Infeasibilities in the market model of CAPRI – how they are dealt with

- Wolfgang Britz and Heinz-Peter Witzke –

Background

The multi-market model of CAPRI currently features about 36.000 equations and the exact number of variables (square system). Some of the equations, especially those relating to TRQs, are highly non-linear, and in some cases the solver will end up with infeasibilities. It is very hard to judge in such cases if the final solution can be used for policy analysis or not: a small infeasibility in a price transmission may relate to a large problem in market management - say, the market price is below the administrative one – whereas in other cases, a few hundred tons of market imbalances may be acceptable. As the user of the model cannot be expected to analyze in detail the solution listing, the code should comprise elements which ensure feasibility as far as possible.

Remedies against infeasibilities

Infeasibilities provoked by bounds

The scaling and "search" tactic of solvers as CONOPT depends inter alia on bounds introduced on the variables. In a square system, such bounds, once they become binding, must lead to infeasibility as there are not longer enough degree of freedom to find a solution.

On the other, using +/- inf bounds on a all variables is not a solution, for two reasons. Firstly, certain operation like divisions or taking square roots and exponents require variables in a certain domain. If the solver will during its search for a solution evaluate function based on variables or intermediate results from several variables provoking undefined operations, it cannot maintain its usual algorithm, and will almost certainly stop with infeasibilities. Accordingly, security bounds are introduced on all variables (see arm\setStartVal.gms, arm\setStartValPol.gms). Those bounds should normally guarantee feasibility and avoid math error trapping by undefined operations. However, it cannot be excluded that the solver will move a variable

against the bounds defined in the program. If that happens, a small program (arm\widen_bounds) checks if a variable hits its upper or lower bound, and if a marginal value is attached to it, and then widen the bounds, see example code below.

```
Hcon.LO(RMS,XXX)      $ ( (ABS(Hcon.m(RMS,XXX)) GT 0.00)      and (Hcon.LO(RMS,XXX) EQ Hcon.L(RMS,XXX)) )          = Hcon.LO(RMS,XXX) * 0.001;
Hcon.UP(RMS,XXX)      $ ( (ABS(Hcon.m(RMS,XXX)) GT 0.00)      and (Hcon.UP(RMS,XXX) EQ Hcon.L(RMS,XXX)) )          = Hcon.UP(RMS,XXX) * 1000.;
Hcon.FX(RMS,XXX)      $ ( (ABS(Hcon.m(RMs,XXX)) GT 0.00)      and (Hcon.L(RMS,XXX)  LE 1.E-10)          )          = HCon.L(RMS,XXX);
```

The example also shows that once a variable may hit a very small lower bound, it may be taken out completely form the solution process. Naturally, the equations in “arm\market_model.gm” must be defined such that the equation linked to fixed variable is taken out from the model, see below:

```
*
* ----- definition of human consumption for the Generalised Leontief expenditure funtion
*
 XiS_(RMS,XXX)  $ ( (Hcon.LO(RMS,XXX) ne HCon.UP(RMS,XXX)) and DATA(RMS,"HCon",XXX,"CUR")) ..
*
     Hcon(RMS,XXX) =E=
       (GIs(RMS,XXX)/Gs(RMS) * ( DATA(RMS,"Ince","Levl","CUR")/DATA(RMS,"INHA","LEVL","CUR") - FS(RMS))
          + PD(RMS,XXX,"CUR")) * DATA(RMS,"INHA","LEVL","CUR")/1000.;
```

However, as some variable require “security” bounds to avoid math error trapping, it cannot be excluded that infeasibilities cannot be removed. Currently, such candidates are consumer prices (CPRI) of which is square root is taken based on the Generalized Leontief functional form used in final demand. Further on, trade flows (Flows), domestic sales (DSales) and the Armington aggregate of trade flows (Arm2) must be defined strictly positive, along with the related prices used in the share equations.

Reducing non-smoothness

Gradient based solvers as CONOPT are vulnerable in case of highly non-linear relations or derivatives. In some cases, infeasible solutions can be found where no variable hits any bound.

Currently, there are two cases where code reduces non-smoothness in order to help the solver: the TRQ fudging function and the fudging of the complex tariff system for fruits & vegs, as shown below:

```
TRQSlope(RM,XXX) $ (     (Tariffa.m(RM,"RW",XXX) GT 0)
                      or (Tariffs.m(RM,"RW",XXX) GT 0)
                      or SUM(RM1 $ (Tariffa.M(RM,RM1,XXX) GT 0), 1)
                      or SUM(RM1 $ (Tariffs.M(RM,RM1,XXX) GT 0), 1))
  = MAX(100,TRQSlope(RM,XXX) - 100);
```

And

```
*
*  -- flatten fugding of tariff determination under EU fruits & vegs import regime if infeasible
*
  EntryPriceFac(RM,RM1,XXX,"CUR") $ ( (DATA(RM,"TriggerP",XXX,"CUR") gt eps) and DATAI(RM,RM1,XXX,"CUR")
                                     and (ABS(EntryPrice1_.M(RM,RM1,XXX)) GT eps))
   = EntryPriceFac(RM,RM1,XXX,"CUR") * 0.9;
```

Whereas the changes to the TRQs are reset in each iteration before the market model is solved again, the changes to Fruits & vegs are retained (possibly should be changed).

The problem with reducing the non-smoothness is the fact that the parameterization of the model is changed, i.e. the model with and without those changes will give other results under the same policy scenario. Consequently, comparing two policy scenarios, some of the differences shown could be triggered by reduced non-smoothness in between the scenarios. Accordingly, it would be best to reset such changes during the iterations back to the settings used in the baseline wherever possible.

Introducing slacks

The by far most straightforward way to increase infeasibility is to convert some of the equation into inequalities by introducing slacks. By doing so, additional degrees of freedom are introduced for the solver, and feasibility can be achieved. The down side of slacks is the fact that indeed the desired functional and logical relations between the variable are softened, e.g. supply is not longer strictly based on the underlying supply function. In order to reduce that effect, one may add an objective function to the square system minimizing the size of the slacks. That tactic is introduced for oilseed processing which has proven to be regular source of infeasibilities in the past:

```
*
*     ---- production of oils and cakes (Leontief)
*
 ProcO_(RMS,XXX) $ ( (CAK(XXX) or OIL(XXX)) and SUM(SED $ ( SED_TO_CAK(SED,XXX) OR SED_TO_OIL(SED,XXX)),
                                                            DATA(RMS,"Proc",SED,"CUR")*DATA(RMS,"Prcy",XXX,"CUR"))
                           and   (Production.lo(RMS,XXX) ne Production.up(RMS,XXX))
                           and   DATA(RMS,"Prod",XXX,"CUR")) ..

     Production(RMS,XXX) =E= SUM(SED $ ( SED_TO_CAK(SED,XXX) OR SED_TO_OIL(SED,XXX)),
                                                              Proc(RMS,SED) * (Prcy(RMS,XXX)+Fudge(RMS,XXX)));
                        T
```

Those slacks are also added to the objective function:

```
FLIPFLOP_           .. FLIPFLOP =E= 10
                         + SUM( (RMS,XXX)
                                  $ ( (CAK(XXX) or OIL(XXX)) and SUM(SED $ ( SED_TO_CAK(SED,XXX) OR SED_TO_OIL(SED,XXX)),
                                                                       DATA(RMS,"Proc",SED,"CUR")*DATA(RMS,"Prcy",XXX,"CUR"))
                                                and   (Production.lo(RMS,XXX) ne Production.up(RMS,XXX))
                                                and   DATA(RMS,"Prod",XXX,"CUR")), SQR(Fudge(RMS,XXX) * DATA(RMS,"Prod",XXX,"CUR")));
```

The slacks (fudge) are set to zero in “arm\simu_market.gms” before feasibility checks are started, given the model a chance to find a solution without slacks even if those

were introduced in earlier iterations. The introduction of the slacks is introduced in "arm\widen_bounds":

```
*
* --- introduce stepwise some slack in oilseed processing, starting with very small corridor, and only
*     for the region/product combination which reach 20% of the marginal value of the highest constraints (reduction of infeasibilies)
*
  Fudge.lo(RMS,XXX)     $ ( ABS(ProcO_.M(RMS,XXX)) GT SMAX(RMS1, ABS(ProcO_.M(RMS1,XXX))/5)) = - max(1.E-8,abs(Fudge.l(RMS,XXX)))*10;
  Fudge.up(RMS,XXX)     $ ( ABS(ProcO_.M(RMS,XXX)) GT SMAX(RMS1, ABS(ProcO_.M(RMS1,XXX))/5)) = + max(1.E-8,abs(Fudge.l(RMS,XXX)))*10;
```

The slacks are fixed during the final solution of the market model after the pre-step – see explanation below – to reduce solution time.

Speeding up the solution process by pre-step solving

A key issue in all the problems discussed above is solution time. The solver may require several minutes for the complete model until the minimal sum of infeasibilities had been found by the solver. Therefore, and more generally, to reduce solution time, pre-steps had been introduced. Those are based on the assumption that cross-price between certain groups of products are generally small, so that the overall problem can be portioned in groups solved independently, and once a solution is found for all of those groups, the full model can be solved must faster.

Those groups are defined in "sets.gms":

```
SET XXGRP / XMEAT "Meats"
            XMILK "Dairy"
            XCERE "Cereals"
            XVEGE "Vegetables"
            XFRUI "Fruits"
            XOILS "Oilseeds, cakes and oils"
            XREST "Other products"
          /;
ALIAS (XXGRP,XXGRP1);

SET XXGRP_XX(*,XX)

    / XCERE.(WHEA,BARL,RYEM,OATS,MAIZ,OCER,RICE)
      XMEAT.(BEEF,PORK,POUM,SGMT,EGGS)
      XMILK.(FRMI,CHES,BUTT,CREM,SMIP,WMIP,COCM,MILK)
      XFRUI.(APPL,CITR,TAGR,OFRU)
      XVEGE.(TOMA,TABO,OVEG,TWIN,OLIO)
      XOILS.(SOYA,SUNF,RAPE,SOYO,SUNO,RAPO,SOYC,SUNC,RAPC)
      XREST.(POTA,SUGA,PULS,TOBA,TEXT)
    /;
```

And in "arm\simu_market.gms", there are two solution blocks: one relating to the pre-steps, and a second one relating to the full model. During the solution blocks of single blocks, the cross-price effects and those quantity variables entering behavioral equations on the right side which are not in the block are fixed:

```
*
*       --- fix cross price effects and production showing up in behavioural equation for feed use
*           for all products (will be released for products in current pre-solve step)
*
     ProcMarg.FX(RMS,XX)     = ProcMarg.L(RMS,XX);
     Cpri.FX(RMS,XX)         = Cpri.L(RMS,XX);
     PPri.FX(RMS,XX)         = PPri.L(RMS,XX);
     Arm1P.FX(RM,XX)         = Arm1P.L(RM,XX);
     Production.FX(RMS,XX)   = Production.L(RMS,XX);
```

And only the variables in the current block are allowed to move:

```
*
*          --- define which products to include in current pre-solve
*
           XXX(XX) $ XXGRP_XX(XXGRPS,XX) = YES;
*
*          --- re-introduce the normal bounds for the products in the current pre-solve
*
           ProcMarg.LO(RMS,XXX)     =    Bounds(RMS,"ProcMarg",XXX,"LO");
           ProcMarg.UP(RMS,XXX)     =    Bounds(RMS,"ProcMarg",XXX,"UP");

           CPri.LO(RMS,XXX)         =    Bounds(RMS,"CPri",XXX,"LO");
           CPri.UP(RMS,XXX)         =    Bounds(RMS,"CPri",XXX,"UP");

           PPri.LO(RMS,XXX)         =    Bounds(RMS,"PPri",XXX,"LO");
           PPri.UP(RMS,XXX)         =    Bounds(RMS,"PPri",XXX,"UP");

           Arm1P.LO(RM,XXX)         =    Bounds(RM,"Arm1P",XXX,"LO");
           Arm1P.UP(RM,XXX)         =    Bounds(RM,"Arm1P",XXX,"UP");

           Production.LO(RMS,XXX)   =    Bounds(RMS,"Production",XXX,"LO");
           Production.UP(RMS,XXX)   =    Bounds(RMS,"Production",XXX,"UP");
```

The process is repeated, which means that the cross prices and quantity effects are updated in an iterative way.