Optimization with Scipy (1)

Intro to python scipy optimization module

Harry Lee
January 17, 2018
CEE 696
1. Introduction

2. scipy.optimize for local unconstrained optimization

3. Constrained Optimization
Introduction
Find values of the variable $x$ to give best (min or max) of an objective function $f(x)$ subject to any constraints (restrictions) $g(x), h(x)$

$$\min_x f(x)$$

subject to

$g_i(x) \geq 0, \ i = 1, \ldots, m$

$h_j(x) = 0, \ i = 1, \ldots, p$

$x \in X$

Assume $X$ be a subset of $\mathbb{R}^n$

$x : n \times 1$ vector of decision variables, i.e., $x = [x_1, x_2, \ldots, x_n]$

$f(x)$: objective function, $\mathbb{R}^n \to \mathbb{R}$

$g(x)$: $m$ inequality constraints $\mathbb{R}^n \to \mathbb{R}$

$h(x)$: $p$ equality constraints $\mathbb{R}^n \to \mathbb{R}$
My first example

Find values of the variable $x$ to give the minimum of an objective function $f(x) = x^2 - 2x$

$$\min_x x^2 - 2x$$

- $x$ : single variable decision variable, $x \in \mathbb{R}$
- $f(x) = x^2 - 2x$: objective function, $\mathbb{R} \rightarrow \mathbb{R}$
- no constraints

Thus, we are solving a single variable, unconstrained minimization problem.
import numpy as np
import scipy.optimize as opt

objective = np.poly1d([1.0, -2.0, 0.0])
print(objective)

x0 = 3.0
results = opt.minimize(objective, x0)
print("Solution: x=%f" % results.x)

import matplotlib.pylab as plt
x = np.linspace(-3, 5, 100)
plt.plot(x, objective(x))
plt.plot(results.x, objective(results.x), 'ro')
plt.show()
Objective function

- Objective function: minimize $f(x)$
- Maximize $f(x) = \text{Minimize } -f(x)$
- Examples
  1. Maximize total pumping rates $\sum Q_i$, $Q_i$: pumping rate at well $i$
  2. Minimize operation costs $\sum cQ_i$, $cQ_i$: operation cost at well $i$
Constraint set

• Simple bounds (box constraints): \( l_i \leq x_i \leq u_i \)
• Linear constraints
  \[ Ax = b \]
• Nonlinear constraints
  • inequality constraint \( g_i(x) \geq 0 \)
  • equality constraint \( h_i(x) = 0 \)

Optimization solution should be in a feasible region that satisfies all the constraints.
Optimization problems can be classified based on

- the type of constraints
- nature of the equations involved
- permissible value of the decision variables
- deterministic nature of the variables
- number of objective functions
Optimization problems can be classified based on the type of constraints

- Unconstrained optimization
- Constrained optimization
Optimization problems can be classified based on the permissible value of decision variables

- Discrete optimization
- Continuous optimization
Optimization problems can be classified based on the equations involved

- Linear programming
- Nonlinear programming
  - Quadratic programming
  - Geometry programming
  - Global optimization

programming = optimization
Optimization problems can be classified based on the deterministic nature of the decision variables

- Deterministic optimization
- Stochastic optimization
Optimization problems can be classified based on the number of objective functions

- singleobjective problem
- multiobjective problem
What information we have at hand

- function information e.g., \( f(x) \)
- Perhaps gradient \( f'(x) \)
- Hopefully Hessian \( f''(x) \)
Topics we will cover

• 1D optimization/Line search
• Local optimization
  • Steepest Descent
  • Newton, Gauss-Newton
  • Conjugate Gradient
• Linear Programming
• Global optimization
  • convex optimization
  • stochastic search/evolutionary algorithm
• Stochastic optimization (under uncertainty)
• Multi-objective optimization
• PDE-based optimization
• Recent developments
scipy.optimize for local unconstrained optimization
The `scipy.optimize` package provides several commonly used optimize algorithm.

`help(scipy.optimize)`

- Unconstrained and constrained minimization of multivariate scalar functions
- Global (brute-force) optimization routines
- Least-squares minimization, curve fitting
- Scalar univariate functions minimizers and root finders
- Multivariate equation system solvers
Let’s assume you know how to develop a general (black-box) optimization program. Then what inputs do you need?

- objective function
- constrain functions
- optimization method/solver
- additional parameters:
  - solution accuracy (numerical precision)
  - maximum number of function evaluations
  - maximum number of iterations
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)

- **fun** (callable) objective function to be minimized
- **x0** (ndarray) initial guess
- **args** (tuple, optional) extra arguments of the objective function and its derivatives (jac, hes)
- **method** (str, optional) optimization methods
- **jac** (bool or callable, optional) Jacobian (gradient)
- **hess, hessp** (callable, optional) Hessian (2nd-order grad.) and Hessian-vector product
- **bounds** (sequence, optional) bounds on x
- **tol** (float, optional) tolerance for termination
- **options** (dic, optional) method options
- **callback** (callable, optional) function called after each iteration
my_first_optimization.py again

\[
\min_x f(x^2 - 2x)
\]

```python
import numpy as np
import scipy.optimize as opt
import matplotlib.pylab as plt

objective = np.poly1d([1.0, -2.0, 0.0])

x0 = 3.0
results = opt.minimize(objective, x0)
print("Solution: x=%f" % results.x)

x = np.linspace(-3, 5, 100)
plt.plot(x, objective(x))
plt.plot(results.x, objective(results.x), 'ro')
plt.show()
```
Optimization result object

- **x** (ndarray) The solution of the optimization.
- **success** (bool) Whether or not the optimizer exited successfully.
- **status** (int) Termination status of the optimizer.
- **message** (str) Description of the cause of the termination
- **fun, jac, hess** Values of objective function, its Jacobian and its Hessian (if available)
  - **hess_inv** (object) Inverse of the objective function’s Hessian; Not available for all solvers
- **nfev, njev, nhev** (int) Number of evaluations of the objective functions and of its Jacobian and Hessian
- **nit** (int) Number of iterations performed by the optimizer
- **maxcv** (float) The maximum constraint violation.
```python
def objective(x, coeffs):
    return coeffs[0] * x**2 + coeffs[1] * x + coeffs[2]

x0 = 3.0
mycoeffs = [1.0, -2.0, 0.0]
myoptions = {'disp': True}
results = opt.minimize(objective, x0, args=mycoeffs,
                        options = myoptions)

print("Solution: x=\%f" % results.x)

x = np.linspace(-3, 5, 100)
plt.plot(x, objective(x, mycoeffs))
plt.plot(results.x, objective(results.x, mycoeffs), 'ro')
plt.show()
```
Constrained Optimization
\[
\min_x \quad f(x^2 - 2x)
\]
subject to \quad x - 2 \geq 0

objective = np.poly1d([1.0, -2.0, 0.0])
cons = ({'type': 'ineq',
         'fun': lambda x: np.array([x[0] - 2])})
results = opt.minimize(objective, x0=3.0, constraints = cons,
                        options = {'disp':True})

- constraint is defined in a dictionary with type, fun, jac, args (extra arguments for fun and jac)
- Here we use lambda function for its brevity (but not recommended, use def).
min \begin{align*}
x \quad f(x^2 - 2x)
\end{align*}
subject to \quad x - 2 \geq 0

objective = np.poly1d([1.0, -2.0, 0.0])
bnds = ((2, None),) # tuple for 1D box constraint
results = opt.minimize(objective, x0=3.0, bounds=bnds, options = {'disp':True})