1, 0):

```
double[,] points = new double[,] { { 0.5, -0.5 }, { 0.5, 1 }, { 1, 0 } };

OptimaFinder.SimplexMethod(fxy, ref points);
for (int i = 0; i < points.GetLength(0); i++)
{
    double x = points[i, 0];
    double y = points[i, 1];
    double z = fxy(new double[] { x, y });
    Console.WriteLine("{0} > {1}, {2}", z, x, y);
}
```

The output is:

```
8.00017876070579 > 2.00134256482124, 5.01330256462097
8.00021033753718 > 2.00329840183258, 4.98587703704834
8.00037997260392 > 1.98293352127075, 4.99058151245117
```

As you can see, for the specified initial guesses, the Simplex method finds the minimum point at (2, 5, 8).



3. Conclusions

Optimization is an essential component of numerical analysis. **Scinet Math** supports a variety of optimization methods.

For one-variable functions, Bisection method and the Newton's method are easier to use and produce more reliable results. These methods perform well for finding both the local minima and maxima. The results may vary by the choice of the initial guesses and or the interval specified. For n-variable functions the Simplex method is a great tool to solve the optimization problem.

As we have demonstrated in the previous sections, Scinet Math provides many great techniques to solve the complex optimization problems with just a few lines of source code.

Whitepaper Series

Scinet Math – Numerical Integration

Copyright © OBACS Corporation 2009. All rights reserved.

Printed in CANADA, June 2009.

 OBACS Corporation

Address 5625 Silverdale Dr. N.W.,
Calgary, Alberta, Canada T3B 4N5

Website <http://www.obacs.com>
Information info@obacs.com
Sales sales@obacs.com
Technical Support support@obacs.com