1. **Optimization : scipy.optimize**

Finding a numerical solution to minimization or equality is known as optimization.

For a wide range of business and technical processes, optimization is frequently the final frontier that must be won to create meaningful value. We demonstrate optimization using SciPy, Python's most famous scientific analysis tool and highlight special applications in the machine learning field.

The scipy.optimize package contains methods for scalar and multi-dimensional function reduction, curve fitting, and root discovery.

scipy.optimize package contains modules as below:

* Global optimization routines (like anneal(), basinhopping()).
* Unconstrained and constrained minimization of multivariate scalar functions (minimize()) with various algorithms (e.g. BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA or SLSQP).
* Multivariate equation system solvers (root()) using various algorithms (e.g. hybrid Powell, Levenberg-Marquardt or large-scale methods such as Newton-Krylov)
* Root finders (newton()) and Scalar univariate functions minimizers (minimize\_scalar()).
* Curve fitting (curve\_fit()) algorithms and Least-squares minimization (leastsq()).

For multivariate scalar functions in scipy.optimize , the minimize() function offers a common interface to unconstrained and constrained minimization techniques. Different available algorithms can be chosen for method parameter in minimize() function and optimization can be performed.

**Example 1 : Minimizing a simple scalar function sin(x).exp[(x-0.6)^2]**

*import numpy as np*

*from scipy import optimize*

*def scalar1(x):*

*return (np.sin(x)\*np.exp(-0.1\*(x-0.6)\*\*2))*

*x = np.arange(-9,10,0.03)*

*result = optimize.minimize\_scalar(scalar1)*

*print(result)*

*Output:*

*fun: -0.6743051024666711*

*nfev: 15*

*nit: 10*

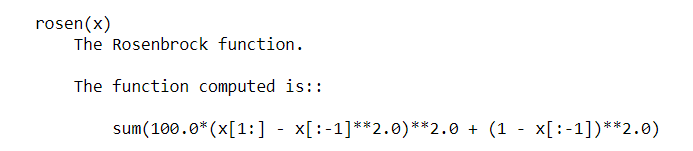
*success: True*

*x: -1.2214484245210282*

*x is the minima in the output*

**Example 2 : Rosenbrook function**

A test problem for gradient-based optimization techniques is the Rosenbrook function (rosen). In SciPy, it is defined as follows:



In the below example, minimize method with the Nelder Mead Algorithm is applied. Nelder Mead algorithm is used to find min/max of function in multidimensional space.

*from scipy import optimize*

*a = [2, 1.3, 3.9, 0.6]*

*b = optimize.minimize(optimize.rosen, a, method='Nelder-Mead')*

*print(b)*

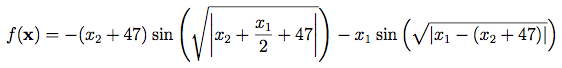
*print(b.x) #minima value*

**Example 3 :** [**Global optimization**](https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html#id41)

In the face of several local minima, global optimization seeks to identify the global minimum of a function within defined boundaries. Global minimizers typically search the parameter space effectively while employing a local minimizer (e.g., minimize) behind the scenes. A variety of good global optimizers are included in SciPy. We'll apply them to the same objective function, the (aptly titled) egg holder function, in this case as the function looks like an egg holder.

Optimizing the egg holder function and finding global minima is difficult to do manually as it has a lot of local minima but using scipy.optimize .

Global Optimization of eggholder function



*#eggholder function*

*def eggholder(x):*

*return (-(x[1] + 47) \* np.sin(np.sqrt(abs(x[0]/2 + (x[1] + 47))))*

*-x[0] \* np.sin(np.sqrt(abs(x[0] - (x[1] + 47)))))*

*bounds = [(-512, 512), (-512, 512)]*

*from scipy import optimize*

*results = dict()*

*results['shgo'] = optimize.shgo(eggholder, bounds) #shgo - finds global minimum of a function using SHG optimization.*

*results['shgo']*

**Example 4 : Least squares**

The least squares algorithm finds a local minimum of the cost function F given the residuals f(x) (an m-dimensional real function of n real variables) and the loss function rho(s) (a scalar function) (x).

Example : Find a minimum of the Rosenbrock function without bounds on the independent variables.

*#Rosenbrock Function*

*import numpy as np*

*from scipy.optimize import least\_squares*

*def fun\_rosenbrock(x):*

*return np.array([10 \* (x[1] - x[0]\*\*2), (1 - x[0])])*

*input = np.array([1, 2])*

*res = least\_squares(fun\_rosenbrock, input)*

*print (res)*

**Example 5 : Root finding**

The root-finding problem is one of the most fundamental numerical approximation problems. For a given function f, this procedure entails locating a root (or zero, or solution) of an equation of the type f (x) = 0.

Root Finding of the equation x2 + 2cos(x) = 0

*import numpy as np*

*from scipy.optimize import root*

*def func(x):*

*return x\*2 + 2 \* np.cos(x)*

*sol = root(func, 0.5)*

*print (sol)*