Stanford University, ICME

CME 338 Large-Scale Numerical Optimization

Instructor: Michael Saunders Spring 2019

Notes 3: **Suggested projects**

The class project can be one of your own choice, or perhaps one of the suggestions below. It's always good to write code that can be used by others (perhaps forever!). Ideally we will be able to add your work to the list of freely available software here: `http://stanford.edu/group/SOL/software.html/`.

At some point the projects may seem open-ended. Don't worry excessively. Grades will be awarded based on best effort within a reasonable time. Perhaps you will be motivated to continue adding features during the summer (except ICME's Indira urges you to finish by the end of spring!).

1 MINRES with local reorthogonalization

This is intended for users of GMRES on symmetric systems for which the preconditioner is so good that not many iterations are expected. Like LSMR, it will still be efficient on problems of arbitrary size if the input parameter `localSize` is no bigger than 10 or 20 (say).

GMRES is intended for square unsymmetric systems, but it is often used on symmetric saddle-point systems because it stores all Arnoldi vectors and will generally take fewer iterations than MINRES. With local reorthogonalization we can make MINRES as effective as GMRES while retaining symmetry. It feels "better".

A preliminary version of MINRES with local reorthogonalization has been implemented in `http://stanford.edu/group/SOL/software/minres/minresReorthog.zip` by ICME student Danielle Maddix. This could be used as a starting point. The project is to make sure that the code works with a positive-definite preconditioner `M`.

2 USYMQR

Saunders, Simon, and Yip [17] proposed two methods for solving square systems $Ax = b$ based on orthogonal tridiagonalization of A. Twenty years later, Reichel and Ye [15] discussed the same tridiagonalization for rectangular systems. We have a Fortran 90 implementation of USYMLQ for compatible systems $Ax = b$. It is of interest to implement USYMQR for both $Ax = b$ and $\min \|Ax - b\|_2$, and to test it on realistic examples.

3 PDCO search directions

PDCO has many methods for computing search directions $(\Delta x, \Delta y, \Delta z)$. See `pdcoSet.m` in `pdco5.zip` (`http://stanford.edu/group/SOL/software/pdco/`). There is room for additional methods. For example, `Method = 23` (SYMMLQ on Ch 9 system (20)) has not been implemented. If an iterative method for symmetric indefinite systems is to be used, it may be preferable to work with a system analogous to $K_{2.5r}$ on p67.

4 Implementing LUSOL in Julia

As mentioned in class and notes07, LUSOL is the "engine" for several established solvers: MINOS, SQOPT, SNOPT, lp_solve, MILES, PATH, PATHNLP [11, 12, 13, 4, 5, 6, 9, 16, 2, 3, 14]. We have f77 and f90 versions of LUSOL (both about 6500 lines of code). A possible project is to translate one of these into Julia.

5 A stable primal-dual method for linear optimization

Reference [8] describes an intriguing approach to solving vanilla linear optimization problems

$$\min\ c^T x \ \ \text{st} \ \ Ax = b,\ x \geq 0, \tag{1}$$

based on finding a well-conditioned basis such that $A = \begin{pmatrix} B & E \end{pmatrix}$, where B is block-diagonal or triangular, so that linear systems involving B or B^T can be solved efficiently. (Evidently most of the LPnetlib problems contain such a B.) The associated reduced-gradient (null-space) operator

$$N = \begin{pmatrix} -B^{-1}E \\ I \end{pmatrix}$$

allows the primal solution to be treated as $x = \widehat{x} + Nv \geq 0$, where $A\widehat{x} = b$. Assuming we have such an $\widehat{x}$, the optimal primal-dual variables (x, y, z), which satisfy $x \geq 0$ and $z = c - A^T y \geq 0$, solve problem (1) iff they satisfy the bilinear optimality equation

$$F_\mu(v, y) \equiv ZXe - \mu e = 0 \tag{2}$$

with $\mu = 0$, where $X \equiv \text{diag}(\widehat{x} + Nv)$ and $Z = \text{diag}(z)$. The strategy is to apply Newton's method for nonlinear equations to (2) with μ decreasing toward 0. The Jacobian of F_μ is

$$J \equiv \nabla F_\mu(v, y) = \begin{pmatrix} ZN & -XA^T \end{pmatrix}, \tag{3}$$

and preconditioned LSQR is applied to the Newton equation

$$J \begin{pmatrix} \Delta v \\ \Delta y \end{pmatrix} = -F_\mu(v, y). \tag{4}$$

A key feature is that J does not necessarily become ill-conditioned as μ approaches 0. Numerical results in [8] show that Newton's method can compute (x, y, z) to full precision.

Given such excellent performance, a worthwhile project will be to derive and implement an analogous method for LO problems with general bounds $\ell \leq x \leq u$.

6 BPprimal with general bounds

`BPprimal.m` is a MATLAB code that solves the regularized BPDN problem

$$\begin{array}{ll} \underset{x,\,y}{\text{minimize}} & \|x\|_1 + \frac{1}{2}\delta\|x\|_2^2 + \frac{1}{2}\lambda\|y\|_2^2 \\ \text{subject to} & Ax + \lambda y = b. \end{array}$$

The active-set method for `BPprimal.m` is the reduced-gradient method with basic variables y throughout ($B = \lambda I$). The superbasic variables are nonzero elements of x associated with their support S from A. The project task is to include general bounds:

$$\begin{array}{ll} \underset{x,\,y}{\text{minimize}} & \|x\|_1 + \frac{1}{2}\delta\|x\|_2^2 + \frac{1}{2}\lambda\|y\|_2^2 \\ \text{subject to} & Ax + \lambda y = b, \quad \ell \leq x \leq u. \end{array}$$

Download `http://stanford.edu/group/SOL/software/asp/matlab/asp-v1.0.zip` to obtain the MATLAB package ASP. `BPprimal.m` uses several auxiliary functions from another solver `BPdual.m` within ASP. Some initial documentation is in `asp/doc/bpprimal.pdf`.

7 Complex-step estimates of gradients

Let $F(x)$ be a smooth real scalar function of the real scalar x with gradient $g(x)$ and Hessian $H(x)$, and let h be a small real step. Taylor series expansions give

$$F(x+h) = F(x) + hg(x) + \frac{h^2}{2!}H(x) + O(h^3),$$
$$F(x-h) = F(x) - hg(x) + \frac{h^2}{2!}H(x) + O(h^3).$$

Thus, $g(x)$ can be estimated by forward differences or central differences as

$$g_f(x) = (F(x+h) - F(x))/h + O(h),$$
$$g_c(x) = (F(x+h) - F(x-h))/2h + O(h^2),$$

where $h \approx \epsilon^{1/2}$ and $\epsilon^{1/3}$ respectively give best accuracy on a machine with floating-point precision ϵ (according to [7], under certain assumptions). In double-precision arithmetic, this means $h \approx 10^{-8}$ and 10^{-5}. The error in the gradient approximations is then $O(10^{-8})$ and $O(10^{-10})$.

If $F(z)$ is an analytic function of complex z and h is a tiny step off the real axis, the Taylor series is

$$F(x+ih) = F(x) + ihg(x) - \frac{h^2}{2!}H(x) - \frac{ih^3}{3!}F'''(x) + O(h^4),$$

and the complex-step estimate of the gradient at real x is

$$g_{cs}(x) = \text{imag}(F(x+ih))/h + O(h^2),$$

where $\epsilon \le h \le \epsilon^{1/2}$ gives best accuracy, as described by Moler [10]. Thus in double precision with $h = \epsilon^{1/2} \approx 10^{-8}$ or smaller down to ϵ, the error in the gradient approximation should be $O(\epsilon) \approx 10^{-16}$ (full precision).

The project can involve a particular function $F(x)$[1] that arises in optimal design of computer experiments [1]. We have a Fortran 90 implementation of the function and have been optimizing it using MINOS in quadruple precision, with gradients estimated by forward and central differencing. For difficult cases, $O(\epsilon^{1/2})$ accuracy in the gradients is not enough (where quad $\epsilon \approx 10^{-34}$!). The function involves the error function `erf(x)`. The complex-step method therefore needs the complex error function `erf(x+ih)`, which Fortran doesn't have! Luckily h is very small. We can construct a series that converges rapidly.

Note

Projects 6 and 7 have been implemented by Varun Vasudevan (2019) and by Bazyli Klockiewicz and Nimit Sohoni (2018).

[1] The IMSPE function (Integrated Mean-Square Prediction Error).

References

[1] S. B. Crary. New research directions in computer experiments: ϵ-clustered designs. In *JSM Proceedings, Statistical Computing Section*, pages 5692–5706, Alexandria, VA, 2013. `https://drcrary.files.wordpress.com/2014/06/crary-src2012proceedingsrev20131010.pdf`.

[2] S. Dirkse, M. C. Ferris, and T. S. Munson. PATH nonlinear complementarity solver. `http://pages.cs.wisc.edu/~ferris/path.html`, accessed May 2017.

[3] M. C. Ferris and T. S. Munson. PATH nonlinear complementarity solver. `https://www.gams.com/latest/docs/solvers/path/`, accessed May 2017.

[4] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. SIGEST article.

[5] P. E. Gill, W. Murray, and M. A. Saunders. User's guide for SQOPT 7: Software for large-scale linear and quadratic programming. `http://www.scicomp.ucsd.edu/~peg` (see Software), 2006.

[6] P. E. Gill, W. Murray, M. A. Saunders, and E. Wong. User's Guide for SNOPT 7.7: Software for Large-Scale Nonlinear Programming. Center for Computational Mathematics Report CCoM 18-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2018.

[7] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1981.

[8] M. Gonzalez-Lima, H. Wei, and H. Wolkowicz. A stable primal-dual approach for linear programming under nondegeneracy assumptions. *Comput. Optim. Appl.*, 44:213–247, 2009. `http://stanford.edu/class/cme338/refs/GonWW2007.pdf`.

[9] lp_solve open source LP and MILP solver. `http://groups.yahoo.com/group/lp_solve/`.

[10] C. B. Moler. Complex Step Differentiation. Cleve's Corner: Cleve Moler on Mathematics and Computing. `https://blogs.mathworks.com/cleve/2013/10/14/complex-step-differentiation/`.

[11] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Math. Program.*, 14:41–72, 1978.

[12] B. A. Murtagh and M. A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. *Math. Program. Study*, 16:84–117, 1982.

[13] B. A. Murtagh and M. A. Saunders. MINOS nonlinear optimization solver. `https://www.gams.com/latest/docs/solvers/minos/`, accessed May 2017.

[14] PATH nonlinear complementarity solver. `https://www.gams.com/latest/docs/solvers/path/`, accessed May 2017.

[15] L. Reichel and Q. Ye. A generalized LSQR algorithm. *Numer. Linear Algebra Appl.*, 15:643–660, 2008.

[16] T. F. Rutherford. MILES nonlinear complementarity solver. `https://gams.com/latest/docs/solvers/miles/`, accessed May 2017.

[17] M. A. Saunders, H. D. Simon, and E. L. Yip. Two conjugate-gradient-type methods for unsymmetric linear equations. *SIAM J. Numer. Anal.*, 25(4):927–940, 1988.