Introduction to Local and Global Optimization for NLP

Marco Trubian

Dipartimento di Scienze dell'Informazione (DSI)
Università degli Studi di Milano

Calculating Derivatives

Most algorithms for nonlinear optimization and nonlinear equations require knowledge of derivatives.

- Sometimes the derivatives are easy to calculate by hand, and it is reasonable to expect the user to provide code to compute them
- In other cases, the functions are too complicated or they are given as Black Boxes, so we look for ways to calculate or approximate the derivatives automatically
- A number of interesting approaches are available, of which the most important are probably Finite Differencing

Forward-difference

- A popular formula for approximating the partial derivative $\frac{\partial f}{\partial x_i}$ of a smooth function f at a given point $\boldsymbol{x}$ is the forward-difference, or one-sided-difference, approximation, defined as

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}) \approx \frac{f(\boldsymbol{x} + \varepsilon e_i) - f(\boldsymbol{x})}{\varepsilon}$$

  where ε is a small positive scalar and e_i is the i-th unit vector
- This process requires evaluation of f at the point $\boldsymbol{x}$ as well as the n perturbed points $\boldsymbol{x} + \varepsilon e_i$, $i = 1, 2, \ldots, n$: a total of $(n+1)$ points.
- the introduced error is $O(\varepsilon)$
- when the ratio of function values to second derivative values does not exceed a modest size, the following choice of ε is fairly close to optimal:

$$\varepsilon = \sqrt{u}$$

  where u is a bound on the relative error that is introduced whenever an arithmetic operation is performed on two floating-point numbers (u is about $1.1 \times 10^{-}16$ in doubleprecision IEEE floating-point arithmetic.)

Central-difference formula

- A more accurate approximation to the derivative can be obtained by using the central-difference formula

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}) \approx \frac{f(\boldsymbol{x} + \varepsilon e_i) - f(\boldsymbol{x} - \varepsilon e_i)}{2\varepsilon}$$

- This process requires evaluation of f at the point $\boldsymbol{x}$ as well as the $2n$ perturbed points $\boldsymbol{x} \pm \varepsilon e_i$, $i = 1, 2, \ldots, n$: a total of $2n + 1$ points.
- the introduced error is $O(\varepsilon^2)$
- when the ratio of function values to second derivative values does not exceed a modest size, the following choice of ε is fairly close to optimal:

$$\varepsilon = u^{1/3}$$

Approximating the hessian

- We can obtain the Hessian by applying the techniques described above to the gradient ∇f. This approach ignores symmetry of the Hessian. We can recover symmetry by adding the approximation to its transpose and dividing the result by 2
- For the case in which gradients are not available, we can derive formulae for approximating the Hessian that use only function values

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}) \approx \frac{f(\boldsymbol{x} + \varepsilon e_i + \varepsilon e_j) - f(\boldsymbol{x} + \varepsilon e_i) - f(\boldsymbol{x} + \varepsilon e_j) + f(\boldsymbol{x})}{\varepsilon^2}$$

- If we wished to approximate every element of the Hessian with this formula, then we would need to evaluate f at $\boldsymbol{x} + \varepsilon(e_i + e_j)$ for all possible i and j (a total of $n(n+1)/2$ points) as well as at the n points $\boldsymbol{x} + \varepsilon e_i$, $i = 1, 2, \dots, n$.

Other techniques: Automatic differentiation

Global Optimization

The object of Global Optimization (GO) is to find a solution of a given non-convex mathematical programming problem.

- By solution we mean here a global solution, as opposed to a local solution; i.e., a point where the objective function attains the optimal value with respect to the whole search domain.
- We require the objective function and/or the feasible region to be nonconvex because in convex mathematical programming problems every local optimum is also a global one. Consequently, any method solving a convex problem locally also solves it globally.

Example

Minimum energy configuration for compex molecules.

- The Lennard-Jones model for clusters of Argon, Krypton, Nickel, Gold.
- The energy of a couple of athoms in position $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^3$, at a r distance from each other, is given by the difference of two components:

$$v(r) = \frac{1}{r^{12}} - \frac{2}{r^6}$$

- The energy of a clauster of N athoms in positions $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^3$, is given by

$$\min f(\boldsymbol{x}) = \sum\nolimits_{i=1}^{N} \sum\nolimits_{j=1}^{i-1} \left(\frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^{12}} - \frac{2}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^{6}} \right)$$

- The number of local (not global) optima has an exponential growth with N.

Global Optimization

Most GO algorithms are two-phases.

- The solution space S is explored exhaustively in the global phase, which iteratively identifies a promising starting point $\boldsymbol{x}$.
- In the local phase, a local optimum $\boldsymbol{x}^*$ is found starting from each $\boldsymbol{x}$. The local phase usually consists of a deterministic local descent algorithm which the global phase calls as a black-box function.
- The global phase can be stochastic or deterministic.
    - Algorithms with a stochastic global phase are usually heuristic algorithms,
    - whereas deterministic global phases often provide a certificate of optimality, making the algorithm precise.

Stochastic global phases

Stochastic global phases identify the starting points $\boldsymbol{x}$

- either by some kind of sampling in S (sampling approach), or
- by trying to escape from the basin of attraction of the local minima $\boldsymbol{x}^*$ found previously (escaping approach), or
- by implementing a blend of these two approaches.

Stochastic global phases do not offer certificates of optimality of the global optima they find, and they usually only converge to the global optimum with probability 1 in infinite time.
In practice, though, these algorithms are very efficient, and are the only viable choice for solving reasonably large-scale NLPs.
The efficiency of stochastic GO algorithms usually depends on the proper fine-tuning of the algorithmic parameters controlling intensification of sampling, extent of escaping and verification of termination conditions.

Deterministic global phases

- Deterministic global phases usually work by partitioning S into smaller sets $S_1, \ldots, S_p$. The problem is then solved globally in each of the subsets S_j.
- The global solution of each restriction of the problem to S_j is reached by recursively applying the global phase to each S_j until a certificate of optimality can be obtained for each S_j.
- The certificate of optimality is obtained by computing upper and lower bounds, *ub*, *lb*, to the objective function value.
- A local optimum $\mathbf{x}^*$ in S_j is considered global when $|ub - lb| < \varepsilon$, where $\varepsilon > 0$ is a (small) constant.
- The certificate of optimality for the global optimum of the problem with respect to the whole solution space S is therefore really a certificate of ε-global optimality.
- Such global phases are called Branch-and-Select
  - the partitioning of the sets S_j is called branching;
  - the algorithm relies on selection of the most promising S_j for the computation of bounds.
- Deterministic algorithms perform well on small and medium-scale problems.
- Their efficiency seems to depend strongly on the particular instance of the problem at hand, and on the algebraic formulation of the problem.

The presence of Global Information

$$\min f(\boldsymbol{x}) = \sum_{i=1}^{N} \sum_{j=1}^{i-1} \left(\frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^{12}} - \frac{2}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^{6}} \right)$$

This problem has a strong structure.
The function

$$f(r_{12}, r_{13}, \ldots, r_{N-1,N}) \to \sum \frac{1}{r_{ij}^6} - 2 \sum \frac{1}{r_{ij}^3}$$

is the difference of two convex function: d.c. programming

Reformulation of d.c. problems

The unconstrained d.c. problem can be transformed in an equivalent problem with a linear objective function and two constraints:

$$\min g(\boldsymbol{x}) - h(\boldsymbol{x})$$

with g, h convex, is equivalent to

$$\begin{array}{ll} \min & z \\ s.t. & g(\boldsymbol{x}) - h(\boldsymbol{x}) \leq z \end{array}$$

which is equivalent to

$$\begin{array}{ll} \min & z \\ s.t. & g(\boldsymbol{x}) \leq w \\ & h(\boldsymbol{x}) + z \geq w \end{array}$$

For the last model there are suitable cutting plane techniques

Multistart

Multistart (MS) algorithms are conceptually the most elementary GO algorithms: many local descents are performed from different starting points.

- Starting points are sampled with a rule that is guaranteed to explore the solution space exhaustively (in infinite time)
- the local minimum with the best objective function value is considered the global optimum.
- MS algorithms are stochastic GO algorithms with a sampling approach.

Main problem: the same local optimum is identified many times when the sampling rule picks starting points in the basin of attraction of the same local optimum.

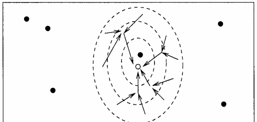
Multi Level Single Linkage

Clustering: to inhibit multiple local descents to start in the same basin of attraction.

- Sampled starting points are grouped together in clusters of nearby points, and only one local descent is performed in each cluster
- In the Multi Level Single Linkage (MLSL) method: a point $\boldsymbol{x}$ is clustered together with a point $\boldsymbol{y}$ if $\boldsymbol{x}$
    - is not too far from $\boldsymbol{y}$ and
    - $f(\boldsymbol{y}) < f(\boldsymbol{x})$.
- The clusters are then represented by a directed tree, the root of which is the designated starting point
- As the number of problem variables increases, the sampled points are further apart and cannot be clustered together so easily.

MS algorithms for GO usually perform rather well on medium to large scale problems.

MS with random and quasi-random sampling is, to date, the most promising approach to solving the Lennard-Jones potential energy problem



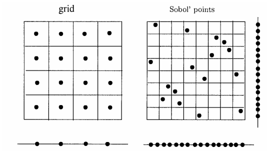
MS and Sobol sequences I

A deterministic sampling rule employs Low-Discrepancy Sequences (LDSs) of starting points called Sobol' sequences whose distributions in Euclidean space have very desirable uniformity properties.

- Uniform random distributions where each point is generated in a time interval are guaranteed to be uniformly distributed in space in infinite time with probability 1.
- Sobol' sequences are guaranteed to be distributed in space as uniformly as possible even in finite time.
- For any integer N $>$ 0, the first N terms of a Sobol' sequence do a very good job of filling the space evenly.
- Any projection on any coordinate hyperplane of the Euclidean space $\mathbb{R}^n$ containing N n-dimensional points from a Sobol' sequence will still contain N projected $(n-1)$-dimensional Sobol' points.

MS and Sobol sequences II

Sobol' sequences has been used by GO methods to successfully solve the Kissing Number Problem (KNP - determining the maximum number of non- overlapping spheres of radius 1 that can be arranged adjacent to a central sphere of radius 1) up to 4 dimensions.



Variable Neighbourhood Search

VNS escapes from the current local minimum $\boldsymbol{x}^*$ by initiating other local searches from starting points sampled from a neighbourhood of $\boldsymbol{x}^*$ which increases its size iteratively until a local minimum better than the current one is found.

VNS for box-constrained NLPs: the neighbourhoods arise naturally as hyperrectangles of growing size centered at the current local minimum $\boldsymbol{x}^*$.

- $k = 1$, choose random point $\boldsymbol{x}_k$, perform local descent to find a local minimum $\boldsymbol{x}^*$
- **while** $k < k_{max}$
    - define a neighbourhood $N_k(\boldsymbol{x}^*)$;
    - sample a random point $\boldsymbol{x}$ from $N_k(\boldsymbol{x}^*)$;
    - perform local descent from $\boldsymbol{x}$ to find a local minimum $\boldsymbol{x}'$;
    - **if** $f(\boldsymbol{x}') < f(\boldsymbol{x}^*)$ $\boldsymbol{x}^* = \boldsymbol{x}'$ and $k = 0$;
    - $k = k + 1$;

If $N_k(\boldsymbol{x})$ is taken to be a hyperrectangle $H(\boldsymbol{x})$ of size k centered at $\boldsymbol{x}$ we take $N_k(\boldsymbol{x}) = H_k(\boldsymbol{x}) \setminus H_{k-1}(\boldsymbol{x})$.

Spatial Branch-and-Bound

Spatial Branch-and-Bound (sBB) algorithms are the extension of traditional Branch-and-Bound (BB) algorithms to continuous solution spaces.

They are termed spatial because they successively partition the Euclidean space where the problem is defined into smaller and smaller regions where the problem is solved recursively by generating converging sequences of upper and lower bounds to the objective function value.

Spatial Branch-and-Bound

1. (Initialization) Initialize a list $\mathcal{L} = \{R\}$ where R comprises the entire set of variable ranges. Set $U := \infty$; $\boldsymbol{x}^* = \emptyset$;
2. (Choice of Region) If $\mathcal{L} = \emptyset$ then Stop; else remove the region R with lowest associated lower bound from $\mathcal{L}$.
3. (Lower Bound) Generate a convex relaxation of the original problem in the selected region R and solve it to obtain an underestimation lb_R of the objective function with corresponding solution $\boldsymbol{x}$. If $lb > U$ or the relaxed problem is infeasible, go back to step 2.
4. (Upper Bound) Solve the original problem in R with a local minimization algorithm to obtain a locally optimal solution $\boldsymbol{x}'$ with $u = f(\boldsymbol{x}')$.
5. (Pruning) If $U > u$ then $\boldsymbol{x}^* = \boldsymbol{x}'$; $U := u$. Delete all regions R' in $\mathcal{L}$ which have $lb_{R'} > U$.
6. (Check Region) If $u - lb_R < \varepsilon$, accept u as the global minimum for R and return to step 2;
7. (Branching) Apply a branching rule to R to split it into sub-regions. Add these to $\mathcal{L}$, assigning to them an (initial) lower bound. Go back to step 2.

Convex relaxation I

The main algorithmic difference among different sBB is the way the convex relaxation is derived.
The standard automatic way to generate a convex relaxation consists in linearizing all non convex terms in the objective function and constraints and then replacing each nonconvex definition constraint with the respective upper concave and lower convex envelopes.
Convex and concave envelopes are suggested for various types of fractional terms.
E.g a convex underestimator for the term $\frac{x}{y}$, where $x \in [x^L x^U]$ and $y \in [y^L, y^U]$ are strictly positive, is as follows:

$$\begin{aligned} z &\geq \frac{x^L}{y_a}(1-\lambda) + \frac{x^U}{y_b}\lambda \\ y^L &\leq y_a \leq y^U \\ y^L &\leq y_b \leq y^U \\ y &= (1-\lambda)y_a + \lambda y_b \\ x &= x^L + (x^U - x^L)\lambda \\ 0 &\leq \lambda \leq 1 \end{aligned}$$

Convex relaxation II

The convex relaxation of general twice-differentiable nonconvex terms is carried out by using a quadratic underestimation (based on the α parameter). Quadratic underestimations work for any twice-differentiable nonconvex term, but are usually very slack.
A function $f(\boldsymbol{x})$ is underestimated over the entire domain $[\boldsymbol{x}^L, \boldsymbol{x}^U] \in \mathbb{R}^n$ by the function $L(\boldsymbol{x})$ defined as follows:

$$L(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{n} \alpha_i (x_i^L - x_i)(x_i^U - x_i)$$

where the α_i are positive scalars that are sufficiently large to render the underestimating function convex.

Convex relaxation III

$$L(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{n} \alpha_i (x_i^L - x_i)(x_i^U - x_i)$$

- this kind of underestimator does not introduce any new variable or constraint
- Since the sum $\sum_{i=1}^{n} \alpha_i (x_i^L - x_i)(x_i^U - x_i) < 0$, $L(\mathbf{x})$ is an underestimator for $f(\mathbf{x})$. Furthermore, since the quadratic term is convex, all nonconvexities in $f(\mathbf{x})$ can be overpowered by using sufficiently large values of the α_i parameters.
- $L(\mathbf{x})$ is convex if and only if its Hessian matrix $H_L(\mathbf{x})$ is positive semi-definite.

$$H_L(\mathbf{x}) = H_f(\mathbf{x}) + 2\Delta$$

where $\Delta \equiv \text{Diag}(\alpha_i)$