Global Optimization

for practical engineering applications

Harry Lee

4/9/2018

CEE 696

Table of contents

Global Optimization

Global optimization

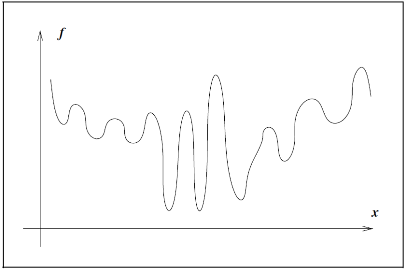

Figure 1: Fig 2.2 from Nocedal & Wright [2006]

- We have learned “local” optimization methods
- How can we solve the optimization problem like one with objective function in Figure 1?

When global optimization is needed?

- We saw Newton's method converges fast to the (local) optimum
- What if our objective function is very complex with many local optima as in Figure 1?
- One solution might be the one we start multiple initial guesses
- In many nonlinear optimization problems, the objective function f has a large number of local minima and maxima.
- Furthermore, f may be non-differentiable and non-continuous.
- Global optimization finds the maximum or minimum over all input values, as opposed to finding local minima or maxima.

Two ways to solve global optimization

1. Formulate/modify a global optimization problem into a tractable one with local optimization (e.g., convex optimization)
2. Use global optimization methods

Here, we focus on global optimization methods for general optimization problems.

If time is allowed, we will cover convex optimization.

Python Scipy optimization routines

Equation (Local) Minimizers

leastsq(func, x0[, args, Dfun, full_output, ...])	Minimize the sum of squares of a set of equations.
least_squares(fun, x0[, jac, bounds, ...])	Solve a nonlinear least-squares problem with bounds on the variables.
nnls(A, b)	Solve `argmin_x || Ax - b ||_2` for `x>=0`.
lsq_linear(A, b[, bounds, method, tol, ...])	Solve a linear least-squares problem with bounds on the variables.

Global Optimization¶

basinhopping(func, x0[, niter, T, stepsize, ...])	Find the global minimum of a function using the basin-hopping algorithm
brute(func, ranges[, args, Ns, full_output, ...])	Minimize a function over a given range by brute force.
differential_evolution(func, bounds[, args, ...])	Finds the global minimum of a multivariate function.

Rosenbrock function

rosen(x)	The Rosenbrock function.
rosen_der(x)	The derivative (i.e.
rosen_hess(x)	The Hessian matrix of the Rosenbrock function.

Global optimization algorithms

- deterministic approach vs. stochastic approach
- stochastic, metaheuristic approaches are popular in engineering because they are easy to use
    - Simulated Annealing
    - Evolutionary Computation
        - Genetic Algorithm (GA)
        - Differential Evolution
    - particle swarm optimization, ant colony optimization and so on

Global optimization algorithms - stochastic approach

- Smart random search
- Computational cost (number of objective function evaluation) is high
- They are random, i.e., optimization result can change in every run
- Easy to implement - your numerical simulation considered as black box
- No guarantee for convergence.

Simulated Annealing (Basin Hopping in scipy.optimize)

- A probabilistic technique for approximating the global optimum of a given function
- Name comes from metal annealing, heating and cooling of a material to increase the size of its crystals and reduce their defects
- Adapted from Metropolis-Hastings algorithms in 1953, which is used to generate sample states of thermodynamic system
- “basinhopping” in scipy.optimization (“simulated annealing” has been deprecated)

Simulated Annealing (Basin Hopping in scipy.optimize)

- A metal is heated to a high temperature
- The metal is gradually cooled on a specific schedule you specify
- As the metal cools, its atoms settle into an optimal (thermal-equilibrium) crystalline structure
- Annealing improves the cold-working properties of metal

Simulated Annealing

1. Start with initial guess
2. Calculate energy E of initial guess (i.e., objective value)
3. Set initial temperature T
4. While T > cutoff/stopping T
    4.1 find a test solution
    4.2 Calculate E of the solution (i.e., objective value)
    4.3 δ = obj(test) - obj(previous solution)
    4.4 if δ < 0: update solution and obj
    4.5 elif $\exp(-\frac{\delta}{T}) > \mathit{random.uniform}(0, 1)$: update solution and obj
    4.6 decrease T
5. End

Mechanisms in Simulated Annealing

- Early in the search, SA explores the decision variable space
- As the search progresses, SA refines the search

How does it work?

- https://www.youtube.com/watch?v=iaq_Fpr4KZc

Scipy.optimize.basinhopping

```
scipy.optimize.basinhopping(func, x0, niter=100, T=1.0,
                      stepsize=0.5,minimizer_kwargs=None
                      take_step=None, accept_test=None,
                      callback=None, interval=50,
                      disp=False, niter_success=None)
```

func Function to be optimized

x0 initial guess

T temperature parameter for the accept or reject criterion

minimizer_kwargs arguments for local optimization routine

take_step user-defined step-taking algorithm

accept_test user-defined step-acceptance algorithm

callback function called for all minimum found

interval interval for how often to update the stepsize

Scipy.optimize.basinhopping

Download and run the example script
`http://www2.hawaii.edu/~jonghyun/classes/S18/CEE696/files/example_basinhopping.py`

Genetic Algorithm (GA)

- Developed by John Holland in 1975, many developments since then
- applications to real world problems
- Mimics mechanics of natural selection and genetics

Genetic Algorithm

1. Create Initial population
2. Selection individuals
3. Crossover & Mutation
4. Evaluation Fitness function
5. Repeat 2 - 5 until converges

GA - Chromosome representation & initial population

- Let's assume our decision variable x is an integer $\in [0, 127]$ for an optimization problem.
- with $x' = bin(x)$, i.e., string represented in 6-length binary
- For example, x' can be 000000 (0), 000001 (1), 000010 (2), 000011 (3), $\cdots$, 111111 (127).
- Then generate "n" number of random strings (chromosome). This is our initial population.
- For example, we have an initial population of 6 parents:
- 001000, 010010, 101110, 000101, 100000, 010111
- Their fitness (of survival) is determined by user-defined objective function, i.e., obs(x)

GA operators - Crossover & Mutation

Now we are generating offsprings of initial population. We are mating parent: 101110, 000101

Crossover: choose one or two points in the chromosome

1. One point crossover: 101110, 000101 => 101101, 000110
2. Two point crossover: 101110, 000101=> 100110, 001101

Mutation:

1. Bit inversion 1 => 0, 0 => 1

Create n offspring for the new generation and repeat this step until the maximum number of iterations.

Scipy.optimize.differential_evolution

GA is similar to differential evolution algorithm and python offers differential_evolution

```
differential_evolution(func, bounds, args=(),
           strategy='best1bin', maxiter=1000,
           popsize=15, tol=0.01, mutation=(0.5, 1),
           recombination=0.7, seed=None,
           callback=None, disp=False, polish=True,
           init='latinhypercube', atol=0)
```

strategy The differential evolution strategy to use

popsize A multiplier for setting the total population size. The population has popsize * len(x) individuals.

mutation user-defined step-taking algorithm

recombination crossover probability

seed random seed for pseudo random algorithm

init population initialization method (latinhypercube or random)

Differential Evolution

```
from scipy.optimize import rosen, differential_evolution
bounds = [(0,2), (0, 2), (0, 2), (0, 2), (0, 2)]
result = differential_evolution(rosen, bounds)
print(result.x)
%(array([1., 1., 1., 1., 1.])
print(result.fun)
%1.9216496320061384e-19
```

Scipy.optimize.brute

More like deterministic optimization technique

```
scipy.optimize.brute(func, ranges, args=(),
                     Ns=20, full_output=0,
                     finish=<function fmin>,
                     disp=False)
```

ranges grid points or bounds for function evaluations

Ns a number of grid points if bounds are defined in ranges

finish a local optimization can be called with the result of brute force minimization as initial guess

Exercise - groundwater supply optimization

Use global optimization approaches for our previous examples

basinhopping:

`https://www2.hawaii.edu/~jonghyun/classes/S18/CEE696/files/opt_max_pumping_basinhopping.py`

differential differential_evolution:

`https://www2.hawaii.edu/~jonghyun/classes/S18/CEE696/files/opt_max_pumping_DE.py`