Technical Documentation

Franz Hubert & Ekaterina Orlova & Onur Cobanli
Humboldt-Universität zu Berlin

Gas Model Documentation

We give a documentation of the gas sector model and related calculations. Together with the data-files and the codes, available from "http://www.ms-hns.de/research_gas", it should help the reader to check and replicate the results of the following papers:

- Franz Hubert & Onur Cobanli: Pipeline Power [pipe1]
- Franz Hubert & Ekaterina Orlova: Competition or Countervailing Power for the European Gas Market [reg1]
- Franz Hubert & Ekaterina Orlova: Network Access and Market Power [reg2]
- Onur Cobanli: Central Asian Gas in Eurasian Power Game [pipe2]

While the papers differ in their economic focus and in many technical details they all use variants of a model of the European gas-network and notions from cooperative game theory to analyze the power structure in the Eurasian gas trade. Each paper starts from a broad description of the network: its geographical scope, major players etc. In this respect, we have four basic variants (pipe1, pipe2, reg1, reg2); one for each paper.

All four papers analyze how the bargaining power of the players is affected by various changes such as a new pipeline, liberalization of pipeline access, a merger, increase of demand etc. Each of these scenarios correspond to a distinct cooperative game, for which we have a unique identifier, the *variant-name* or *VN*.[1] These games are formulated and solved using software written in Mathematica and Matlab, which are described in the following.

[1] Typically the variant name consists of several parts referring to specific settings such as geographic scope, set of players etc. These variant names are used as identifiers to build file names for the results, e.g. the Shapley values are saved in a file `VN-Shapley`).

A cooperative game is characterized by a set of players N and a value function v. For each possible subset of players $S \in N$ (also called coalition), $v(S)$ gives the maximal joint payoff which the coalition S can achieve on its own. In other words, v is the result of a number of related optimization problems. These optimization problems share a common structure, because they are derived from the same broad network model, but they differ in the sense that smaller coalitions have only access to parts of the whole network.

So the analysis proceeds in four steps.

1. We characterize the general network optimization problem of the cooperative game. For each variant we specify the instruments and parameters of the network optimization problem. These parameters include the specification of access rights, so that we can derive the embedded sub-network optimization problems of smaller coalitions. We refer to this representation of the game *VN-parameters*.

2. We calculate the numerical values of the value function by solving all sub-network optimization problems for a particular variant/game. We call this representation *VN-values*. Since we look at a large set of coalitions, this step is computationally the most demanding one.

3. Using the numerical value function, we calculate for each variant various solutions for cooperative games, such as Shapley value, nucleolus, core. We refer to the solutions as *VN-Shapley*, *VN-nucleolus*, etc.

4. Finally, we compare the solutions of different variants to assess the impact of pipeline investment, regulatory changes etc. and build the tables in the papers.

The code which defines the parameters of the network optimization problem, calculates the value function and then the Shapley value is written in Mathematica (step 1-3). The code which calculates the Nucleolus and the minimum and maximum values of the core is written Matlab (step 3). Further evaluations of the results are again written in Mathematica (step 4). In the next sections, we give a brief overview on the main programming tools for each of these steps.

We save results to a number of files in plain text format. The following files contain results. VN stands for variant-name.

Files containing results

name	content
description of network optimization model	
VN-parameters-Mathematica	The parameters for the optimization problem in the format required for `calculateValueOneCoalition[]` .
VN-parameters-General	Same as above in a simplified format for use with other optimization software.
value function	
VN-value-Full	Explicit list of coalitions and values, as well as any errors reported from the calculation (very large).
VN-value-Mathematica	Values in a compressed format, suitable for Mathematica's `Subsets[]` function.
VN-value.nuc	Values in a compressed format, suitable for calculating the Nucleolus using Matlab code of Johannes Reijnierse.
cooperative solutions	
VN-Shapley	the Shapley Value
VN-Nucleolus	the Nucleolus
VN-MinCore	the minimal values players receive in the core
VN-MaxCore	the maximal values players receive in the core
technical files	
VN-nucl.dat	log and results from calculating the nucleolus.
VN-MinCore.dat	log and results from calculating the minimal core
VN-MaxCore.dat	log and results from calculating the maximal core

Directories

name	content
\EAGas-model	Mathematica notebooks and corresponding packages for setting up the network optimization problem, calculating the value function, solving for the Shapley value.
\games+tools	Mathematica and Matlab code to convert files, to calculate minimal and maximal values in the core.
\nucleolus HR	Matlab code provides by Hans Reijnierse for calculating the nuleolus.
\pipe1	special code and results related to Hubert & Cobanli: Pipeline Power
\pipe2	special code and results related to Cobanli: Central Asian Gas
\reg1	special code and results related to Hubert & Orlova: Competition or Countervailing Power
\reg2	special code and results related to Hubert & Orlova: Network Access and Market Power

1 General Network Optimization Problem

All papers share a common data base from which the calibrations and definitions of their network optimization problems are obtained using two Mathematica notebooks, a common one `Gas Parameters` and an additional one which is individual for each paper. There are also packages to visualize the data base and the parameter settings.

All code of this section is written in Mathematica. The general network optimization problem is saved in files named `VN-parameters-*`, where the * stands for different formats.

1.1 data & calibration

definition of data.

The data are defined in **`Gas Data Base`** using a similar format as the data provided by Mathematica. All data, which are needed for the model specification and the displays (tables and maps) are assigned to global variables by loading the Mathematica package `Gas Data Base` .
requires: nothing

visualization of data.

`Gas Data Visu` defines functions for the display of data.
requires: `Gas Data Base` , `FH Tools`

Data Overview

package	function	needs
`Gas Data Base`	assignes data to global variables	*nothing*
`Gas Data Visu`	defines functions for display of data	`Gas Data Base` `FH Tools`
file output:	*none*	

1.2 set-up for network optimization

The topology of the network is defined by a set of nodes R and a set of directed links L (the geographical scope). Each link $\{i, j\} \in L$ connects two nodes, which might be

R_P production nodes, R_C consumption nodes, or R_T transit nodes.[2] For each link we have (piecewise linear) cost reflecting transportation and/or production cost.

The game is defined by a set of players N and a value function v, mapping the set of subsets of N into real numbers. A coalition $S \subseteq N$ has access to $L(S) \subseteq L$ (the access regime). The value of a coalition S is obtained by maximizing the joint surplus (gross surplus from consumption s minus cost of transportation and production T) using the gas-flows x_{ij} in the pipelines which are accessible:

$$v(S) = \max_{\{x_{ij} | \{i,j\} \in L(S)\}} \left\{ \sum_{\{i,j\} \in L(S),\ j \in R_C} s_j(x_{ij}) - \sum_{\{i,j\} \in L(S)} T_{ij}(x_{ij}) \right\} \tag{1}$$

subject to

$$\begin{aligned} x_{ij} &\geq 0, & \forall\, i \in R_P \text{ or } j \in R_C & \quad \text{(non-negativity)} \\ \textstyle\sum_i x_{it} &= \textstyle\sum_j x_{tj}, & \forall\, t \in R_T(S) & \quad \text{(balancing)} \\ |x_{ij}| &\leq k_{ij}, & \forall\, \{i,j\} \in L(S) & \quad \text{(capacity constraints)} \end{aligned}$$

The capacity constraint is dropped when we allow for investment. In this case T also accounts for investment cost.

To keep the network optimization problem simple, we assume a linear demand (quadratic surplus function) and piece-wise linear cost functions.

definition of functions and some variants.

By loading **`Gas Parameters`** we define routines, which specify the functions and parameters of the optimization problem using the data provided by `Gas Data Base` .

The complete specification of the general network optimization problem (all the technical and demand parameters as well as the access rights) are assigned to global variables by calling: `reSetParTo[`*`parameter-list`*`]`, which in turn calls: `setGeoScope[]`, `setPlayers[]`, `setPipeAccess[]`, `setLinkParameter[]`, `setDemandParameter[]`. These routines define the geographical scope of the network, the players, the access regime, parameters for the individual links and for demand. Different (but not all) versions of these settings can be combined. `selectVarList` allows for an interactive selection of predefined arguments for the subroutines.

[2] Production and consumption nodes are always linked to a transit node.

feasiblePipes[*coalition*] returns the links to which a coalition has access.

In Gas Parameters we provide only the base variants, used in the different papers. To obtain the specific definitions for a paper, an additional file has to be loaded; e.g. Gas_Parameters pipe1.m or Gas_Parameters reg2.m. These define a unique *variant-name* (*VN*) for each network optimization (game), which will be part of the names of files for storing results etc. We also define setVar[*VN*] to return the arguments for reSetParTo[] .

parametersToFile[*VN*, "Mathematica"] saves the parameter settings of a game to a file with a name VN-parameters-Mathematica from which the settings can be recovered using fileToParameters[*VN*, "Mathematica"]. When writing "Mathematica" can be replaced by "General" to obtain a more compact format.

Gas Parameters and Gas Parameters *.m require Gas Data Base .

visualization of parameter settings.

Gas Parameters Visu defines functions for the display of the parameter setting, once they have been assigned by calling reSetParTo[]. There are tables and maps, some of them interactive. Various functions are collected in the commands: showMainParCurrent, showAllParCurrent, which display most of the settings.

requires: Gas Data Base , Gas Data Visu , FH Tools , Gas Parameters .

Starting from defined games.

Gas Parameters min collects the those routines which are needed if the parameter settings are already saved to files VN-parameters-Mathematica. If loaded there is no need to load other notebooks.

example

The Mathematica notebook workspace parameters.nb illustrates these steps.

Parameters Overview

package	function	needs
`Gas Parameters`	defines functions for the assignment, saving and recovering of parameters as well as parameters for some basic variants → `reSetParTo[]` → `parameterToFile[]` → `fileToParameters[]`	`Gas Data Base`
`Gas Parameters *.m`	defines additional functions and all the variant names for the individual papers (*: pipe1, pipe2, reg1, reg2)	`Gas Data Base` `Gas Parameters`
`Gas Parameters min`	collects functions needed for recovering the parameters-settings from the file and starting the optimization	*nothing*

`Gas Parameters Visu`	defines functions for display of parameters after they have been assigned using `reSetParTo[]` .
	→ `showMainParCurrent`
	→ `showAllParCurrent`
		`Gas Data Base`
		`Gas Data Visu`
		`FH Tools`
		`Gas Parameters`

file output:	`VN-parameters-Mathematica` `VN-parameters-General`	
for illustration:	**`workspace parameters.nb`**	

→ : main functions defined in the package; VN : variant name

2 value function

Given our assumption on functional forms, we obtain the value function by maximizing surplus (quadratic) minus cost (piece-wise linear) subject to balancing constraints for transit nodes and non-negativity constraints for production and consumption links.

2.1 the network optimization

By loading `Gas Prog` and `Gas ProgLP` we define the functions used for solving the network optimization problem. `calculateValueOneCoalition[]` establishes the sub-network, which is accessible for a given coalition of players and calls `LinProg[]` from `Gas ProgLP` to calculate the payoff (value).

The general optimization routines coming with Mathematica turned out to be too slow. To speed up the process `LinProg[]` approximates the quadratic surplus functions by piece-wise linear functions and uses ”LinearProgramming” to solve the resulting linear optimization problem.

2.2 visualization of the result

`Gas Prog Visu` defines `displayResChartLP[]`, and `displayResTableLP[]` for the display of the optimal network usage using the output created by `calculateValueOneCoalition[]` . It needs the full parameter definitions from `reSetParTo[]` and requires `Gas Data Base` , `Gas Data Visu` , `FH Tools` , `Gas Parameters` , `Gas Prog` , `Gas ProgLP` .

Network Optimization Overview

package	function	needs
Gas Prog	finds accessible network for a given coalition of players and calls LinProg[] from Gas ProgLP to calculate the payoff (value). → calculateValueOneCoalition[]	Gas Parameters min Gas ProgLP
Gas ProgLP	creates a linear programming problem to calculate the optimal network usage for a given network configuration. → LinProg[]	Gas Parameters min
Gas Prog Visu	display of the optimal network usage using output created by calculateValueOneCoalition[] . Needs parameters from reSetParTo[] → displayResChartLP[] → displayResTableLP[]	Gas Data Base Gas Data Visu FH Tools Gas Parameters Gas Prog Gas ProgLP
file output:	None	
for illustration:	**workspace program.nb**	

→ : main functions defined in the package

2.3 calculating the value function (and the Shapley value)

Gas ValFuncShap defines functions for the calculation of the value function. Using the unique variant-name VN we recover the parameters from the associated file VN-parameters-Mathematica. Then we calculate the value of all coalitions (repeatedly calling calculateValueOneCoalition[]). Depending on the number of players, this step may take a long time. The results are saved in two formats. VN-values-Full has the value, the coalition and possible error-messages and is very large. VN-values-Mathematica has only the numerical values ordered as the coalitions are ordered by Mathematica's Subsets[] command i.e. $\{\}, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}$.

As we have the value function already, it is convenient to invoke FH Shapley and write the Shapley value into VN-Shapley.

For some changes, i.e. if two players merge, it is not necessary to run all optimization problems again. The new value function can be obtained from the old one by re-matching values with coalitions. Suppose we start with a game $\{N, v\}$ and let

players a and b merge. We define the new game as $\{N, w\}$ by making a a 'proxy' player and b a dummy player. The new value function w is obtained from v as

$$w(S) = \begin{cases} v(S \cup b) & \text{if} \quad a \in S, \quad b \notin S \\ v(S \setminus b) & \text{if} \quad a \notin S, \quad b \in S \\ v(S) & \text{else.} \end{cases}$$

In these cases we save only the original value function.

Calculating Value Function & Shapley Value

package	does	loads

`Gas ValFuncShap`	provides operations for the calculation of the value function (which may take a long time) and all Shapley Values.
	→ `calcValFuncShap[]`
	→ `calcShapleyValue[]`
	→ `assignValueFunction[]`
		`Gas Parameters min`
		`Gas Prog`
		`Gas ProgLP`
		`FH Shapley`

file output:	`VN-value-Full`
	`VN-value-Mathematica`
	`VN-Shapley`	

for illustration:	**`workspace ValFuncShap.nb`**	

→ : main functions defined in the package ; VN : variant name

3 solving the game

We consider several solutions for the games defined by the different variants: Shapley value, nucleolus, and the core, which we characterize by the minimal and maximal values which a player can achieve. The starting point is always the set of players and the value function as specified in `VN-value-Mathematica`.

We express the solutions as absolute values and as relative values (in per cent of the value of the Grand coalition). In addition we report the player's value when he is alone and we give the solutions (absolute and relative) for the zero normalized game.

For the calculation of nucleolus and minimal and maximal values in the core, we use Matlab code.

3.1 Shapley Value

We calculate the Shapley value intermediately after calculating the value function (see section 2.3). The function is defined in `FH Shapley` . In addition we have some tools to rearrange and aggregate players once the Shapley values are calculated.

Calculating the Shapley Value

package	does	loads
`FH Shapley`	functions to calculate the Shapley Value for a set of players and a value function. → `shapleyValue[]` → `allShapleyValues[]`	*nothing*
`FH Shapley tools`	functions to rearrange and aggregate players in the output of `FH Shapley` (or other solutions).	*nothing*
for illustration:	**`workspace ValFuncShap.nb`**	

→ : main functions defined in the package

3.2 Nucleolus

To calculate the nucleolus we use Matlab code provided by Hans Reijnierse. It implements an algorithm described in Potters, J. A.; Reijnierse, J. H.; Ansing, M. (1996): Computing the Nucleolus by Solving a Prolonged Simplex Algorithm, *Mathematics of Operation Research*, 21(3) 757-68. The algorithm in turn is based on the characterization of the nucleolus as the lexicographical center of the game developed in Maschler, M.; Peleg, B. Shapley, L. S. (1979): Geometric Properties of the Kernel, Nucleolus, and Related Solution Concepts, *Mathematics of Operation Research*, 4(4), 303-38.

We first convert `VN-value-Full` into `VN-value.nuc`. This file is used by Matlab program `calcNucleolus`, which invokes Reijnierse's command "nucleolus". The log and results are written into `VN-nucl.dat`. We switch back to Mathematica code to further process `VN-nucl.dat`, extracting the nucleolus and those coalitions and their excesses which determine the solution.

Calculating the Nucleolus

package	does	loads
`convert-nucleolus`	Preparing input for Matlab, reading Matlab output and writing it to files. Convert `VN-value-Full` into `VN-value.nuc`, extract results from `VN-nucl.dat` (Matlab output) into Mathematica, and prepare the input for the tables. → `writeMatlabInputFunc[]` → `vectorNucleolusList[]` → `vectorPlayersNucleolus[]` → `writeToFileAllValues[]`	*nothing*
`calcNucleolus` (Matlab)	Reads `VN-value.nuc`, calculates the nucleolus, writes `VN-nucl.dat`.	`nucleolus`
`nucleolus` (Matlab)	package to calculate the nucleolus written by on Potters, Reijnierse, Ansing (1996).	

file output:	`VN-value.nuc`
	`VN-nucl.dat` (from Matlab)
	`VN-Nucleolus`	

for illustration:	**`workspace nucleolus.nb`**	

→ : main functions defined in the package ; VN : variant name

3.3 Core

As the core is characterized by a large number of inequalities, we restrict attention to the extreme values which a player can obtain in the core. For each player we find the minimal and the maximal value in core.

As with the nucleolus we use Matlab to compute the values.

Characterizing the Core

package	does	loads
convert-core	collects routines for creating matrices and writing "*.csv" files for optimization in Matlab. We also define functions to extract the values from Matlab output files, to compare these values with the nucleolus and the Shapley value, to prepare the input for the table. → writeMatricesVariantsMatlab[] → readMatlabCore[] → ShapleyMinNuclMax[] → writeConceptsToFile[]	convert-nucleolus
calcMaxMinCore (Matlab)	Reads variants.csv, matrices.csv and VN_matrixname.csv, calculates the minimum and the maximum for each player, writes VN-MinCore.dat and VN-MaxCore.dat.	*nothing*
file output:	VN_matrixname.csv variants.csv matrices.csv VN-MinCore.dat (from Matlab) VN-MaxCore.dat (from Matlab) VN-MinCore VN-MaxCore VN-Concepts	
for illustration:	**workspace core.nb**	