Educational framework for constructing black box optimization methods based on Python modules

Nadia Udler - Fordham University; Joel Castle, Eli Udler - University of Connecticut

Introduction

Many pressing real world problems can be stated as problems of global optimization (or black box optimization). They range from robot parameter tuning and biological system analysis to strategy optimization in electronic trading systems and optimization of chemical processes. For such objectives (with many extrema) local optimization methods (so called smooth optimization methods that are based on the information about derivative of objective function) will not work.
Our approach for constructing global optimization methods is similar to smooth optimization theory but instead of derivatives of objective function the derivatives of so called potential functions are used.
The detailed derivation of methodology for constructing nonlocal optimization methods is described in [1], and minimum set of their building blocks is obtained. This methodology doesn't specify the exact optimization algorithm. However it is shown how to obtain the extensions of well known optimization methods (Nelder and Mead, Tunnelling, Simulated Annealing, Covariance Matrix Adaptation, Natural Gradient evolution strategy) based on the generic scheme. At the same time, new algorithms can be constructed using this approach by varying certain parameters.
Real life problems are best approached with a library of optimization methods to help study the nature of the problem. The advantage of having a library of methods with many adjustable parameters is that methods can be compared in a timely manner and the best one (for the given problem) in terms of speed and precision can be selected.
We suggest creating a library of modules based on the new methodology. Such a library can be based on standard optimization software (for example, Python scipy.optimize or pyOpt). By varying parameters of the standard optimization procedures and making hybrids we obtain the whole universe of methods. The new approach helps to understand the nature of the parameters of iterative optimization algorithms in general.
This framework can be a foundation of a whole curriculum for studying data mining/machine learning methods.

References

[1] Kaplinskii A.I., Propoi A.I., First-order nonlocal optimization methods that use potential theory, *Automation and Remote Control* , 1994, №7, pp.94-103.
[2] Bruce Eckel, Python 3. Patterns, Recipes and Idioms.

Smooth optimization reminder

General form of iterative optimization algorithm can be defined using 2 parameters, direction of search *and step-size.*

$$X^q = X^{q-1} + \alpha^q S^q$$

q - iteration number, S – vector search direction, α – step size,

X_0 – initial solution

The algorithm determines the search direction S according to some criteria.

Gradient-based algorithms use gradient information to decide where to move.

Gradient-free algorithms use sampling and/or heuristics.

Consider a function J(X), $X = \{x_1, x_2, \ldots x_n\}$

The gradient of J(X) at point X^0 is a vector of length n

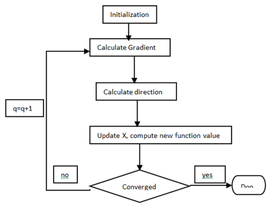
$$\nabla J(X^0) = \begin{bmatrix} \frac{\partial J}{\partial x_1}(X^0) \\ \frac{\partial J}{\partial x_2}(X^0) \\ \cdots \\ \frac{\partial J}{\partial x_n}(X^0) \end{bmatrix}$$

Hassian matrix: Consider a function J(X), $X = \{x_1, x_2, \ldots x_n\}$

The second derivative of J(X) at a point X^0 is a matrix of size nXn

$$H(X^0) = \nabla^2 J(X^0) = \begin{bmatrix} \frac{\partial^2 J}{\partial x_1^2}(X^0) & \frac{\partial^2 J}{\partial x_1 \partial x_2}(X^0) & \ldots & \ldots & \frac{\partial^2 J}{\partial x_1 \partial x_n}(X^0) \\ \frac{\partial^2 J}{\partial x_2 \partial x_1}(X^0) & \ldots & & & \\ \ldots & & & & \\ \ldots & & & & \\ \frac{\partial^2 J}{\partial x_n \partial x_1}(X^0) & & & & \frac{\partial^2 J}{\partial x_n^2}(X^0) \end{bmatrix}$$

Gradient-based optimization process:



Black box optimization

- Construct machine learning strategies based on systematic approach.
- Obtain the library of essential modules , or building blocks (that comprise the foundation of any optimization strategy)
- We would like to optimize a function that can be non smooth or given by values only algorithmically defined (**black box** function)

$$f(x) \to min$$

Main steps of the approach include function randomization

$$F(X) = E(f(x)) \to min$$

- Derivative of the target functional $F(X)$ in the direction of the random vector Y at the point X^0 is :

$$\delta_Y F(X^0) = \left[\frac{d}{d\varepsilon} F(X^0 + \varepsilon Y)\right]_{\varepsilon=0} = \left[\frac{d}{d\varepsilon} F(X^\varepsilon)\right]_{\varepsilon=0} = \left[\frac{d}{d\varepsilon} \int_{R^n} f(x) p_{X^\varepsilon}(x) dx\right]_{\varepsilon=0}$$

where $p_{X^\varepsilon}(x)$- density function of the random vector X^ε

Directional derivative of randomized functional would define the direction of the search on the next iteration. See [1] for the full derivation of the approach.

Building Blocks

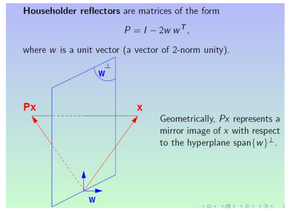
- Householder transformation (compare to reflections in Nelder and Mead algorithm),
- Shor operator (operator of space dilation)
- Rotation of the coordinate system (in variable metric algorithms, such as Covariance Matrix Adaptation and Shor r-algorithm, to achieve separability needed for parallel computations)
- Automatic differentiation (computing derivatives of algorithmically defined functions, and thus applying methods of smooth optimisation to non smooth functions).
- Newtonian potential
- Smoothing: independent (convolution) and dependent variables
- MLE/Stochastic approximation
- Natural Gradient – substitute for gradient in CMA , Natural Gradient evolution strategies, and other stochastic algorithms

The *Householder transformation* is considered a basic building block of many well-known optimization methods

Example 1: **Nelder and Mead** algorithm (a deterministic algorithm) :iteration step – Householder reflection

Example 2: **Shor r-algorithm** (deterministic) – uses an extension of the Householder transformation called space dilation operator $R_a(r) = I + (a-1)rr^t$ and uses subgradients (if gradient of objective function is not available).

Example 3. **Covariance Matrix Adaptation Evolution Strategy** (stochastic algorithm). Distribution of points on each iteration is updated according to space dilation



Example 4. **Natural gradient Evolution Strategy** (stochastic algorithm) the notion of "gradient" in the iteration updates of the following form:

$$R_a(r) = I + 2ww^t$$

is replaced with "natural gradient" and "stochastic gradient."
(In natural gradient the notion of distance in Euclidean sense is replaced with *Kullback Leibler distance* between distributions)

Example 5: **Particle Swarm Optimization algorithm** (population based algorithm): the notion of ***particle*** and its ***speed*** is used in place of gradient.

We build a foundation for the whole curriculum in data science/machine learning topics

Python Modules

- AlgoPy/Autograd(automatic differentiation)
- Tensorflow/Google(computational graph)
- PyOpt.SolvOpt (Shor's r-algorithm)
- SciPy (scientific subroutines)
- Scikit Learn (machine learning)
- Orange(machine learning visualization)

Simulation engine

```
# The strategy interface:
class FindMinima:
    # Find the sequence of points:
    def algorithm(self, line) : pass

# The various strategies:
class LeastSquares(FindMinima):
    def algorithm(self, line):
        return [ 1.1, 2.2 ] # Dummy

class NewtonsMethod(FindMinima):
    def algorithm(self, line):
        return [ 3.3, 4.4 ] # Dummy

class Bisection(FindMinima):
    def algorithm(self, line):
        return [ 5.5, 6.6 ] # Dummy

class ConjugateGradient(FindMinima):
    def algorithm(self, line):
        return [ 3.3, 4.4 ] # Dummy

# The "context" controls the strategy:
class MinimaSolver:
    def __init__(self, strategy):
        self.strategy = strategy

    def minima(self, line):
        return self.strategy.algorithm(line)

    def changeAlgorithm(self, newAlgorithm):
        self.strategy = newAlgorithm

solver = MinimaSolver(LeastSquares())
line = [1.0, 2.0, 1.0, 2.0, -1.0, 3.0, 4.0, 5.0, 4.0]
print(solver.minima(line))
solver.changeAlgorithm(Bisection())
print(solver.minima(line))
```

See [2]

Glossary

The Euclidean distance between 2 points is the length of the segment connecting them

$$d(p,q) = d(q,p) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+(q_3-p_3)^2\ldots+(q_n-p_n)^2}$$

Fisher information matrix (FIM) - a matrix that is used to measure the amount of information that a random variable *X* carries about an unknown parameter *ϑ* of a distribution that models *X*
X - observable random variable, *ϑ* - unknown parameter which alters the probability of *X*
given *N* parameters, *ϑ* becomes a *N* x 1 vector, and the Fisher information matrix is *N* x *N*
$[I(\theta)]_{i,j} = \mathrm{E}[(\frac{\delta}{\delta\theta_i} log\, f(X;\theta))(\frac{\delta}{\delta\theta_j} log\, f(X;\theta)|\theta]$ note: this is a positive, semidefinite symmetric matrix

Householder transformation - a linear transformation that describes a reflection about a plane containing the origin $P = I - 2vv^H$ where I is the identity matrix, v is the unit vector and H is a Hermitian transpose

vector field - for a function $F(x,y) = P(x,y,z)i + Q(x,y,z)j + R(x,y,z)k$, a function of a space whose value at each point is a vector quantity.

divergence - $div\ F = \nabla \cdot F = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \cdot (P,Q,R) = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}$ where F is a continuously differentiable vector field and F = *P*i + *Q*j + *R*k.

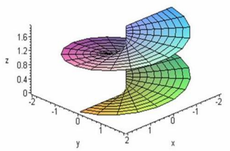
Newtonian potential a vector field F(x) is a potential filed if there is a function φ : Ω → R, such that F(x) = ∇φ. The function φ is called the potential

gradient - a multivariable generalization of the derivative; a vector-valued function that is the equivalent of a derivative but for multivariable equations
note: the gradient is the slope of the tangent lines of a multivariable function, it takes different forms depending on the system
$(\nabla f(x)) \cdot v = D_v f(x)$ where ∇ is the differential operator and v is any vector

$$\nabla f = \frac{\delta f}{\delta x} i + \frac{\delta f}{\delta y} j + \frac{\delta f}{\delta z} k$$

Jacobian - the matrix for all first-order partial derivatives of a vector-valued function
$J_{ij} = \frac{\partial f_i}{\partial x_j}$ note: when the matrix is square, both the matrix and its corresponding determinant are called the "Jacobian"

Hessian - a square matrix of second-order partial derivatives of a scalar-valued function
$H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ this is related to the Jacobian matrix by H(f(x)) = J(∇f(x))ᵀ



Expression for directional derivative that determines the optimization algorithms for smooth and non smooth optimization.

First representation is the source of smooth optimization methods

$$\delta_Y F(X) = \int_{R^n} \left(\frac{\partial f(x)}{\partial x}, \tilde{y}(x)\right) p(x)\, dx = -\int_{R^n} (f(x) - c)\, \mathrm{div}[p(x)\tilde{y}(x)]\, dx.$$

Second representation is the source of non smooth optimization methods