Reference Implementation of Powell's Derivative-Free Optimization Solvers

What

This is the reference implementation of Powell's derivative-free optimization solvers, namely COBYLA, UOBYQA, NEWUOA, BOBYQA, and LINCOA.

This package is part of a research project funded by the Hong Kong Research Grants Council and the Hong Kong Polytechnic University (PolyU). It is still **under intensive development**, and there is no release yet. If you want to use the above-mentioned solvers, see the website and repository of PDFO instead.

Dedicated to late Professor M. J. D. Powell FRS (1936–2015).

Why

The goal is to implement these solvers in modern languages — first **modern** Fortran (F2003 or later), and then MATLAB, Python, and probably Julia and R. It will be a faithful implementation, in the sense that the new code will be mathematically equivalent to Powell's, except for the bug fixes and improvements that we make intentionally.

The focus is to implement Powell's solvers in a **structured** and **modularized** way so that they are **readable**, **maintainable**, and **extendable**. The new code will have no GOTO (of course) and will use matrix-vector procedures instead of loops whenever possible.

How

The mission is nontrivial due to the delicacy of Powell's algorithms and the unique style of his code. We started the Fortran code by refactoring Powell's code into the free form via a small MATLAB tool written by ourselves. However, such refactored code is far from what we want, because it inherits completely the structure and style of Powell's code except for the layout. Extensive modifications are needed to reorganize (indeed, to **rewrite**) the code. To maintain the faithfulness and quality of our implementation, intensive tests are conducted after each and every tiny modification, the test problems coming from the CUTEst set. The tests verify not only the faithfulness of our implementation, but also check that the solvers behave properly even if they are invoked with improper inputs or encounter failures of function evaluations.

All the tests are automated by GitHub Actions. As of July 2022, more than 20,000 "workflows" have been successfully run by GitHub Actions (see https://github.com/zequipe/gitpersonal/actions and https://github.com/zequipe/pdfo_ref/actions). Normally, each workflow consists of ~ 5 **randomized** tests that are conducted in parallel, each test taking from tens of minutes to several hours (the maximum is 6 hours, after which the workflow will be canceled automatically). In other words, our implementation has been tested by $\sim 10^5$ hours (or ~ 10 years) of randomized computation.

Bug fixes

The **old Fortran 77 implementation** has the following known issues, which **have been fixed in this modernized reference implementation**. Note that all the problems are bugs in the code rather than flaws in the algorithms. All the examples below are issues in Nlopt, a package providing interfaces to the **old Fortran 77 implementation**.

- The solvers may crash with segmentation faults
  - BOBYQA uninitialised variables in rare cases #133
  - Use of uninitialized variable in BOBYQA altmov #36
- The solvers may get stuck in infinite loops. For example, see
  - COBYLA freezes (though maxeval and maxtime are given) #370
  - COBYLA hangs #118
  - NEWUOA_BOUND stuck in infinite loop inside MMA #117
- The constrained solvers may not return the best point that is evaluated by the solver; sometimes, the returned point can have a large constraint violation even if the starting point is feasible. For example, see
  - COBYLA optimizer gives unexpected output #182
  - Last Result Returned Not Optimized Result #110
  - COBYLA returns last evaluated function which might not be minimum #57