NLPEC

Contents

1 Introduction

The GAMS/NLPEC solver, developed jointly by Michael Ferris of UW-Madison and GAMS Development, solves MPEC and MCP models via reformulation of the complementarity constraints. The resulting sequence of NLP models are parameterized by a scalar μ and solved by existing NLP solvers. The resulting solutions used to recover an MPEC or MCP solution.

GAMS/NLPEC serves a number of purposes. In many cases, it is an effective tool for solving MPEC models, the only such tool available within GAMS. It also serves as a way to experiment with the many reformulation strategies proposed for solving MPEC and MCP models. Without something like NLPEC (and a library of models to test with) a comprehensive and thorough test and comparison of the various reformulation strategies would not be possible. To better serve these purposes, NLPEC has an open architecture. The model reformulations are written out as GAMS source for solution via an NLP solver, so it is possible to view this source and modify it if desired.

A brief note about notation is in order. The GAMS keyword `positive` is used to indicate nonnegative variables. The same holds for nonpositive variables and the GAMS keyword `negative`.

2 Usage

GAMS/NLPEC can solve models of two types: MPEC and MCP. If you did not specify NLPEC as the default MPEC or MCP solver, use the following statement in your GAMS model before the solve statement:

```
option MPEC=nlpec; { or MCP }
```

You can also make NLPEC the default solver via the command line:

```
gams nash MPEC=nlpec MCP=nlpec
```

You can use NLPEC with its default strategy and formulation, but most users will want to use an options file (Section 4) after reading about the different types of reformulations possible (Section 3). In addition, an understanding of the architecture of NLPEC (Section 5) will be helpful in understanding how GAMS options are treated. Although NLPEC doesn't use the GAMS options `workspace`, `workfactor`, `optcr`, `optca`, `reslim`, `iterlim`, and `domlim` directly, it passes these options on in the reformulated model so they are available to the NLP subsolver.

3 Reformulation

In this section we describe the different ways that the NLPEC solver can reformulate an MPEC as an NLP. The description also applies to MCP models - just consider MCP to be an MPEC with a constant objective. The choice of reformulation, and the subsidiary choices each reformulation entails, are controlled by the options (see Section 4.1) mentioned throughout this section.

The original MPEC model is given as:

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x,y) \tag{1.1}$$

subject to the constraints

$$g(x,y) \leq 0 \tag{1.2}$$

and

$$y \text{ solves MCP}(h(x,\cdot), \mathbf{B}). \tag{1.3}$$

In most of the reformulations, the objective function (1.1) is included in the reformulated model without change. In some cases, it may be augmented with a penalty function. The variables x are typically called upper level variables (because they are associated with the upper level optimization problem) whereas the variables y are sometimes termed lower level variables.

The constraints (1.2) are standard nonlinear programming constraints specified in GAMS in the standard fashion. In particular, these constraints may be less than inequalities as shown above, or equalities or greater than inequalities. The constraints will be unaltered by all our reformulations. These constraints may involve both x and y, or just x or just y, or may not be present at all in the problem.

The constraints of interest are the equilibrium constraints (1.3), where (1.3) signifies that $y \in \mathbf{R}^m$ is a solution to the mixed complementarity problem (MCP) defined by the function $h(x,\cdot)$ and the box $\mathbf{B}$ containing (possibly infinite) simple bounds on the variables y. A point y with $a_i \leq y_i \leq b_i$ solves (1.3) if for each i at least one of the following holds:

$$\begin{array}{c} h_i(x,y) = 0 \\ h_i(x,y) > 0, \;\; y_i = a_i; \\ h_i(x,y) < 0, \;\; y_i = b_i. \end{array} \tag{1.4}$$

As a special case of (1.4), consider the case where $a = 0$ and $b = +\infty$. Since y_i can never be $+\infty$ at a solution, (1.4) simplifies to the nonlinear complementarity problem (NCP):

$$0 \leq h_i(x,y), 0 \leq y_i \text{ and } y_i h_i(x,y) = 0, i = 1, \ldots, m \tag{1.5}$$

namely that h and y are nonnegative vectors with h perpendicular to y. This motivates our shorthand for (1.4), the "perp to" symbol $\perp$:

$$h_i(x,y) \quad \perp \quad y_i \in [a_i, b_i] \tag{1.6}$$

The different ways to force (1.6) to hold using (smooth) NLP constraints are the basis of the NLPEC solver.

We introduce a simple example now that we will use throughout this document for expositional purposes:

$$\begin{array}{ll} \min\limits_{x_1,x_2,y_1,y_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & y_1 - y_2 + 1 \leq x_1 \perp y_1 \geq 0 \\ & x_2 + y_2 \perp y_2 \in [-1, 1] \end{array}$$

This problem has the unique solution $x_1 = 0$, $x_2 = -1$, $y_1 = 0$, $y_2 = 1$. Note that $f(x,y) = x_1 + x_2$ and $g(x,y) = x_1^2 + x_2^2 - 1$ are the objective function and the standard nonlinear programming constraints for this problem. The function $h(x,y)$ is given by:

$$h(x,y) = \begin{bmatrix} x_1 - y_1 + y_2 - 1 \\ x_2 + y_2 \end{bmatrix}$$

and

$$a = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = \begin{bmatrix} \infty \\ 1 \end{bmatrix}.$$

This example is written very succinctly in GAMS notation as:

```
$TITLE simple mpec example

variable f, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up =  1;

equations cost, g, h1, h2;

cost..  f =E= x1 + x2;
g..     sqr(x1) + sqr(x2) =L= 1;
h1..    x1 =G= y1 - y2 + 1;
h2..    x2 + y2 =N= 0;

model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min f;
```

Note that the equation `cost` is used to define f, the constraint `g` defines the function g, and h is defined by `h1` and `h2`. The complementarity constraints utilize the standard GAMS convention of specifying the orthogonality relationship between h and y in the `model` statement. The interpretation of the "." relies on the bounds a and b that are specified using `positive`, `negative`, or `lo` and `up` keywords in GAMS. Note that since `h2` really specifies a function h_2 and not a constraint $h_2(x,y) = 0$, we use the GAMS syntax `=N=` to ensure this is clear here. Since the relationships satisfied by h_1 and h_2 are determined by the bounds, `=G=` could also be replaced by `=N=` in `h1`.

In describing the various reformulations for (1.6), it is convenient to partition the y variables into free $\mathcal{F}$, lower bounded $\mathcal{L}$, upper bounded $\mathcal{U}$ and doubly bounded $\mathcal{B}$ variables respectively, that is:

$$\mathbf{B} := \{y = (y_\mathcal{F}, y_\mathcal{L}, y_\mathcal{U}, y_\mathcal{B}) : a_\mathcal{L} \le y_\mathcal{L},\ y_\mathcal{U} \le b_\mathcal{U},\ a_\mathcal{B} \le y_\mathcal{B} \le b_\mathcal{B}\}.$$

We will assume (without loss of generality) that $a_\mathcal{B} < b_\mathcal{B}$. If $a_i = b_i$ then (1.6) holds trivially for the index i and we can remove the constraint h_i and its corresponding (fixed) variable y_i from the model. The complementarity condition for variables in $y_i \in \mathcal{F}$ is simply the equality $h_i(x,y) = 0$ so these equality constraints are moved directly into the NLP constraints g of the original model as equalities. Thus, NLPEC needs only to treat the singly-bounded variables in $\mathcal{L}$ and $\mathcal{U}$ and the doubly-bounded variables in $\mathcal{B}$. In the above example, $\mathcal{L} = \{1\}$, $\mathcal{U} = \emptyset$ and $\mathcal{B} = \{2\}$.

3.1 Product reformulations

Product reformulations all involve products of y_i with h_i, or products of y_i with some auxiliary or slack variables that are set equal to h_i. The underlying point is that the constraints (1.3) are entirely equivalent to the following system of equalities and inequalities:

$$\begin{gathered} w_\mathcal{L} = h_\mathcal{L}(x,y),\ a_\mathcal{L} \le y_\mathcal{L},\ w_\mathcal{L} \ge 0 \text{ and } (y_\mathcal{L} - a_\mathcal{L})^T w_\mathcal{L} = 0 \\ v_\mathcal{U} = -h_\mathcal{U}(x,y),\ y_\mathcal{U} \le b_\mathcal{U},\ v_\mathcal{U} \ge 0 \text{ and } (b_\mathcal{U} - y_\mathcal{U})^T v_\mathcal{U} = 0 \\ w_\mathcal{B} - v_\mathcal{B} = h_\mathcal{B}(x,y),\ a_\mathcal{B} \le y_\mathcal{B} \le b_\mathcal{B},\ w_\mathcal{B} \ge 0,\ v_\mathcal{B} \ge 0 \\ (y_\mathcal{B} - a_\mathcal{B})^T w_\mathcal{B} = 0,\ (b_\mathcal{B} - y_\mathcal{B})^T v_\mathcal{B} = 0. \end{gathered} \tag{1.7}$$

Note that each inner product is a summation of products of nonnegative terms: a slack variable and the difference between a variable and its bound. In each of these products, either the slack variable or its complement must be zero in order to have a solution. Complementarity is forced by the multiplication of these two terms. The above reformulation is specified using option `reftype mult`.

There are a number of variations on this theme, all of which can be specified via an options file. All of the inner products could be put into the same equation, left as in (1.7) above, or broken out into individual products (one for each $i \in \mathcal{L} \cup \mathcal{U}$, two for each $i \in \mathcal{B}$). For example, the complementarity constraints associated with lower bounded variables involve nonnegativity of $w_{\mathcal{L}}$, $y_{\mathcal{L}} \geq a_{\mathcal{L}}$ and either of the following alternatives:

$$(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = \sum (i \in \mathcal{L}(y_i - a_i)w_i = 0$$

or

$$(y_i - a_i)w_i = 0,\ i = 1, \ldots, m$$

These different levels of aggregation are chosen using option `aggregate none|partial|full`.

Since all of the inner products in (1.7) involve nonnegative terms, we can set the inner products equal to zero or set them ≤ 0 without changing the feasible set. To choose one or the other, use the option `constraint equality|inequality`.

As a concrete example, consider the option file

```
reftype mult
aggregate none
constraint inequality
```

applied to the simple example given above. Such an option file generates the nonlinear programming model:

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 \leq \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1] \\ & (y_2 + 1)w_2 \leq \mu,\ (1 - y_2)v_2 \leq \mu \end{array} \tag{1.8}$$

By default, a single model is generated with the value μ set to 0. There are many examples (e.g. interior point codes, many LP and NLP packages, published results on reformulation approaches to MPEC) that illustrate the value of starting with a "nearly-complementary" solution and pushing the complementarity gap down to zero. For this reason, the inner products in (1.7) above are always set equal to (or $\leq$) a scalar μ instead of zero. By default μ is zero, but options exist to start μ at a positive value (e.g. `InitMu 1e-2`), to decrease it by a constant factor in a series of looped solves (e.g. `NumSolves 4`, `UpdateFac 0.1`), and to solve one last time with a final value for μ (e.g. `FinalMu 0`). If the following lines are added to the option file

```
initmu 1.0
numsolves 4
```

then five consecutive solves of the nonlinear program (1.8) are performed, the first one using $\mu = 1$ and each subsequent solve dividing μ by 10 (and starting the NLP solver at the solution of the previous model in this sequence).

As a final example, we use a combination of these options to generate a sequence of nonlinear programs whose solutions attempt to trace out the "central path" favored by interior point and barrier algorithms:

```
reftype mult
constraint equality
initmu 1.0
numsolves 4
updatefac 0.1
finalmu 1e-6
```

produces 6 nonlinear programs of the form

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1,1],\ (y_2+1)w_2 = \mu,\ (y_2-1)v_2 = \mu \end{array}$$

for values of $\mu = 1, 0.1, 0.01, 0.001, 0.0001$ and $1e-6$.

3.1.1 Slacks and doubly bounded variables

Slack variables can be used to reduce the number of times a complex nonlinear expression appears in the nonlinear programming model, as was carried out in (1.7). For a simpler illustrative example the NCP constraints (1.5) are equivalent to the constraints:

$$w_i = h_i(x,y), 0 \leq w_i, 0 \leq y_i \text{ and } y_i w_i = 0, i = 1, \ldots, m$$

This reformulation has an additional equality constraint, and additional variables w, but the expression h_i only appears once. There are cases when this formulation will be preferable, and the simple option `slack none`|`positive` controls the use of the w variables.

When there are doubly bounded variables present, these two slack options work slightly differently. For the `positive` case, the reformulation introduces two nonnegative variables w_i and v_i that take on the positive and negative parts of h_i at the solution as shown in (1.7). Since this is the default value of the option `slack`, the example (1.8) shows what ensues to both singly and doubly bounded variables under this setting.

For the case `slack none`, Scholtes proposed a way to use a multiplication to force complementarity that requires no slack variables:

$$h_i \quad \perp \quad a_i \leq y_i \leq b_i \iff$$

$$a_i \leq y_i \leq b_i,\ (y_i - a_i)h_i \leq \mu,\ (y_i - b_i)h_i \leq \mu \tag{1.9}$$

Note that unlike the inner products in Section 3.1, we can expect that one of the inequalities in (1.9) is unlikely to be binding at a solution (i.e. when h_i is nonzero). Therefore, we cannot use an equality in this reformulation, and furthermore the products must not be aggregated. Thus, if you use this option, the reformulation automatically enforces the additional options `constraint inequality` and `aggregate none` on the doubly bounded variables, even if the user specifies a conflicting option. Thus the option file

```
reftype mult
slack none
```

results in the model

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\ & (x_1 - y_1 + y_2 - 1)y_1 = \mu \\ & y_2 \in [-1,1],\ (y_2+1)(x_2+y_2) \leq \mu,\ (y_2-1)(x_2+y_2) \leq \mu \end{array}$$

Note that the complementarity constraint associated with y_1 is an equality (the default) while the constraints associated with y_2 are inequalities for the reasons outlined above.

In the case of doubly bounded variables, a third option is available for the slack variables, namely `slack one`. In this case, only one slack is introduced, and this slack removes the need to write the function h_i twice in the reformulated model as follows:

$$h_i(x,y) \quad \perp \quad a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i,\ w_i = h_i(x,y),\ (y_i - a_i)w_i \leq \mu,\ (y_i - b_i)w_i \leq \mu$$

Note that the slack variable w that is introduced is a free variable. It is not known before solving the problem whether w_i will be positive or negative at the solution.

We take this opportunity to introduce a simple extension to our option mechanism, namely the ability to set the options for singly and doubly bounded variables differently. For example, the option file

```
reftype mult
slack positive one
```

sets the option `slack positive` for the singly bounded variables and the option `slack one` for the doubly bounded variables resulting in the model

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu_1 \\ & w_2 = x_2 + y_2,\ y_2 \in [-1, 1],\ (y_2 + 1)w_2 \leq \mu_2,\ (y_2 - 1)w_2 \leq \mu_2 \end{array}$$

Additional options such as

```
initmu 1.0 3.0
numsolves 2
updatefac 0.1 0.2
```

allow the values of μ for the singly and doubly bounded variables to be controlled separately. In this case μ_1 takes on values of 1, 0.1 and 0.01, while μ_2 takes on values 3.0, 0.6 and 0.12 in each of the three nonlinear programming models generated.

3.2 NCP functions

An NCP-function is a function $\phi(r, s)$ with the following property:

$$\phi(r, s) = 0 \Longleftrightarrow r \geq 0, s \geq 0, rs = 0$$

Clearly, finding a zero of an NCP-function solves a complementarity problem in (r, s). We can replace the inner products of nonnegative vectors in (1.7) with a vector of NCP functions whose arguments are complementary pairs, e.g. $(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = 0$ becomes $\phi(y_i - a_i, w_i) = 0, i \in \mathcal{L}$ and arrive at another way to treat the complementarity conditions. Note that an NCP function forces both nonnegativity and complementarity, so constraints to explicitly force nonnegativity are not required, though they can be included.

Examples of NCP functions include the min function, min(r,s), and the Fischer-Burmeister function

$$\phi(r, s) = \sqrt{r^2 + s^2} - r - s$$

There is no requirement that an NCP function be nonnegative everywhere (it may be strictly negative at some points), so there is little point in setting the option `constraint`; it will automatically take on the value `constraint equality`. NCP functions cannot be aggregated, so the `aggregate` option will always be set to `none`.

Since the arguments to the NCP functions are going to be nonnegative at solution, we cannot use the functions h_i directly in the case of doubly-bounded variables. We must use slacks $w - v = h_i$ to separate h_i into its positive and negative parts (but see Section 3.2.1 below). The slacks can be `positive` or `free`, since the NCP function will force positivity at solution. For the singly-bounded variables, slacks are optional, and can also be `positive` or `free`.

Both of the NCP functions mentioned above suffer from being non-differentiable at the origin (and at points where $r = s$ for the min function). Various smoothed NCP-functions have been proposed that are differentiable. These smooth functions are parameterized by μ, and approach the true NCP-function as the smoothing parameter approaches zero. For example, the Fischer-Burmeister function includes a perturbation μ that guarantees differentiability:

$$\phi_{FB}(r, s) := \sqrt{r^2 + s^2 + 2\mu} - (r + s). \tag{1.10}$$

You can choose these particular NCP functions using option `RefType min|FB|fFB`. The difference between the last two is that `RefType FB` writes out GAMS code to compute the function ϕ_{FB}, while `RefType fFB` makes use of a GAMS intrinsic function NCPFB(r,s,mu) that computes ϕ_{FB} internally. In general, using the GAMS intrinsic function should work better since the intrinsic can guard against overflow, scale the arguments before computing the function, and use alternative formulas that give more accurate results for certain input ranges.

As an example, the option file

```
reftype fFB
slack free
initmu 1e-2
```

generates the reformulation

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1 \\ & \phi_{FB}(w_1, y_1, \mu) = 0 \\ & w_2 - v_2 = x_2 + y_2 \\ & \phi_{FB}(y_2 + 1, w_2, \mu) = 0,\ \phi_{FB}(1 - y_2, v_2, \mu) = 0 \end{array}$$

with a value of $\mu = 0.01$. Following a path of solutions for decreasing values of μ is possible using the options discussed above.

Each of the two arguments to the NCP function will be nonnegative at solution, but for each argument we have the option of including a nonnegativity constraint explicitly as well. This results in the 4 values for the option `NCPBounds none|all|function|variable`. When no slacks are present, this option controls whether to bound the function h_i as well as including it in the NCP function, e.g. $h_i \geq 0, \phi(h_i, y_i - a_i) = 0$. When slacks are present, we require that the slack setting be consistent with the bound setting for the function argument to the NCP function, where `NCPBounds none|variable` is consistent with free slack variables and `NCPBounds all|function` is consistent with positive slack variables.

Thus, the option file

```
reftype min
slack positive
NCPBounds function
```

generates the reformulation

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0 \\ & \min(w_1, y_1) = \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0 \\ & \min(y_2 + 1, w_2) = \mu,\ \min(1 - y_2, v_2) = \mu \end{array}$$

The `NCPBounds function` option means that the variable argument to the NCP function (in this case y) does not have its bounds explicitly enforced. It should be noted that this nonlinear program has nondifferentiable constraints for every value of μ. For this reason, the model is constructed as a `dnlp` model (instead of an `nlp` model) in GAMS.

A smoothed version of the min function was proposed by Chen & Mangasarian:

$$\phi_{CM}(r, s) := r - \mu \log(1 + \exp((r - s)/\mu)). \tag{1.11}$$

This function is not symmetric in its two arguments, so $\phi_{CM}(r, s) \neq \phi_{CM}(s, r)$. For this reason, we distinguish between the two cases. Unlike the Fischer-Burmeister function ϕ_{FB}, ϕ_{CM} is not defined in the limit (i.e. for $\mu = 0$) if you use GAMS code to compute it. However, the GAMS intrinsic NCPCM(r,s,mu) handles this limit case internally. The option `RefType CMxf|CMfx|fCMxf|fCMfx` chooses a reformulation based on the function ϕ_{CM}. Again, the last two choices use the GAMS intrinsic function.

3.2.1 Doubly bounded variables

Like the mult reformulation (1.7), reformulations using NCP functions are appropriate as long as we split the function h_i matching a doubly-bounded variable into its positive and negative parts $w_i - v_i = h_i$. To avoid this, Billups has proposed using a composition of NCP functions to treat the doubly-bounded case:

$$h_i \quad \perp \quad a_i \leq y_i \leq b_i \iff$$

$$\phi_{FB}(y_i - a_i, \phi_{FB}(b_i - y_i, -h_i)) = 0 \tag{1.12}$$

Use option `RefType Bill|fBill` to choose such a reformulation for the doubly-bounded variables. The first option value writes out the function in explicit GAMS code, while the second writes it out using the GAMS intrinsic function NCPFB.

3.3 Penalty functions

All of the reformulations discussed so far have reformulated the complementarity conditions as constraints. It is also possible to treat these by moving them into the objective function with a penalty parameter $1/\mu$: as μ goes to zero, the relative weight placed on complementarity increases. Ignoring the NLP constraints, we can rewrite the original MPEC problem as

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x,y) + \frac{1}{\mu}((y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} + (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} + (y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} + (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}}) \tag{1.13}$$

subject to the constraints

$$\begin{array}{c} g(x,y) \leq 0 \\ w_{\mathcal{L}} = h_{\mathcal{L}}(x,y),\ a_{\mathcal{L}} \leq y_{\mathcal{L}},\ w_{\mathcal{L}} \geq 0 \\ v_{\mathcal{U}} = -h_{\mathcal{U}}(x,y),\ y_{\mathcal{U}} \leq b_{\mathcal{U}},\ v_{\mathcal{U}} \geq 0 \\ w_{\mathcal{B}} - v_{\mathcal{B}} = h_{\mathcal{B}}(x,y)\ a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}},\ w_{\mathcal{B}} \geq 0, v_{\mathcal{B}} \geq 0 \end{array} \tag{1.14}$$

Choose this treatment using option `refType penalty`. The options `aggregate` and `constraint` are ignored, since the inner products here are all aggregated and there are no relevant constraints. It is possible to do a similar reformulation without using slacks, so the options `slack none|positive` can be used in conjunction with this reformulation type.

The following option file shows the use of the `penalty` reformulation, but also indicates how to use a different reformulation for the singly and doubly bounded variables:

```
reftype penalty mult
slack none *
initmu 1.0
numsolves 2
updatefac 0.1 0.2
```

applied to the simple example given above. The "*" value allows the `slack` option to take on its existing value, in this case `positive`. Such an option file generates the nonlinear programming model:

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_2,v_2} & x_1 + x_2 + 1/\mu_1 y_1(x_1 - y_1 + y_2 - 1) \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1,1] \\ & (y_2+1)w_2 \leq \mu_2,\ (1-y_2)v_2 \leq \mu_2 \end{array}$$

The penalty parameter μ_1 is controlled separately from the doubly bounded constraint parameter μ_2. For consistency with other options, the penalty parameter in the objective is $1/\mu$ meaning that as μ_1 tends to zero, the penalty increases. The option `initmu` has only one value, so both the singly and doubly bounded μ values are initialized to 1. In the above example, three solves are performed with $\mu_1 = 1, 0.1$ and 0.01 and $\mu_2 = 1, 0.2$ and 0.04.

3.4 Testing for complementarity

In some cases a solution to the reformulated model may not satisfy the complementarity constraints of the original MPEC, e.g. if a large penalty parameter is used in the reformulation. It can also happen that the solution tolerances used in the NLP solver allow solutions with small error in the NLP model but large error in the original MPEC. For example if $x = f(x) = .001$ then the NLP constraint $xf(x) = 0$ may satisfy the NLP feasibility tolerance but it's not so easy to claim that either x or $f(x)$ is zero. The NLPEC solver includes a check that the proposed solution does in fact satisfy the complementarity constraints. The complementarity gap is computed using the definition common to all GAMS MCP solvers in computing the *objval* model attribute for an MCP model. The tolerance used for this complementarity gap can be adjusted using the `testtol` option.

4 Options

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

For most GAMS solvers, the use of an options file is discouraged, at least for those unfamiliar with the solver. For NLPEC, however, we expect that most users will want to use an options file from the very beginning. NLPEC is as much a tool for experimentation as it is a solver, and as such use of the options file is encouraged.

Option values can take many different types (e.g. strings, integers, or reals). Perhaps the most important option to remember is one with no value at all: the `help` option. `Help` prints a list of the available options, along with their possible values and some helpful text. The options file is read sequentially, so in case an option value is set twice, the latter value takes precedence. However, any consistency checks performed on the options values (e.g. `RefType fBill` cannot be used with `aggregate full`) are made after the entire options file is read in, so the order in which different options appear is not important, provided the options are not specified twice.

4.1 Setting the Reformulation Options

While NLPEC has many options, there is a small set of five options that, taken together, serve to define the type of reformulation used. Listed in order of importance (highest priority items first), these *reformulation options* are the `RefType`, `slack`, `constraint`, `aggregate` and `NCPBounds` options. In some cases, setting the highest-priority option `RefType` is enough to completely define a reformulation (e.g. `RefType penalty` in the case of doubly-bounded variables). In most cases though, the lower-priority options play a role in defining or modifying a reformulation. It's useful to consider the reformulation options in priority order when creating option files to define reformulations.

Some of the combinations of the reformulation options don't make sense. For example, the use of an NCP function to force complementarity between its two input arguments requires a separate function for each complementary pair, so setting both `RefType min` and `aggregate full` is inconsistent. NLPEC implements consistency checks on the reformulation options using the priority order: Given a consistent setting of the higher priority options, the next-highest priority option is checked and, if necessary, reset to be consistent with the items of higher priority. The end result is a set of consistent options that will result in a working reformulation. NLPEC prints out the pre- and post-checked sets of reformulation options, as well as warning messages about changes made. In case you want to use an option that NLPEC doesn't think is consistent, you can use the `NoCheck` option: this supresses the consistency checks.

Each of the reformulation options in the table below takes two values - one for the singly-bounded variables in $\mathcal{L} \cup \mathcal{U}$ and another for the doubly-bounded variables in $\mathcal{B}$. If one option value appears, it sets both option values. When setting both option values, use an asterisk "*" to indicate no change. So for example, an option file

```
RefType  fCMxf
RefType  *  fBill
```

first sets the `RefType` to `fCMxf` for all variable types, and then resets the `RefType` to `fBill` for doubly-bounded variables.

Option	Description	Default
`reftype`	Determines the type of reformulation used - see Section 3 for details. Our notation and descriptions are taken from a special case of the MPEC, the NCP: find $x \geq 0, f(x) \geq 0, x^T f(x) = 0$. `mult` inner-product reformulation $x^T f = 0$ (Section 3.1) `min` NCP-function $\min(x, f)$ (Section 3.2) `CMxf` Chen-Mangasarian NCP-function $\phi_{CM}(x, f) := x - \mu \log(1 + \exp((x - f)/\mu))$, written explicitly in GAMS code (Section 3.2) `CMfx` Chen-Mangasarian NCP-function $\phi_{CM}(f, x) := f - \mu \log(1 + \exp((f - x)/\mu))$, written explicitly in GAMS code (Section 3.2) `fCMxf` Chen-Mangasarian NCP-function $\phi_{CM}(x, f) := x - \mu \log(1 + \exp((x - f)/\mu))$, using GAMS intrinsic NCPCM(x,f,μ) (Section 3.2) `fCMfx` Chen-Mangasarian NCP-function $\phi_{CM}(f, x) := f - \mu \log(1 + \exp((f - x)/\mu))$, using GAMS intrinsic NCPCM(f,x,μ) (Section 3.2) `FB` Fischer-Burmeister NCP-function $\phi_{FB}(x, f) := \sqrt{x^2 + f^2 + 2\mu} - (x + f)$, written explicitly in GAMS code (Section 3.2) `fFB` Fischer-Burmeister NCP-function $\phi_{FB}(x, f) := \sqrt{x^2 + f^2 + 2\mu} - (x + f)$, using GAMS intrinsic NCPFB(x,f,μ) (Section 3.2) `Bill` Billups function for doubly-bounded variables, written explicitly in GAMS code (Section 3.2.1) `fBill` Billups function for doubly-bounded variables, using GAMS intrinsic NCPFB(x,f,μ) (Section 3.2.1) `penalty` Penalization of non-complementarity in objective function (Section 3.3)	mult/mult
`slack`	Determines if slacks are used to treat the functions h_i. For single-bounded variables, we use at most one slack (either free or positive) for each h_i. For doubly-bounded variables, we can have no slacks, one slack (necessarily free), or two slacks (either free or positive) for each h_i. `none` no slacks will be used `free` free slacks will be used `positive` nonnegative slacks will be used `one` one free slack will be used for each h_i in the doubly-bounded case.	positive/positive

`constraint`	Determines if certain constraints are written down using equalities or inequalities. E.g. to force $w \geq 0$ and $y \geq 0$ to be complementary we can write either $w^T y \leq 0$ or $w^T y = 0$. This option only plays a role when bounding a quantity whose sign cannot be both positive and negative and which must be 0 at a solution.
	`equality`
	`inequality`
		equality/equality

`aggregate`	Determines if certain constraints are aggregated or not. E.g. to force $w \geq 0$ and $y \geq 0$ to be complementary we can write either $w^T y \leq 0$ or $w_i^T y_i = 0, \forall i$. `none` Use no aggregation `partial` Aggregate terms in $\mathcal{L} \cup \mathcal{U}$ separately from those in $\mathcal{B}$ `full` Use maximum aggregation possible	none/none
`NCPBounds`	Determines which of the two arguments to an NCP function $\phi(r, s)$ are explicitly constrained to be nonnegative (see Section 3.2). The explicit constraints are in addition to those imposed by the constraint $\phi(r, s) = 0$, which implies nonnegativity of r and s. `none` No explicit constraints `function` Explicit constraint for function argument `variable` Explicit constraint for variable argument `all` Explicit constraints for function and variable arguments	none/none

4.2 General Options

Option	Description	Default
allsolves	In case multiple (looped) solves are specified, the default is to skip subsequent solves when any solve terminates without getting a solution. Setting this flag removes the check and all solves are done, regardless of previous failures.	no
finalmu	Final value of the parameter μ. If specified, an extra solve is carried out with μ set to this value. Can be set independently for singly and doubly bounded variables.	none
initmu	Initial value of the parameter μ. A single solve of the nonlinear program is carried out for this value. Note that μ must be positive for some settings of reftype, e.g. penalty. Can be set independently for singly and doubly bounded variables.	0.0
initslo	The lower bound for any artificials that are added.	0
initsup	The upper bound for any artificials that are added.	inf
nocheck	Turns off the consistency checks for the reformulation options (see Section 4.1).	off
numsolves	Number of extra solves carried out in a loop. This should be set in conjunction with the updatefac option.	0
subsolver	Selects NLP or DNLP subsolver to run. If no subsolver is chosen, the usual procedure for setting the solver is used.	auto
subsolveropt	Selects an options file for use with the NLP or DNLP subsolver.	0
testtol	Zero tolerance used for checking the complementarity gap of the proposed solution to the MPEC.	1e-5
updatefac	The factor that multiplies μ before each of the extra solves triggered by the numsolves option. Can be set independently for singly and doubly bounded variables.	0.1

4.3 The Outdated equreform Option

In the early versions of NLPEC the only way to set the reform type was via the equreform option. Each valid equreform value represented a preselected combination of the options from Section 4.1. This made it difficult to experiment with combinations not preselected, so the options in Section 4.1 were added. Be default, the equreform option has value 0 and is not used. To get the old behavior, set equreform to a positive value - this will force the options in Section 4.1 to be ignored. The general options in Section 4.2 are used no matter how the reformulation type is selected - via RefType or equreform.

Option	Description	Default
equreform	Old way to set the type of reformulation used.	0

The values allowed for equreform and their implications are given below.

	L/U				B			
equref	**reftype**	**sign**	**slacks**	**free-y**	**reftype**	**sign**	**slacks**	**free-y**
1	$<,>_i$	$=\mu$	bnd		$<,>_i$	$=\mu$	bnd	
2	$<,>_i$	$<=\mu$	bnd		$<,>_i$	$<=\mu$	bnd	
3	$<,>_i$	$=\mu$	bnd		Scholtes	$<=\mu$	one	
4	$<,>_{L+U+B}$	$=\mu$	bnd		$<,>_{L+U+B}$	$=\mu$	bnd	
5	$<,>_{L+U+B}$	$=\mu$	none		$<,>_{L+U+B}$	$=\mu$	bnd	
6	$<,>_{L+U}$	$=\mu$	none		Scholtes	$<=\mu$	one	
7	$<,>_{L+U}$	$<=\mu$	none		Scholtes	$<=\mu$	none	
8	$<,>_i$	$=\mu$	none		Scholtes	$<=\mu$	none	
9	$<,>_{\text{obj}}$		bnd		$<,>_{\text{obj}}$		bnd	
10	$<,>_{\text{obj}}$		none		$<,>_{\text{obj}}$		bnd	
11	$<,>_i$	$=\mu$	none		$<,>_i$	$=\mu$	bnd	
12	F-B	$=0$	none	free	F-B	$=0$	free	free
13	F-B	$=0$	none	free	Billups	$=0$	none	free
14	min	$=\mu$	free	free	min	$=\mu$	free	free
15	min	$<=\mu$	bnd		min	$<=\mu$	bnd	
16	C-M(x,f)	$=0$	free	free	C-M(x,f)	$=0$	free	free
17	C-M(x,f)	$=0$	bnd		C-M(x,f)	$=0$	bnd	
18	$<,>_{L,U}$	$<=\mu$	none		$<,>_B$	$<=\mu$	bnd	
19	$<,>_i$	$<=\mu$	none		$<,>_i$	$<=\mu$	bnd	
20	$<,>_{L,U}$	$<=\mu$	bnd		$<,>_B$	$<=\mu$	bnd	
21	$<,>_{L+U+B}$	$<=\mu$	bnd		$<,>_{L+U+B}$	$<=\mu$	bnd	
22	F-B	$=0$	bnd		F-B	$=0$	bnd	
23	F-B	$=0$	free	free	F-B	$=0$	free	free
24	C-M(f,x)	$=0$	none	free	C-M(f,x)	$=0$	free	free
25	C-M(f,x)	$=0$	none		C-M(f,x)	$=0$	bnd	
26	NCPF	$=0$	none	free	NCPF	$=0$	free	
27	NCPF	$=0$	none	free	Billups†	$=0$	none	free
28	NCPF	$=0$	bnd		NCPF	$=0$	bnd	
29	NCPF	$=0$	free	free	NCPF	$=0$	free	free
30	NCPCM(x,f)	$=0$	free	free	C-M(x,f)	$=0$	free	free
31	NCPCM(x,f)	$=0$	bnd		C-M(x,f)	$=0$	bnd	
32	NCPCM(f,x)	$=0$	none	free	NCPCM(f,x)	$=0$	free	free
33	NCPCM(f,x)	$=0$	none		NCPCM(f,x)	$=0$	bnd	

5 Open Architecture

In this section we describe the architecture of the NLPEC solver, i.e. the way the solver is put together. This should be useful to anybody using NLPEC for experiments or to those wanting to know the details of how NLPEC works.

The foundation for the NLPEC solver is the software library (also used in the GAMS/CONVERT solver) that allows us to write out a scalar GAMS model that is mathematically equivalent to the original, or to write out selected pieces of such a model. Using this software, NLPEC creates a GAMS NLP model (default name: nlpec.gms) using one of the reformulation strategies from Section 3. This model may contain many new variables and/or equations, but it will surely contain the (non)linear expressions defining the original model as well. Once the model nlpec.gms has been created, NLPEC calls gams to solve this model, using the current NLP solver. After the model has solved, NLPEC reads the NLP solution, extracts the MPEC solution from it, and passes this MPEC solution back to GAMS as it terminates.

There are a number of advantages to this architecture. First, its openness makes it easy to see exactly what reformulation is being done. The intermediate NLP file nlpec.gms is always available after the run for those wishing to know the details about the reformulation or for debugging in case things didn't work out as expected. It would also be possible to modify this file to do some quick and dirty experiments with similar reformulation strategies. Another advantage is the variety of NLP solvers that can be plugged in to solve the reformulated

model. There is no need to program (and debug) an interface to an NLP package to run experiments with an NLP solver - the existing GAMS link is all that is needed. It is also easy to experiment with non-default solver options that may be more appropriate for reformulated MPEC models or for a particular choice of reformulation.