blockSQP user's manual

Dennis Janka

September 23, 2015

Contents

1 Introduction

`blockSQP` is a sequential quadratic programming method for finding local solutions of nonlinear, nonconvex optimization problems. It is particularly suited for —but not limited to—problems whose Hessian matrix has block-diagonal structure such as problems arising from direct multiple shooting parameterizations of optimal control or optimum experimental design problems.

`blockSQP` has been developed around the quadratic programming solver `qpOASES` [1] to solve the quadratic subproblems. Gradients of the objective and the constraint functions must be supplied by the user in sparse or dense format. Second derivatives are approximated by a combination of SR1 and BFGS updates. Global convergence is promoted by the filter line search of Waechter and Biegler [4, 5] that can also handle indefinite Hessian approximations.

The method is described in detail in [2, Chapters 6–8]. These chapters are largely self-contained. The notation used throughout this manual is the same as in [2]. A publication [3] is currently under review.

2 Installation

The following steps

1. Download and install qpOASES from `https://projects.coin-or.org/qpOASES`.

   It is recommended to use at least release 3.2.0. Alternatively, check out revision 155 from the `qpOASES` subversion repository that is located at `https://projects.coin-or.org/svn/qpOASES/trunk/`. For best performance it is strongly recommended to install the sparse solver `MA57` from HSL as described in the `qpOASES` manual, Sec. 2.2.

2. In the `blockSQP` main directory, open `makefile` and set `QPOASESDIR` to the correct location of the `qpOASES` installation.

3. Compile `blockSQP` by calling `make`. This should produce a shared library `libblockSQP.so` in `lib/`, as well as executable example problems in the `examples/` folder.

3 Setting up a problem

A nonlinear programming problem (NLP) of the form

$$\min_{x \in \mathbb{R}^n} \varphi(x) \tag{1a}$$

$$\text{s.t. } b_\ell \leq \begin{bmatrix} x \\ c(x) \end{bmatrix} \leq b_u \tag{1b}$$

is characterized by the following information that must be provided by the user:

- The number of variables, n,

- the number of constraints, m,
- the objective function, $\varphi : \mathbb{R}^n \longrightarrow \mathbb{R}$,
- the constraint function, $c : \mathbb{R}^n \longrightarrow \mathbb{R}^m$,
- and lower and upper bounds for the variables and constraints, b_ℓ and b_u.

In addition, `blockSQP` requires the evaluation of the

- objective gradient, $\nabla\varphi(x) \in \mathbb{R}^n$, and the
- constraint Jacobian, $\nabla c(x) \in \mathbb{R}^{m\times n}$.

Optionally, the following can be provided for optimal performance of `blockSQP`:

- In the case of a block-diagonal Hessian, a partition of the variable vector x corresponding to the diagonal blocks,
- a heuristic function r to compute a point x where a reduced infeasibility can be expected, $r : \mathbb{R}^n \longrightarrow \mathbb{R}^n$.

`blockSQP` is written in C++ and uses an object-oriented programming paradigm. The method itself is implemented in a class `SQPmethod`. Furthermore, `blockSQP` provides a basic class `ProblemSpec` that is used to specify an NLP of the form (1). To solve an NLP, first an instance of `ProblemSpec` must be passed to an instance of `SQPmethod`. Then, `SQPmethod`'s appropriate methods must be called to start the computation.

In the following, we first describe the `ProblemSpec` class and how to implement the mathematical entities mentioned above. Afterwards we describe the necessary methods of the `SQPmethod` class that must be called from an appropriate driver routine. Some examples where NLPs are specified using the `ProblemSpec` class and then passed to `blockSQP` via a simple C++ driver routine can be found in the `examples/` subdirectory.

3.1 Class `ProblemSpec`

Dense and sparse problems

Implement constructor

Variables must be set

3.1.1 Function `init`

3.1.2 Function `evaluate`

3.1.3 Function `reduceConstrVio`

3.2 Class `SQPmethod`

4 Options and parameters

In this section we describe all options that are passed to `blockSQP` through the `SQPoptions` class. We distinguish between algorithmic options and algorithmic parameters. The former are used to choose between different algorithmic alternatives, e.g., different Hessian approximations, while the latter define internal algorithmic constants. As a rule of thumb, whenever you are experiencing convergence problems with `blockSQP`, you should try different algorithmic options first before changing algorithmic parameters.

Additionally, the output can be controlled with the following options:

Name	Description/possible values	Default
`printLevel`	Amount of onscreen output per iteration 0: no output 1: normal output 2: verbose output	1
`printColor`	Enable/disable colored terminal output 0: no color 1: colored output in terminal	1
`debugLevel`	Amount of file output per iteration 0: no debug output 1: print one line per iteration to file 2: extensive debug output to files (impairs performance)	0

4.1 List of algorithmic options

Name	Description/possible values	Default
`sparseQP`	`qpOASES` flavor 0: dense matrices, dense factorization of red. Hessian 1: sparse matrices, dense factorization of red. Hessian 2: sparse matrices, Schur complement approach	2
`globalization`	Globalization strategy 0: full step 1: filter line search globalization	1
`skipFirstGlobalization`	0: deactivate globalization for the first iteration 1: normal globalization strategy in the first iteration	1

`restoreFeas`	Feasibility restoration phase 0: no feasibility restoration phase 1: minimum norm feasibility restoration phase	1
`hessUpdate`	Choice of first Hessian approximation 0: constant, scaled diagonal matrix 1: SR1 2: BFGS 3: [not used] 4: finite difference approximation	1
`hessScaling`	Choice of scaling/sizing strategy for first Hessian 0: no scaling 1: scale initial diagonal Hessian with σ_{SP} 2: scale initial diagonal Hessian with σ_{OL} 3: scale initial diagonal Hessian with σ_{Mean} 4: scale Hessian in every iteration with σ_{COL}	2
`fallbackUpdate`	Choice of fallback Hessian approximation (see `hessUpdate`)	2
`fallbackScaling`	Choice of scaling/sizing strategy for fallback Hessian (see `hessScaling`)	4
`hessLimMem`	0: full-memory approximation 1: limited-memory approximation	1
`blockHess`	Enable/disable blockwise Hessian approximation 0: full Hessian approximation 1: blockwise Hessian approximation	1
`hessDamp`	0: enable BFGS damping 1: disable BFGS damping	1
`whichSecondDerv`	User-provided second derivatives 0: none 1: for the last block 2: for all blocks (same as `hessUpdate=4`)	0
`maxConvQP`	Maximum number of convexified QPs (`int`>0)	1
`convStrategy`	Choice of convexification strategy 0: Convex combination between `hessUpdate` and `fallbackUpdate` 1: Add multiples of identity to first Hessian [not implemented yet]	0

4.2 List of algorithmic parameters

Name	Symbol/Meaning	Default
`opttol`	ε_{opt}	1.0e-5

`nlinfeastol`	$\varepsilon_{\text{feas}}$	1.0e-5
`eps`	machine precision	1.0e-16
`inf`	∞	1.0e20
`maxItQP`	Maximum number of QP iterations per SQP iteration (`int`>0)	5000
`maxTimeQP`	Maximum time in second for `qpOASES` per SQP iteration (`double`>0)	10000.0
`maxConsecSkippedUpdates`	Maximum number of skipped updates before Hessian is reset (`int`>0)	100
`maxLineSearch`	Maximum number of line search iterations (int>0)	20
`maxConsecReducedSteps`	Maximum number of reduced steps before restoration phase is invoked (int>0)	100
`hessMemsize`	Size of Hessian memory (`int`>0)	20
`maxSOCiter`	Maximum number of second-order correction steps	3

5 Output

6 Notes for developers

References

[1] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, pages 1–37, 2014.

[2] Dennis Janka. *Sequential quadratic programming with indefinite Hessian approximations for nonlinear optimum experimental design for parameter estimation in differential–algebraic equations*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2015. Available at `http://archiv.ub.uni-heidelberg.de/volltextserver/19170/`.

[3] Dennis Janka, Christian Kirches, Sebastian Sager, and Andreas Wächter. An SR1/BFGS SQP algorithm for nonconvex nonlinear programs with block-diagonal Hessian matrix. *submitted to Mathematical Programming Computation*, 2015.

[4] Andreas Wächter and Lorenz T Biegler. Line search filter methods for nonlinear programming: Local convergence. *SIAM Journal on Optimization*, 16(1):32–48, 2005.

[5] Andreas Wächter and Lorenz T Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005.