Optimization in Python

Course 4, Module 7, Lesson 3

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Minimize Function

```
result = sp.minimize(objective_function, x_0, method='L-BFGS-B',
                     jac=objective_jacobian, bounds=bounds,
                     options={'disp' : True})
```

conjugate gradient

- Many optimization algorithms available Nelder-Mead , dogleg & BFGS
  - Specific one chosen depends on “method” parameter
- Model Jacobian passed in through “jac”
- Model constraints passed in through “constraints”
  - Different forms of constraints available
- x_0 gives initial guess for optimizer

Objective Function and Jacobian

- BFGS requires objective function as input, as well as a function to evaluate the Jacobian
- These functions will take a vector of the optimization variables as input

```
def objective_function(x):
    return x[0]**2 + 4*x[0]*x[1]

def objective_jacobian(x):
    return np.array([2*x[0] + 4*x[1], 4*x[0]])
```

Result

```
result = sp.minimize(objective_function, x_0, method='L-BFGS-B',
                     jac=objective_jacobian, bounds=bounds,
                     options={'disp' : True})
print(result.x)
```

- Upon completion, optimization returns a result variable
- The “x” member variable gives the final vector of optimized variables where the local minimum has been reached

Bounds

- Simplest constraints are inequality constraints on optimization variables, denoted as “bounds”

$$
\begin{aligned}
-10 \le x_0 \le 5 \\
-3 \le x_1 \le 4
\end{aligned}
$$

```
bounds = [[-10.0, 5.0], [-3.0, 4.0]]
```

- The i^{th} sub-list of the list denotes the upper and lower bounds for the i^{th} optimization variable
- Bounds are passed to “constraints” parameter in minimize function

Other Constraints

- Can also pass linear constraints and nonlinear constraints to optimizer depending on optimization algorithm
  - More details in SciPy documentation

$$\begin{bmatrix}-5\\1\end{bmatrix} \le \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix}\begin{bmatrix}x_0\\x_1\end{bmatrix} \le \begin{bmatrix}2\\4\end{bmatrix}$$

```
linear_constraint = LinearConstraint([[1, 2], [2, 1]], [2, 4])
```

- Can also combine different constraint methods in a list of constraints

Summary

- Introduced how to set up an optimization problem (L-BFGS) using SciPy
- Showed how to pass Jacobians and parameter bounds to the library’s optimizer

```
#nlopt

import scipy.optimize as sp
import numpy as np

bounds = [[-10.0, 5.0], [-3.0, 4.0]]
x_0 = [1.0, 1.0]

linear_constraint = LinearConstraint([[1, 2],
[2, 1]], [2, 4])

def objective_function(x):
    return x[0]**2 + 4*x[0]*x[1]

def objective_jacobian(x):
    return np.array([2*x[0] + 4*x[1], 4*x[0]])

result = sp.minimize(objective_function, x_0,
method='L-BFGS-B', jac=objective_jacobian,
bounds=bounds, options={'disp' : True})

print(result.x)
```

Solution:

```
>>> [-8.  4.]
```