Optimization using numerical libraries

1 Overview

Question: Why do we calculate derivatives analytically if computers can calculate them numerically?

One answer to the above question is that numerical-optimization methods produce faster and more precise results when we provide them with the analytical derivative (or gradient, or Jacobian) of the objective function. This point is discussed further in these notes.

Analytically, the derivative of a scalar function f of a single variable x is given by:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f\left(x + \Delta x\right) - f\left(x\right)}{\Delta x}. \tag{1}$$

Depending on the type of function, the derivative receives different names such as *gradient* and *Jacobian*.

2 Optimization problems

In many applications, we need to calculate derivatives of functions (e.g., solving optimization problems). In fact, the derivative equations (i.e., in analytical form) may be passed as an input parameter to optimization functions from numerical libraries. While these library functions can calculate the required derivatives numerically, they solve optimization problems more efficiently (and more accurately) if the analytical derivatives are supplied (i.e., user-supplied gradients).

For example, consider the problem of finding the minimum of the Rosenbrock function[1] (Figure 1):

$$f(x, y) = (a - x)^2 + b(y - x^2)^2. \tag{2}$$

[1] `https://en.wikipedia.org/wiki/Rosenbrock_function`

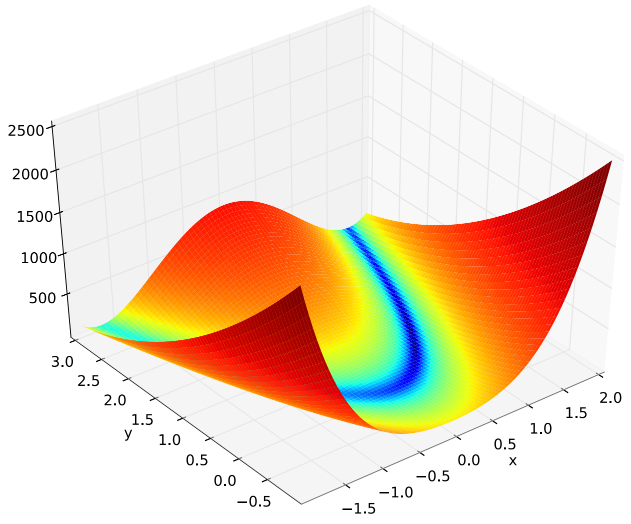

Figure 1: The Rosenbrock function. The global minimum is inside a long, narrow, parabolic shaped flat valley (Figure from `https://en.wikipedia.org/wiki/Rosenbrock_function`)

Its global minimum is at $(x, y) = (a, a^2)$, where $f(x, y) = 0$. The parameters are usually set to $a = 1$ and $b = 100$, i.e.:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2. \tag{3}$$

With this choice of parameters, the minimum is at $(1, 1)$. Indeed, this is the location of the minimum that is found by the following Python code using the `scipy.optimize` library[2]:

```
def f(x):   # The Rosenbrock function
...     return (1.0 - x[0])**2 + 100.0*(x[1] - x[0]**2)**2
>>> optimize.minimize(f, [2, -1], method="CG")
     fun: 1.6...e-11
     jac: array([-6.15...e-06,   2.53...e-07])
 message: ...'Optimization terminated successfully.'
    nfev: 108
     nit: 13
    njev: 27
  status: 0
 success: True
       x: array([0.99999...,  0.99998...])
```

To find the location of the minimum, the optimization function needs the derivatives (i.e., gradient, Jacobian). In the case of the Rosenbrock function from Equation 2, the derivative

[2]`https://scipy-lectures.org/advanced/mathematical_optimization/`

(gradient in this case) is given by:

$$\nabla f(x,y) = \begin{bmatrix} -2(1-x) - 400\,y\,(y-x^2) \\ 200(y-x^2) \end{bmatrix}. \tag{4}$$

While the derivatives can be computed numerically by the library function, the optimization process is faster (and more accurate) if the user provides the analytical gradient to the Python optimization function.

An example of passing the analytic derivative to the optimizer for finding the minimum of the Rosenbrock function is given by the following code:

```
def jacobian(x):
                Gx = -2.0*(1 - x[0]) - 400.0*x[0]*(x[1] - x[0]**2)
        Gy = 200.0*(x[1] - x[0]**2)
...     return np.array((Gx,Gy))
>>> optimize.minimize(f, [2, 1], method="CG", jac=jacobian)
     fun: 2.957...e-14
     jac: array([ 7.1825...e-07,  -2.9903...e-07])
 message: 'Optimization terminated successfully.'
    nfev: 16
     nit: 8
    njev: 16
  status: 0
 success: True
       x: array([1.0000...,  1.0000...])
```

When comparing the two optimization results, we can see that the second option is both more efficient (i.e., less function evaluations and less iterations) and more accurate (i.e., minimum was found at $[1,1]$ instead of at $[0.99999, 0.99999]$) than the first option.

Summary: Numerical-optimization methods produce faster and more precise results when we provide them with the analytical derivative (or gradient, or Jacobian) of the objective function.

References