EXAMINER

Contents

1 Introduction

This document describes GAMS/Examiner, a tool for examining points and making an unbiased, independent assessment of their merit. In short, it checks if solutions are *really* solutions. As an example, it can take a solution point reported as optimal by a solver and examine it for primal feasibility, dual feasibility, and optimality. It has a number of different modes, allowing it to check the input point from GAMS/Base as well as the solution passed by a solver back to GAMS.

Many of the tests done by Examiner (indeed, perhaps all of them) are already being done by the GAMS solvers, so Examiner is in a sense redundant. However, a facility to make an independent, transparent check of a solver's solution is very useful in solver development, testing, and debugging. It is also useful when comparing the solutions returned by two different solvers. Finally, a tool like the Examiner allows one to examine solutions using different optimality tolerances and optimality criteria in a way that is not possible when working with the solvers directly.

GAMS/Examiner is installed automatically with your GAMS system. Without a GAMS/Base license, examiner will run in student or demonstration mode (i.e. it will examine small models only).

2 Usage

Examiner can be used with all supported model types. Since Examiner doesn't really solve any problems, it is not a good choice for a default solver, and it does not appear as an option in the list of possible solver defaults when installing GAMS. However, you can choose Examiner via the command line:

```
gams trnsport LP=examiner;
```

or via a GAMS option statement

```
option LP=examiner;
```

somewhere before the `solve` statement.

Since Examiner is not really a solver, many of the usual GAMS options controlling solvers have no impact on it. However, the `sysout` option is interpreted in the usual way.

The optimality checks done in Examiner are first-order optimality checks done at a given point. A discussion here of these conditions and all they imply would be redundant: any good intro text in optimization will cover them. For linear programming, first-order optimality is all one needs to prove global optimality. For nonlinear programming, these conditions may or may not be necessary or sufficient for optimality; this depends on the convexity of the feasible set and objective and the form of the constraints. For integer programming models, these checks only make sense if we turn the global problem into a local one by adding bounds to the model, essentially fixing each discrete variable to its current value: these bounds are added automatically by Examiner.

Examiner runs in two basic modes of operation: it can examine the input point passed from GAMS/Base to the solver, and it can examine the point passed from the solver back to GAMS. Each mode can be used independent of the other. By default, it will operate in the first mode, examining the initial "solution" passed to it by GAMS, and this only if GAMS indicates it is passing an advanced basis to the solver (cf. the GAMS User Guide and the `bratio` option). If you wish to use the second "solver-check" mode, you may specify an appropriate subsolver using the `subsolver` option (see Section 3). If no `subsolver` is selected, the default solver for the model type being solved is used. In most cases you will want to use an option file to specify exactly what type of examination you wish to perform. The rules for using an option file are described in Chapter 1, "Basic Solver Usage".

2.1 Solution Points: Definition

There are a number of different ways a solution point can be defined. Of course the different definitions will typically result in the same points being produced, but there are cases where this will not be precisely so. Since Examiner is intended to explore and analyze these cases, we must make these definitions precise. The following four points are defined and used in Examiner:

1. The `gamspoint` is the input point provided by GAMS to Examiner. The GAMS input point includes level & marginal values for the rows and columns: Examiner uses these exactly as given.
2. The `initpoint` is determined by the variable levels (primal vars) and equation marginals (dual vars) provided by GAMS to Examiner. These values are used to *compute* the equation levels and variable marginals / reduced costs using the function evaluator in Examiner, instead of using the values passed in by GAMS for these.
3. The `solupoint` is similar to the `initpoint`: it uses the variable levels (primal vars) and equation marginals (dual vars) to *compute* the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.
4. The `solvpoint` is the point returned by the subsolver. The subsolver returns both level & marginal values for the rows and columns: Examiner uses these exactly as given.

2.2 Checks Performed

There are a number of checks that can be performed on any of the solution points defined in Section 2.1. By default, Examiner tries to choose the appropriate checks. For example, if a primal simplex solver returns a models status of nonoptimal, the only checks that makes sense are feasibility in the primal variables and constraints. However, this automatic choice of appropriate checks is not possible when checking points passed in from GAMS/Base.

1. **Primal variable feasibility**: check that all primal variables are within bounds.
2. **Primal constraint feasibility**: check that all primal constraints are satisfied.
3. **Dual variable feasibility**: check that all dual variables are within bounds.
4. **Dual constraint feasibility**: check that all dual constraints are satisfied.
5. **Primal complementary slackness**: check complementarity between the primal variables and the dual constraints / reduced costs.

6. **Dual complementary slackness**: check complementarity between the dual variables / equation marginals and the equation slacks.
7. **Equilibrium condition complementarity**: check complementarity of the equation/variable pairs in complementarity models (MCP, MPEC).

The checks above are implemented with default tolerances. These tolerances can be changed via an option file (see Section 3.2).

There exist different ways to check the items mentioned above. For example, when checking for primal feasibility, different norms can be used to measure the error of the residual. Currently, we have only implemented one way to make these checks. TODO: document this implementation, perhaps add others.

2.3 Scaling

By default, Examiner makes its checks on the original, unscaled model. In many cases, though, it is important to take scaling into account. Consider the effect of row scaling on the simple constraint $x^2 \leq 9$ where $x = 3.5$. Multiplying this constraint through by large or small constants changes the amount of the constraint violation proportionately, but the distance to feasibility is not changed. Applying row scaling to the original model eliminates this problem.

Most solvers scale a model before solving it, so any feasibility or optimality checks and tolerances are applied to the scaled model. The process of unscaling the model can result in a loss of feasibility or optimality. Even though we do not have access to the scales applied by the solver and cannot construct precisely the same scaled model, we can expect to get a better idea of how the solver did by looking at a model scaled by Examiner than by looking at the original.

It is also interesting to see what the model scaling looks like even if we do not apply the scales to do the Examiner checks. If the row scales are in a nice range, say [.1,100], we can have some confidence that the model is well-scaled. In contrast, if the row scales are in the range [1,1e8] we may question the precision of the solution provided.

For each row, Examiner computes the true row scale as

$$\max(\|RHS_i\|, \max_j(\|A_{ij}\| \cdot \max(1, \|x_j\|)))$$

In this way variables with a large level value lead to large scale factors. To make the scale factor independent of the variable values, use an option file line of "AbsXScale 0". This replaces the term $\max(1, \|x_j\|)$ above with 1.

Since the user may wish to limit the size of the scale factors applied, the true row scales are projected onto the scale factor bounds to get the applied scale factors. The scale factors are applied when making a scaled check by dividing the rows by the scale factors and multiplying the corresponding Lagrange multipliers by these same factors. When making unscaled checks information about the true scales is still included in the output to give the user a hint about potential scaling issues.

Note that the scaled and unscaled checks are made independently. By default only the unscaled checks are performed. If you turn the scaled checks on via an option file line "scaled 1", this will not turn off the unscaled checks. You will need an option file line of "unscaled 0" to turn off unscaled checks.

3 Options

For details on how to create and use an option file, see the introductory chapter on solver usage.

3.1 General Options

The following general options control the behavior of GAMS/Examiner. Many of these are boolean (i.e. on/off) options; in this case, zero indicates off, nonzero on.

Option	Description	Default
`absxscale`	If on, the matrix coefficients are multiplied by max(1,abs(x)) when computing the scale factors. If off, the matrix coefficients are taken as is. See Section 2.3	on
`dumpgamspoint`	If on, dump the `gamspoint` to a basis file in GAMS source format.	off
`dumpinitpoint`	If on, dump the `initpoint` to a basis file in GAMS source format.	off
`dumpsolupoint`	If on, dump the `solupoint` to a basis file in GAMS source format.	off
`dumpsolvpoint`	If on, dump the `solvpoint` to a basis file in GAMS source format.	off
`examinegamspoint`	If on, examine the `gamspoint`.	off
`examineinitpoint`	If on, examine the `initpoint`. By default, this option is on if GAMS/Base passes an advanced basis, and off otherwise.	auto
`examinesolupoint`	If on, examine the `solupoint`. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
`examinesolvpoint`	If on, examine the `solvpoint`. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
`fcheckall`	If set, this option forces all the checks from Section 2.2 on or off.	auto
`fcheckdcon`	If set, this option forces the dual contraint feasibility check from Section 2.2 on or off.	auto
`fcheckdcmp`	If set, this option forces the dual complementary slackness check from Section 2.2 on or off.	auto
`fcheckdvar`	If set, this option forces the dual variable feasibility check from Section 2.2 on or off.	auto
`fcheckpcon`	If set, this option forces the primal constraint feasibility check from Section 2.2 on or off.	auto
`fcheckpcmp`	If set, this option forces the primal complementary slackness check from Section 2.2 on or off.	auto
`fcheckpvar`	If set, this option forces the primal variable feasibility check from Section 2.2 on or off.	auto
`perpsys`	Controls output during examination of solution points. If on, print out the point in a way that allows for easy visual inspection and verification of the KKT or first order optimality conditions. First, the primal level values and bounds are printed next to the reduced costs. Next, the duals levels and bounds are printed, next to the row slacks.	off
`returngamspoint`	If on, return the `gamspoint` as a solution to GAMS/Base.	off
`returninitpoint`	If on, return the `initpoint` as a solution to GAMS/Base.	auto
`returnsolupoint`	If on, return the `solupoint` as a solution to GAMS/Base.	auto
`returnsolvpoint`	If on, return the `solvpoint` as a solution to GAMS/Base.	auto
`scaled`	If set, examiner checks will be made on the scaled model.	no
`scaleLB`	Lower bound for applied row scales.	1.0
`scaleUB`	Upper bound for applied row scales.	1.0e300
`subsolver`	Indicates what subsolver to run. By default, the subsolver used is the default subsolver for the model type in question.	auto
`subsolveropt`	If set, indicates what optfile value to pass to the subsolver used. Can also be set via the `subsolver` option by appending a `.n` to the subsolver name, e.g. `subsolver bdmlp.3`	auto
`trace`	If set, trace information will be computed and appended to this file. By	none

3.2 Tolerance Options

The following options can be used to set the various tolerances to non-default values.

Option	Description	Default
`dualcstol`	Dual complementary slackness tolerance. By dual CS we refer to complementary slackness between the dual variables and the primal constraints.	1e-7
`dualfeastol`	Dual feasibility tolerance. This tolerance is used for the checks on the dual variables and the dual constraints.	1e-6
`ectol`	Equilibrium condition complementarity tolerance. Applicable to MCP and MPEC models, where the equilibrium conditions are given by the equation-variable pairs in the model statement.	1e-6
`primalcstol`	Primal complementary slackness tolerance. By primal CS we refer to complementary slackness between the primal variables and the dual constraints.	1e-7
`primalfeastol`	Primal feasibility tolerance. This tolerance is used for the checks on the primal variables and the primal constraints.	1e-6