Convexity Determination in the Microsoft Excel Solver Using Automatic Differentiation Techniques

Ivo P. Nenov, Daniel H. Fylstra, Lubomir V. Kolev
Frontline Systems Inc., USA

Abstract: We present algorithmic tests to determine whether smooth functions are convex over certain regions, in an algebraic modeling system implemented in the Premium Solver Platform for Microsoft Excel. We apply convexity determination to the objective, constraints and bounds of an optimization problem, to determine whether a locally optimal solution will be globally optimal, and whether an interior point algorithm for convex problems can be used as the solution method. The algorithms generally rely on fast and accurate computation of the 'Hessian sign' or the full interval Hessian of each problem function, making automatic differentiation techniques a practical requirement for their use; their effectiveness is improved by using constraint propagation techniques to narrow the intervals over which the functions are evaluated. We compare different convexity determination algorithms on a test set of models and functions defined by expressions in the Microsoft Excel formula language.

Introduction: Convexity is clearly a desirable property for an optimization model. If a model is known or can be shown to be convex, we can be confident of finding a globally optimal solution, and we can apply (among others) high performance interior point methods to find that solution. Linear programming (LP) problems are of course convex, and quadratic programming (QP) problems are convex if the Hessian of the objective is positive semidefinite. But for general smooth nonlinear problems, it is nontrivial to determine whether each problem function is convex *over the feasible region* determined by bounds on the variables and other constraint functions. Yet many smooth nonlinear problems of interest may be convex, more than is commonly realized; for example, Stephen Boyd [1], who has studied many such problems arising in electrical engineering, has described a wide range of practical problems shown to be convex.

In view of the importance of convexity, it is somewhat surprising that existing algebraic modeling systems do not provide much support for testing convexity of optimization problems written in the modeling language. This paper describes an effort to implement convexity determination in the framework of the Microsoft Excel Solver, specifically in the Premium Solver Platform, an upward compatible extension of the Excel Solver aimed at challenging industrial optimization problems of all types. The modeling language in this case is the Excel formula language, which is richly expressive

with over 300 built-in functions. Our convexity tests utilize support in the Premium Solver Platform for model diagnosis, interval evaluation and automatic differentiation of Excel spreadsheet formulas, as previously described [2, 3]. They may prove or disprove convexity or yield inconclusive results, but in many practical cases they yield useful information for the modeler and the solver.

In this paper, functions are assumed to be smooth, though we note in the summary that a variety of techniques exist to convert certain commonly used non-smooth constraint functions (e.g. IF, MIN, MAX) into linear constraints with additional binary integer variables – potentially yielding a mixed-integer convex problem. The *original problem* is: given a function $f : \boldsymbol{x} \in R^n \rightarrow R$, $f \in C^2(\boldsymbol{x})$ (twice continuously differentiable in the box $\boldsymbol{x}$), prove computationally that the function is convex, concave, or neither convex nor concave in $\boldsymbol{x}$. In our implementation four general steps are used, in order of increasing computational cost:

1. Perform model diagnosis as described in [2, 3] to identify linear functions in the model, which are convex without further analysis.
2. Find as narrow as possible bounds on variables (the box $\boldsymbol{x}$) considering the function types in the model (objective, equality, inequality) and applying techniques of *constraint propagation*.

Steps 3 and 4 are then performed for each nonlinear function in the problem:

3. Check the positive definiteness of the interval Hessian $\boldsymbol{H}(\boldsymbol{x})$ (the extension of the Hessian $H(x)$ of $f(x)$, $x \in \boldsymbol{x}$) by computing a value called the *Hessian sign* without computing $\boldsymbol{H}(\boldsymbol{x})$ itself, using methods akin to automatic differentiation.
4. Check the positive definiteness of the interval Hessian $\boldsymbol{H}(\boldsymbol{x})$ by computing $\boldsymbol{H}(\boldsymbol{x})$ itself, using automatic differentiation methods, then analyzing it numerically. In practice, steps 3 and 4 are combined, stopping after step 3 if its test is conclusive.

In all of the above steps the function being tested is evaluated in forward and backward passes on its Directed Acyclic Graph (DAG) through a technique we call

generalized number evaluation, as described in [2, 3]. The process yields for each function (i) a proof of convexity or concavity over the feasible region, (ii) a proof of non-convexity, or (iii) an indeterminate result – the tests neither prove nor disprove convexity.

The goal of finding an outer approximation for the *feasible region* of the optimization model is achieved by forward and backward evaluation with a generalized number we call *reverse interval*. All arithmetic operators and primitive functions are overloaded for such numbers. In forward mode the natural interval extensions are computed and saved in the nodes of the DAG. In reverse mode the inverse operations are executed propagating the constraint on the function value toward its arguments. The intermediate interval values at DAG nodes are intersected with the saved intervals during the forward step, often yielding a narrower box for the arguments. Usually the evaluation starts with a preset *initial box* for arguments $\boldsymbol{x}_o$, and is repeated until no significant reduction of the argument box is achieved. However, in some situations it's possible to start with $\boldsymbol{x}_o = (-\infty,\infty)$ and still determine bounds on the variables.

With (tightened) bounds on the variables, we attempt to determine convexity of a function by evaluating with another generalized number we call an *extended interval gradient* number. The traditional *interval gradient* number used to evaluate in forward mode is extended with an extra data member to accumulate the Hessian sign. Although computationally inexpensive, this approach may fail to determine convexity in case of a vector function.

For this reason, we also accumulate information during the forward evaluation, which allows us to execute a (computationally more expensive) backward sweep, thereby obtaining the full interval Hessian. The latter is used in a final attempt to determine convexity through a numerical analysis, using methods previously proposed by Kolev [4] and Rohn [5] in other contexts.

We present preliminary results from tests of the different methods on a variety of problems expressed as Microsoft Excel models, concluding that such convexity testing methods are an effective and useful tool for the user seeking to solve challenging optimization models.

References for the Extended Abstract

[1] Boyd S., L. Vandenberghe: *Convex Optimization*, London: Cambridge University Press, 2004.

[2] Fylstra D., I. Nenov: Automatic Differentiation in Microsoft Excel, *3rd International Conference/Workshop on Automatic Differentiation*, Nice, France, June 19-23, 2000.

[3] Nenov I., D. Fylstra: Interval methods for accelerated global search in the Microsoft Excel Solver, *Reliable Computing*, vol.9 (2), pp.143-159, April 2003.

[4] Kolev, L.: On the stability of dynamic interval systems, *SCAN-90*, Albena, Bulgaria, September 24-28, 1990.

[5] Rohn J.: Positive Definiteness and Stability of Interval Matrices, *SIAM J. Matrix Anal. App.*, vol.15 (1), pp.175-184, January 1994.