Scipy: Large-scale Constrained Optimization

Antônio Horta Ribeiro

Sub-organization: Scipy

Student Info

- Name: Antônio Horta Ribeiro
- GitHub: antonior92
- Email: antonior92@gmail.com
- Website: antonior92.github.io
- CV: https://www.dropbox.com/s/3qgyuasajp9pjpy/cv_8.pdf?dl=0
- Tel.: +55 (31) 99741-9213
- Timezone: GMT-3

- GSoC Blog RSS Feed URL: http://hortaribeiro.wordpress.com/?feed=rss

University Info

- University Name: Federal University of Minas Gerais (Brazil)
- Major: Electrical Engineering
- Current Year: 2nd year
- Degree: Master

Code Sample

I have contributed to Scipy optimization and signal processing packages in the following pull requests.

- ENH: signal: added irrnotch and iirpeak functions. #6404 — https://github.com/scipy/scipy/pull/6404 (merged)
- ENH: optimize: added iterative trustregion. #6919 — https://github.com/scipy/scipy/pull/6919 (merged)
- ENH: optimize: changes to make BFGS implementation more efficient. #7165 — https://github.com/scipy/scipy/pull/7165 (merged)

Both pull requests: #6919 and #7165, showcase my optimization skills.

Project Info

Title:
"***Scipy: Large-scale Constrained Optimization***"

Abstract:

The project consists of implementing in Scipy an interior-point method for nonlinear problems. The main goal is to implement a constrained optimization algorithm able to deal with a large (and possibly sparse) problems for which the constrained optimization methods currently implemented in Scipy (namely `SLSQP` and `COBYLA`) are largely inappropriate to deal with. Implement benchmark problems and integrate quasi-Newton (namely BFGS, SR1 and L-BFGS) and finite differences (using graph coloring schemes to deal with sparse structures) approximations to the method are also part of the project.

Description:

This project intends to deal with the problem of minimizing a function subject to constraints on its variables, mathematically formulated as the following nonlinear programming (NLP) problem:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & c_E(x) = 0 \\ & c_I(x) \geq 0, \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is a vector of unknowns and the objective function f, the equality constraints c_E and the inequality constraints c_I are twice continuous differentiable functions. In science, engineering and finance many applications can be formulated as the above optimization problem.

The solvers for problem (1) can be classified into three groups, described next (together with a list of the main optimization softwares that uses each approach):

(i) Augmented Lagrangian methods (MINOS [**?**] and LANCELOT [**?**]);

(ii) SQP (Sequential Quadratic Programming) methods (SNOPT [**?**], FILTERSQP [**?**] and KNITRO/ACTIVE [**?**]);

(iii) Interior-point methods (KNITRO/DIRECT, KNITRO/CG [**?**], IPOPT [**?**], LOQO [**?**] and BARNLP [**?**]).

While the first widely available softwares for large-scale optimization (MINOS and LANCELOT) adopted augmented Lagrangian methods, more modern softwares usually adopt interior-point or SQP formulations. The existing benchmarks [**?**, **?**, **?**] don't offer a single approach as the best for all cases and SQP and interior-point formulations often appears as competing state-of-the-art approaches with complementary strengths. According to Nocedal and Wrigth [**?**, p.560 and p.592], SQP methods are the most efficient if the number of constraints is nearly as large as the number of variables and are more robust to badly scaled problems, while interior point methods show their strength in large-scale applications, where they often (but not always) outperform SQP methods. The rapid pace of software development, however, makes it difficult to arrive at concrete conclusions about the classes of problems each approach is best suited for.

Since Scipy already have a SQP method (`SLSQP`) available, an interior point method seems the most sensible choice for this project. Furthermore, interior-points methods often have a good performance in large-scale applications [**?**, p.592], while maintaining a competitive performance for medium/small problems as well [**?**].

The algorithm to be implemented in this project is the interior-point trust-region method described in [**?**]. This implementation has been chosen because of its ability to deal with large problems with, possibly, sparse structure. The trust-region approach allows great freedom in the choice of the Hessian and provides a mechanism for coping with Jacobian and Hessian singularities. Furthermore, it is a

widely tested method, being the algorithm implemented in the commercial package KNITRO/CG [?]; There are comprehensive references about the algorithm and its variations (namely [?, ?, ?]); And, while the algorithm is not very new (it was first described in 1999), it is still object of active search (improvements in the infeasibility detection of this algorithm have just been published in 2014 [?]).

Schedule and Deliverables:

Project milestones are described next.

Further Literature Review — Before the official code begins

- The success of this project is largely depending on a deep knowledge about the methods being implemented and about optimization theory in general. So I intend is to read as much as possible about the theme before the official code period begins so things go as smoothly as possible during the implementation.

Prepare for the Project and Interact with Mentors — Community bounding period

- I already have some familiarity with `scipy.optimize` internal structure from previous contributions. That way, my main concern during the bounding period is to interact with my mentors and figure out small details about the algorithms I intend to implement.

Equality-constrained Quadratic Programming — 1 week

- The interior point method [?] need to solve, in one of its sub-steps, an equality-constrained quadratic programming (EQP) problem:

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x^T H x + g^T x \\ \text{subject to} \quad & Ax = b. \end{aligned} \tag{2}$$

  Nocedal and Wright [?, Sec. 16.1 to 16.3] describe several methods for solving this problem. Among them I intend to implement two: the so-called projected conjugate gradient (CG) method and the null-space method. Both of them can deal with indefinite matrix H and have complementary strengths: the projected CG method is an iterative method, it have low memory-storage requirements and may provide an approximate solutions to the problem within few iterations, being a good choice for large-scale problems; and, the null-space method is a direct factorization method, it provide accurate solutions and may be a better choice for small/medium-sized problems. Both methods can be adapted to deal with sparse structures.

Trust-region Interior Point Method — 3 to 5 weeks

- At this stage of the project, I intend to implement, document and test the trust-region interior point method described in [?]. The general idea of the method is explained next.

  Consider the barrier problem:

$$\begin{aligned} \min_{x} \quad & f(x) - \mu \sum \log s_i \\ \text{subject to} \quad & c_E(x) = 0 \\ & c_I(x) = s. \end{aligned} \tag{3}$$

  One need not to include inequality constraints because the barrier term $-\mu \sum \log s_i$ prevents the components of s from becoming close to zero (Because $(-\log s_i) \to \infty$ when $s_i \to 0$) and guarantee the inequality constraints are being satisfied. Interior-point methods finds approximate solutions to the above barrier problem (3) for a sequence of positive barrier parameters $\{\mu_k\}$ that converges to zero. And, as $\mu \to 0$ the solution of the barrier problem approach the solution of the NLP problem (1).

The interior point method to be implemented solve the barrier problem (3) by sequentially solving a series of EQP problems subjected to trust-region constraints. The EQP solvers implemented during the previous weeks are needed as sub-steps of the method.

Benchmarks — 1 or 2 weeks

- It is expected that small tests and problems will be tried along the development of the algorithm. At this stage, however, I intend to benchmark the algorithm in a more rigorous manner, implementing a subset of the problems from CUTEst [**?**] in Python and testing the implemented method on it. Furthermore, I intend to compare the method with other available optimization software. For instance, IPOPT [**?**] package already have a wrapper for Python, so it should be really easy to compare with it.

Preconditioners — 1 week

- Preconditioning (described in [**?**, pp. 118-120]) is a method of improving the convergence of CG implementations. Currently there are not any preconditioner implemented in Scipy and I intend to implement and test preconditioners for the projected CG method. This will improve the performance of the interior-point methods, since the projected CG is one of its sub-steps.

- Both `trust-ncg` and `Newton-CG` could benefit from the implemented preconditioner, therefore I intend to include preconditioners options on both.

Finite-difference Approximations — 2 weeks

- The implemented algorithm require the first and second derivatives of functions and constraints. What may not easy to compute in some cases, so it is very useful to have some automatic way to estimate these derivatives. At this stage of the project I intend to implement finite-difference methods to estimate derivatives.

  Consider the definition of partial derivative:

  $$\frac{\partial f}{\partial x_i} = \lim_{\epsilon \to 0} \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}.$$

  The basic motivation of finite-difference approximations is to consider that:

  $$\frac{\partial f}{\partial x_i} \approx \frac{f(x + \epsilon e_i) - f(x)}{\epsilon},$$

  for a sufficiently small ϵ. The above is the so-called forward-difference approximation, an alternative formulation is, for instance, the central-difference approximation.

  Given a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ one can estimate its gradient ∇f and its Hessian $\nabla^2 f$ using finite-differences. A vector function $c : \mathbb{R}^n \to \mathbb{R}^m$ may also have its Jacobian J matrix estimated using finite differences. Some of the details of the implementation (e.g. the optimal choice of ϵ) are covered in [**?**, Sec. 8.1].

- The Hessian $\nabla^2 f$ and the Jacobian J may have a known sparse structure. For this situations, the estimation may be performed within fewer functions evaluations using graph coloring schemes [**?**, **?**]. I intend to implement and test these schemes.

  Some work to approximate sparse Jacobians have already been done (functions `approx_derivative` and `check_derivative`) and can be reused. Sparse Hessians approximation algorithms will have to be implemented from scratch, since they are different from sparse Jacobian algorithms because of the Hessian symmetry.

  Its worth notice that this graph coloring schemes works almost without modifications for other derivatives calculation approaches. That way, if more advanced methods for calculating the derivatives (e.g. automatic differentiation [**?**, Sec. 8.2]) are ever implemented in Scipy they could reuse this functions to deal with sparse structures.

Quasi-Newton Hessian Approximations[†]**— 1 or 2 weeks**

- It is useful to be able to use quasi-Newton Hessian approximation methods together with the interior point method. I intend to implement, integrate and test three quasi-Newton approximation methods for the use with the implemented interior point method: BFGS, SR1 and L-BFGS [**?**, Cap. 6 & 7]

  Scipy already have a `BFGS` implementation that could be reused, but SR1 and L-BFGS would have to be implemented from scratch (Scipy have a `L-BFGS-B` method available but it would not be of much use because it is only a wrapper for a FORTRAN function)

Tie Up the Loose Ends — Last week

- Finish to document, debug and optimize the code.

Other Commitments

I will attend to IFAC world congress from 9 to 14 of July.

[†]This is a low priority item. It implementation is dependent on the pace of the project and may be simplified or suppressed if needed.