Intrinsic valuation
of natural gas storage facilities

April 9, 2018

Abstract

This assignment is based on suggestion 6 by Prof. Tony Ware. It discusses a MILP algorithm to compute the intrinsic value of a gas storage facility given prices for monthly futures contracts and physical storage constraints. The MILP formulation uses Special Ordered Sets of type 2 to deal with continuous piecewise linear limits in the monthly injection and withdrawal schedule. The algorithm is implemented using the student version of the CPLEX for MATLAB Toolbox by IBM.

1 Introduction

Seasonality and demand spikes in the natural gas market necessitate the service of gas storage facilities. Traditionally the need for natural gas is dominated by residential heating and shows a prominent seasonal pattern with high demand during winter and low demand during summer. Gas storage facilities act as a buffer between the variable demand and the more constant rate of natural gas production. Gas is typically injected during spring and summer when supply exceeds demand and withdrawn during winter to serve heating demand. Additionally specific peak load facilities are designed to meet short-term demand spikes caused for instance by unexpected temperature extremes.

While the operator of a storage facility usually is subject to regulatory constraints he can also manage the facility with the goal of making profit in the gas market. Storage capacity allows the operator to store gas and vary inventory positions in response to changing market conditions, in particular to inject gas when prices are low and to withdraw it upon rising prices. The monetary value that can be extracted from a storage facility consists of two components: intrinsic and extrinsic value.

The intrinsic value is the maximum profit that can be locked in by exploitation of the predictable seasonal shape of the gas price futures curve. As futures prices for summer months are below the prices for winter months, one goes long in the summer months and short in the winter months. The goal is to find the optimal injection and withdrawal schedule given the current futures prices and taking into account all physical storage constraints. This approach fixes the strategy at the beginning of the valuation period. The resulting intrinsic value is certain but does not take advantage of the operational flexibility inherent to a gas storage facility.

The extrinsic value comes from this very flexibility to quickly respond to movements in gas prices as they unfold. Short-term prices are highly volatile and additional value can be generated by dynamically optimizing injection and withdrawal operations based on the current spot price. Storage offers injection capacity or extra gas supply to profit from unexpected yet favourable price movements. Consequently, models to capture the full extrinsic value of a storage facility are usually spot price models.

In this assignment we develop a MILP algorithm to compute the intrinsic value of a natural gas storage facility based on a given futures price curve. Section 2 elaborates on the physical storage constraints and introduces all relevant problem parameters. We allow piecewise constant ratchets on the daily injection and withdrawal rates which in the monthly view can be approximated by continuous piecewise linear limits. In section 3 we give two formulations of the problem in the framework of mixed integer linear programming (MILP) which enables us to tackle the problem algorithmically via IBM's CPLEX solver. The second formulation use so-called special ordered sets of type 2 to handle the piecewise linear constraints more efficiently than the first formulations which needs additional decision variables. Section 4 demonstrates the effectiveness of the MILP algorithm on three storage example problems. Details on the numerical results can be found in Appendix A together with the Matlab implementation in Appendix B.

2 Problem description

We consider a monthly futures market, i.e. the available futures mature at the beginning of a month and the delivery period ranges over the entire month. For ease of presentation we assume that each month consists of 30 days. Futures contracts can only be purchased in multiples of D (in GJ). The gas storage facility shall be operated over a period of N consecutive months with starting dates T_i for $i = 1, \ldots, N$. Let $F_i(t_0)$ denote the respective futures mid prices (in USD/GJ) as they are observed in the market at time $t_0 \leq T_1$. We assume a constant risk-free interest rate r and a constant bid-ask spread s so that we obtain discounted bid and ask prices

$$b_i(t_0) = e^{-r(T_i - t_0)}(F_i(t_0) - s/2), \tag{1}$$

$$a_i(t_0) = e^{-r(T_i - t_0)}(F_i(t_0) + s/2). \tag{2}$$

The natural gas storage facility has the following physical constraints:

1. **Minimum inventory and total capacity**

At each time t the currently stored gas volume V_t (in GJ) must satisfy $V_{\min} \leq V_t \leq V_{\max}$ with $V_{\min}$ being the minimum inventory and $V_{\max}$ the total capacity of the facility.

2. **Initial and final inventory**

   The facility starts out with a certain volume $V_{\text{start}} = V_{T_1}$. At the end of the operational period the facility needs to hold $V_{\text{end}} = V_{T_{N+1}}$. Alternatively one imposes no terminal condition and any gas left in storage at the end of N months is lost.

3. **Injection-withdrawal fuel costs**

   Both injection and withdrawal are powered by gas and consume a certain percentage $\alpha \in [0, 1]$ of any delivered gas.

4. **Ratchets on injection and withdrawal rates**

   The daily limits $I(V)$ resp. $W(V)$ for injection resp. withdrawal (in GJ/day) are piecewise constant functions of the current gas volume V in the storage.

The goal is to compute the (discounted) intrinsic value of the facility at time $t = t_0$ which is equivalent to finding the optimal futures positions that satisfy all physical storage constraints and yield maximum profit.

We emphasize that in following the intrinsic valuation methodology one performs the optimization only once at the beginning of the operational period. The calculated futures positions are then held for the entire operational period irrespective of any movements in the price curve. This "static" intrinsic strategy may be extended to the "rolling" intrinsic strategy, cf. Gray and Khandelwal [2]. Here the storage holder extracts the initial intrinsic value as above and then dynamically rebalances his futures positions in response to favourable changes in the futures curve. In the following we will only discuss the static variant.

3 MILP formulation

If we assume for a moment constant limits $I, W \geq 0$ on the daily injection and withdrawal rates the problem of determining the optimal futures positions becomes a straightforward integer linear program, cf. Eydeland and Krzysztof [1, p.356]. Let us denote by $x_1, \ldots, x_N \in \mathbb{N}_0$ the number of long positions in gas futures per month and by $y_1, \ldots, y_N \in \mathbb{N}_0$ the number of short positions. Write $c = 1 - \alpha$ for the gas loss factor due to fuel costs. The (ILP) formulation then is

$$\max_{x,y} \quad D\left(-\sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N} b_i y_i\right) \tag{3}$$

$$\text{s.t.} \quad V_{\min} \leq V_{\text{start}} + D\left(\sum_{i=1}^{j} c x_i - \sum_{i=1}^{j} c^{-1} y_i\right) \leq V_{\max} \qquad j = 0, 1, \ldots, N \tag{4}$$

$$V_{\text{end}} = V_{\text{start}} + D\left(\sum_{i=1}^{N} c x_i - \sum_{i=1}^{N} c^{-1} y_i\right) \tag{5}$$

$$0 \leq D x_i \leq I \qquad i = 1, \ldots, N \tag{6}$$

$$0 \leq c^{-1} D y_i \leq W \qquad i = 1, \ldots, N \tag{7}$$

$$x_i, y_i \in \mathbb{Z} \qquad i = 1, \ldots, N \tag{8}$$

Equation 5 is optional depending on whether one imposes a terminal condition on the storage level or not. Note that if we buy x_i futures contracts for the ith month we will only have cDx_i GJ of gas in the storage at the end of the month due to the fuel costs. Vice versa, if we want to sell y_i futures contracts, we need to withdraw $c^{-1}Dy_i$ GJ of gas to fulfil the delivery contract.

Next we want to allow piecewise constant ratchets $I(V), W(V)$ on the daily injection and withdrawal limits. Figure 1 shows an exemplary plot of such daily limits together with the resulting monthly limits. To obtain the monthly limits depending on the gas storage level at month-beginning, the individual daily limits are summed up according to the updated storage level at the beginning of each day and under consideration of the gas loss factor, see Matlab Listing 4.

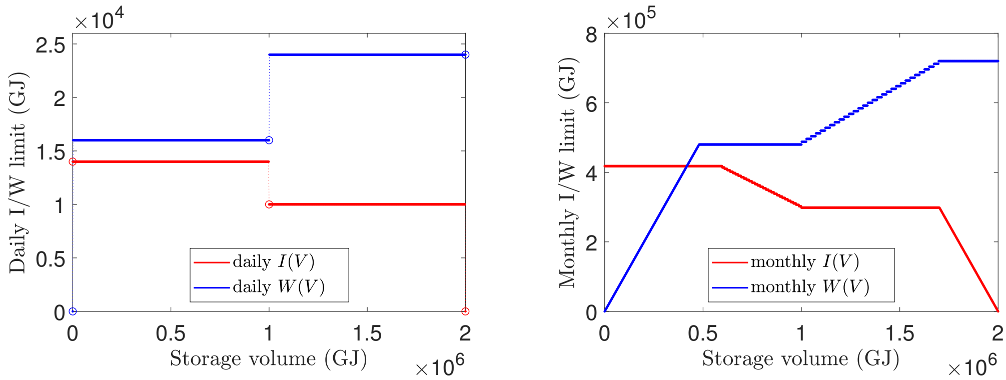

Figure 1. Daily and resulting monthly limits $I(V), W(V)$ for injection resp. withdrawal depending on the gas storage level V.

The resulting monthly limits are piecewise linear and constant functions in V with discontinuities. To reduce the model's complexity we approximate them by continuous piecewise linear functions. A continuous piecewise linear function $g(v)$ with $v_{\min} \leq v \leq v_{\max}$ can be represented by m nodes $v_{\min} = v_1, v_2, \ldots, v_m = v_{\max}$ and their function values $g(v_j)$ such that the graph of g consists of $m-1$ connected line segments. The points $(v_j, g(v_j))$ are called the support points of g. We can hence assume that the approximated monthly injection and withdrawal limits are given by their support points

$$\begin{aligned} &(V_{j,I}\ \ , I_j\ \ ), \quad j = 1, \ldots, m_I \\ &(V_{j,W}, W_j), \quad j = 1, \ldots, m_W \end{aligned}$$

with $V_{1,I} = V_{1,W} = V_{\min}$ and $V_{m_I,I} = V_{m_W,W} = V_{\max}$, see Figure 2.

Continuous piecewise linear functions can be described in the framework of mixed integer linear programming (MILP), cf. [5, p.5]. Additional decision variables with a set of combinatorial constraints need to be introduced in order to represent such functions via their support points. By using convex combinations of two neighbouring nodes one can compute the function values for all points lying in-between those nodes. We exemplarily demonstrate this for the approximated monthly injection limits, the withdrawal limits can be treated in the same way.

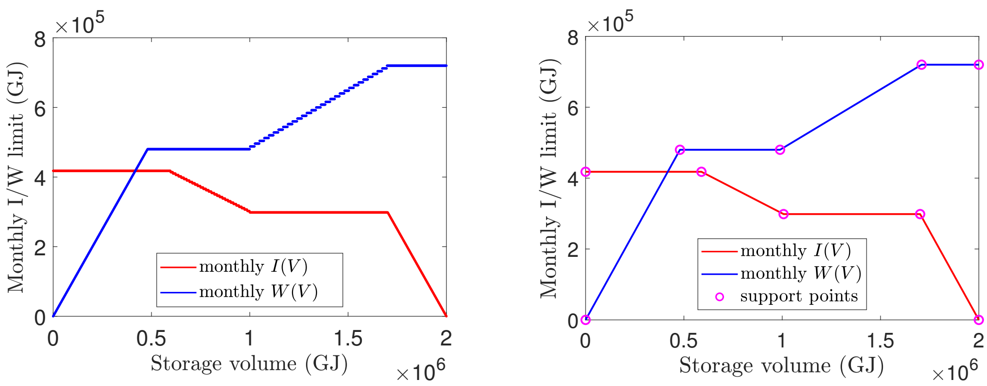

Figure 2. Piecewise linear approximation (on the right) of the monthly injection resp. withdrawal limits (on the left, from Figure 1) with highlighted support points. Calculation by Matlab Listing 5.

Constraints for approximated monthly injection limits:

$$0 \leq Dx_{i+1} \leq \sum_{k=1}^{m_I} \lambda_{k,I}^i I_k \tag{9}$$

$$\sum_{k=1}^{m_I} \lambda_{k,I}^i V_{k,I} = V_{\text{start}} + D\left(\sum_{k=1}^{i} cx_k - \sum_{k=1}^{i} c^{-1} y_k\right) \tag{10}$$

$$\sum_{k=1}^{m_I} \lambda_{k,I}^i = 1 \tag{11}$$

$$\sum_{k=1}^{m_I-1} z_{k,I}^i = 1 \tag{12}$$

$$0 \leq \lambda_{j,I}^i \leq z_{j-1,I}^i + z_{j,I}^i \tag{13}$$

$$z_{0,I}^i = z_{m_I,I}^i = 0 \tag{14}$$

$$z_{j,I}^i \in \{0,1\} \tag{15}$$

where we introduced new decision variables $\lambda_{j,I}^i, z_{j,I}^i$ and the free indices i, j range over $i = 0, \ldots, N-1$ and $j = 1, \ldots, m_I$. The binary variables $z_{j,I}^i$ ensure that at most two adjacent λ-variables $\lambda_{j,I}^i$ (per index i) can be nonzero. These correspond to the two end nodes $V_{j,I}$ of the interval in which the current storage level

$$V_{T_{i+1}} = V_{\text{start}} + D\left(\sum_{k=1}^{i} cx_k - \sum_{k=1}^{i} c^{-1} y_k\right)$$

lies.

In summary when dealing with ratchets Equations 9 to 15 replace Equation 6 and the analogous equations for withdrawal ratchets would replace Equation 7. Since the monthly injection resp. withdrawal limits are additionally capped at $V_{\max}$ resp. floored at $V_{\min}$ they also replace Equation 4. In total we have to introduce $2N(m_I + m_W - 1)$ new variables.

A more efficient formulation involves the use of so called Special Ordered Sets of Type 2 (SOS2). In the context of linear programming a SOS2 is a set of (integer or continuous) decision variables of which at most two can be nonzero. If two variables are nonzero, they must be adjacent in the set. One immediately notices that the SOS2 conditions make the binary z-variables in the equations above obsolete. Many MILP solvers, e.g. the CPLEX for MATLAB Toolbox by IBM used in this assignment, take advantage of SOS2 sets by using special branching strategies which can significantly improve performance, cf. [3, p.321ff]. The final (MILP) formulation then is

$$
\begin{aligned}
\max_{x,y} \quad & D\left(-\sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N} b_i y_i\right) && && (16)\\
\text{s.t.} \quad & V_{\text{end}} = V_{\text{start}} + D\left(\sum_{i=1}^{N} c x_i - \sum_{i=1}^{N} c^{-1} y_i\right) && && (17)\\
& \sum_{k=1}^{m_I} \lambda_{k,I}^i I_k \quad \geq \quad D x_{i+1} && i = 0,\ldots,N-1 && (18)\\
& \sum_{k=1}^{m_W} \lambda_{k,W}^i W_k \ \geq c^{-1} D y_{i+1} && i = 0,\ldots,N-1 && (19)\\
& \sum_{k=1}^{m_I} \lambda_{k,I}^i V_{k,I} \ = V_{\text{start}} + D\left(\sum_{k=1}^{i} c x_k - \sum_{k=1}^{i} c^{-1} y_k\right) && i = 0,\ldots,N-1 && (20)\\
& \sum_{k=1}^{m_W} \lambda_{k,W}^i V_{k,W} = V_{\text{start}} + D\left(\sum_{k=1}^{i} c x_k - \sum_{k=1}^{i} c^{-1} y_k\right) && i = 0,\ldots,N-1 && (21)\\
& \sum_{k=1}^{m_I} \lambda_{k,I}^i \ = 1 && i = 0,\ldots,N-1 && (22)\\
& \sum_{k=1}^{m_W} \lambda_{k,W}^i = 1 && i = 0,\ldots,N-1 && (23)\\
& \{\lambda_{1,I}^i,\ldots,\lambda_{m_I,I}^i\} \quad \text{is SOS2} && i = 0,\ldots,N-1 && (24)\\
& \{\lambda_{1,W}^i,\ldots,\lambda_{m_W,W}^i\} \text{ is SOS2} && i = 0,\ldots,N-1 && (25)\\
& \lambda_{j,I}^i \ \geq 0 && j = 1,\ldots,m_I\ ,\ i = 0,\ldots,N-1 && (26)\\
& \lambda_{j,W}^i \geq 0 && j = 1,\ldots,m_W,\ i = 0,\ldots,N-1 && (27)\\
& x_i, y_i \geq 0 && i = 1,\ldots,N && (28)\\
& x_i, y_i \in \mathbb{Z} && i = 1,\ldots,N && (29)
\end{aligned}
$$

The Matlab procedure that creates above MILP structure for the CPLEX solver can be found as Listing 12 in the appendix.

4 Numerical results

In this section we present numerical results for three exemplary gas storage problems, one of them being the problem given in assignment suggestion 6 by Prof. Tony Ware. In our Matlab implementation the problem parameters can be loaded with Listing 2 and Listing 3. In all three problems we used the futures curve given in the assignment suggestion (left plot in Figure 3), the other parameters are listed per problem in Tables 1, 2 and 3 on the following pages. We briefly note that in the course of the assignment we switched from Matlab's internal MILP solver to IBM's CPLEX for MATLAB Toolbox since it showed superior solving capabilities. The main script to trigger the intrinsic storage valuation is given by Listing 1 in the appendix.

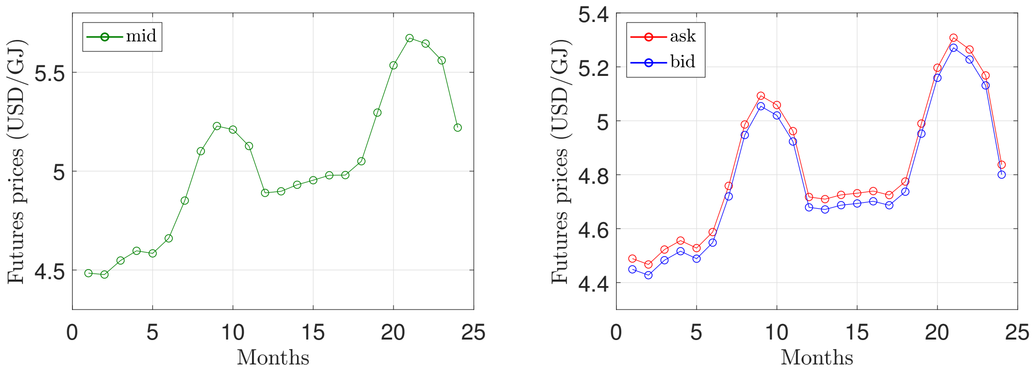

Figure 3. On the left futures mid prices in USD/GJ as given in assignment suggestion. On the right discounted bid-ask prices assuming $r = 4\%$, $T_1 - t_0 = 30$ days, $s = 0.04$ USD/GJ, see Equation 1 and Equation 2.

4.1 Problem 1: Constant daily limits

Parameter	$V_{\min}$	$V_{\max}$	α	D	$T_1 - t_0$	V_{start}	V_{end}	r	s
Value	0	2e6	0.5	2500	30	0	-	0	0
Unit	GJ	GJ	%	GJ	days	GJ	GJ	%	\$/GJ

Table 1. Problem 1 parameters, see section 2. The futures mid prices are taken from Figure 3 and the daily injection/withdrawal limits are given in Figure 4.

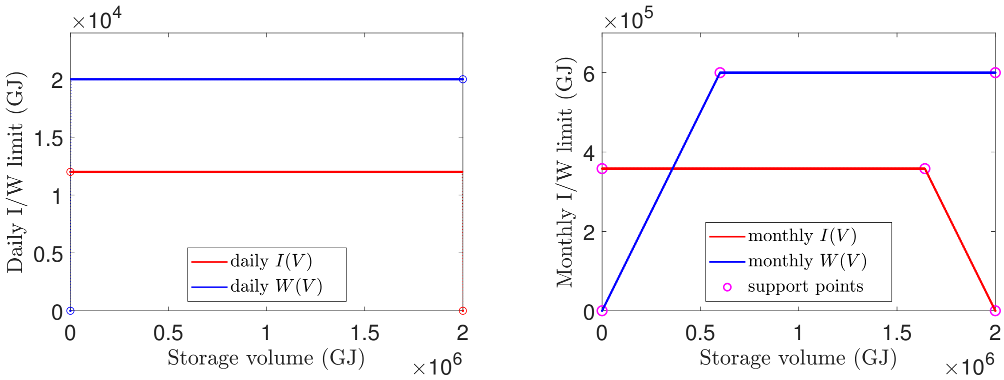

Figure 4. Given daily injection/withdrawal limits as functions of the gas storage level and resulting monthly limits including support points of approximation.

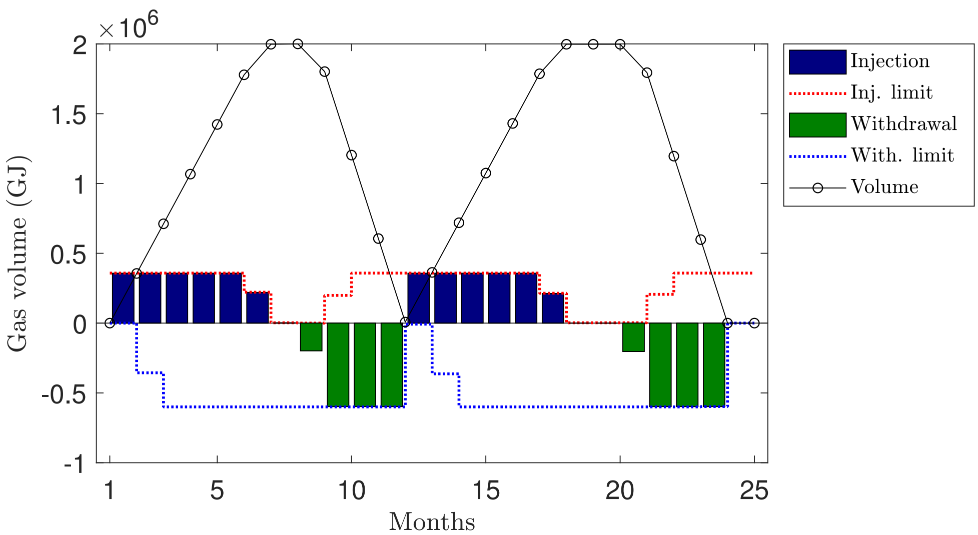

Figure 5. The optimal injection and withdrawal schedule for Problem 1. After providing an initial solution the problem was optimally solved in 7s on a 3y old standard laptop and the intrinsic storage value is 2411527.5 USD. For details see Table 4 in the appendix.

4.2 Problem 2: Simple daily ratchets

Parameter	$V_{\min}$	$V_{\max}$	α	D	$T_1 - t_0$	V_{start}	V_{end}	r	s
Value	0	2e6	0.5	2500	30	0	-	0	0
Unit	GJ	GJ	%	GJ	days	GJ	GJ	%	\$/GJ

Table 2. Problem 2 parameters, see section 2. The futures mid prices are taken from Figure 3 and the daily injection/withdrawal limits are given in Figure 6.

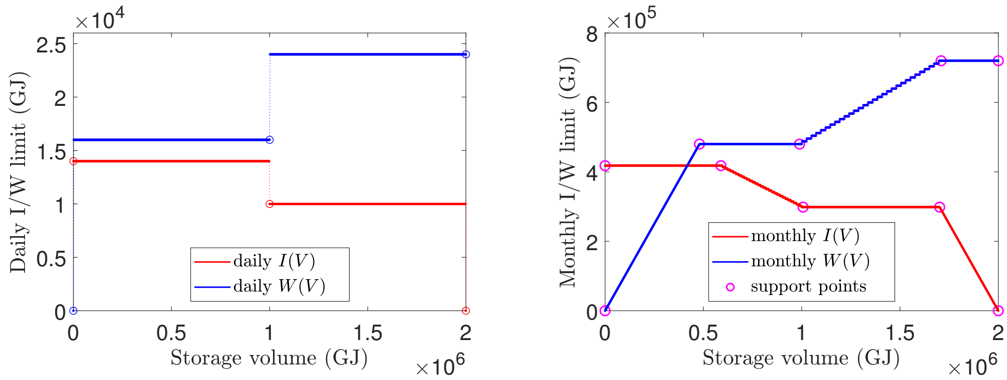

Figure 6. Given daily injection/withdrawal limits as functions of the gas storage level and resulting monthly limits including support points of approximation.

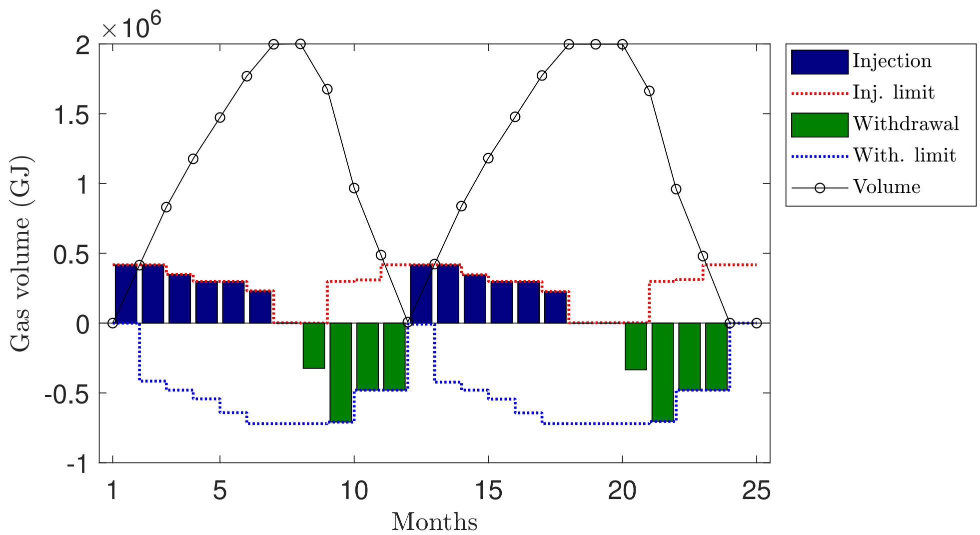

Figure 7. The optimal injection and withdrawal schedule for Problem 2. Without initial solution the problem was optimally solved in 96s on a 3y old standard laptop and the intrinsic storage value is 2428430.0 USD. For details see Table 5 in the appendix.

4.3 Problem 3: Complex daily ratchets

Parameter	V_{min}	V_{max}	α	D	$T_1 - t_0$	V_{start}	V_{end}	r	s
Value	0	2e6	1	1000	30	0	-	2	0.002
Unit	GJ	GJ	%	GJ	days	GJ	GJ	%	\$/GJ

Table 3. Problem 3 parameters, see section 2. The futures bid-ask prices are based on the mid prices from Figure 3 and the daily injection/withdrawal limits are given in Figure 8.

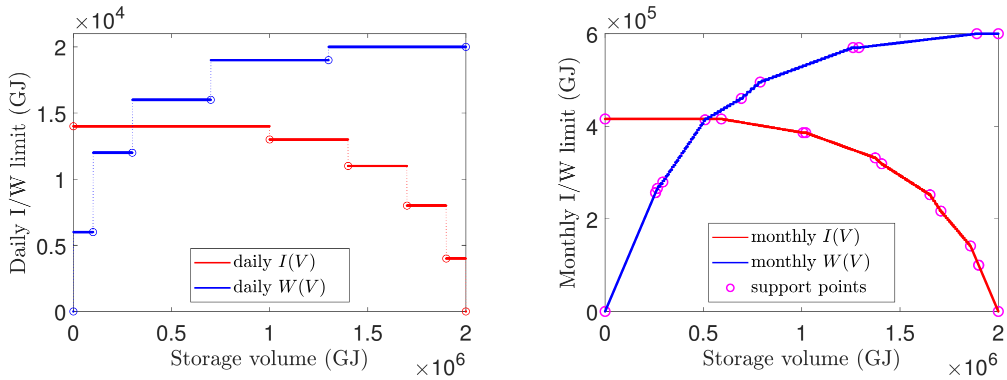

Figure 8. Given daily injection/withdrawal limits as functions of the gas storage level and resulting monthly limits including support points of approximation.

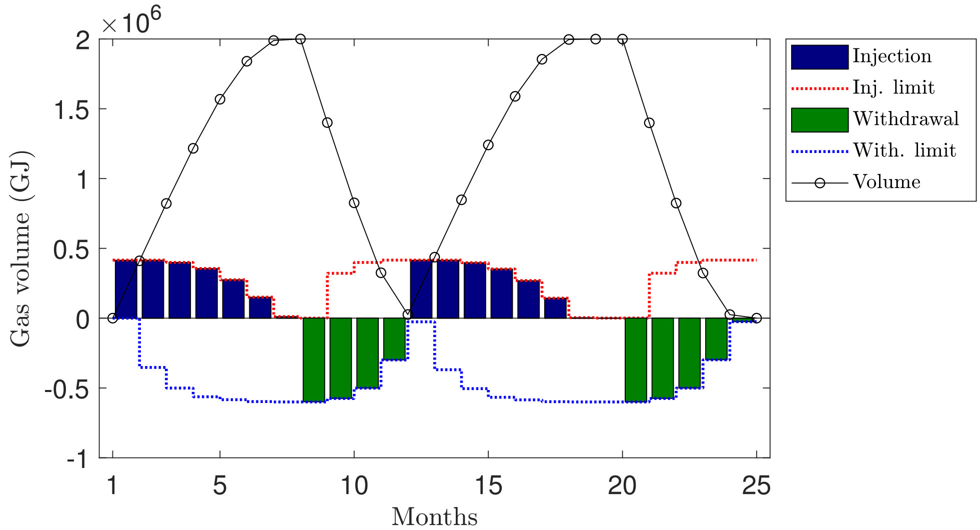

Figure 9. A feasible schedule for Problem 3. The problem couldn't be optimally solved within 30 minutes however the obtained storage value of 1910044.5 USD was reached within 10s and only has a relatve gap of 0.03% to the best known upper bound. For details see Table 6 in the appendix.

5 Conclusion

We have discussed futures based intrinsic valuation of gas storage facilities. By using special ordered sets of type 2 we provided an efficient implementation of the storage valuation problem in the framework of mixed-integer linear programming that can deal with continuous piecewise linear ratchets on the monthly injection and withdrawal limits. The Matlab interface of IBM's LP solver CPLEX (student version) allowed us to optimally solve the suggested assignment problem and variants in low computational time.

One needs to note however that storage operators who want to take full monetary advantage of their storage asset should not solely rely on static futures optimization but also use more sophisticated spot price models to extract the full extrinsic value of their facility. Nevertheless, due to its clear concept and the large availability of free and mature LP solvers, intrinsic valuation is still popular among practitioners. In the end it is always beneficial to have a robust and easy-to-implement valuation method, especially in the setting of complex ratchet constraints, that can serve as a benchmark for more sophisticated models by providing reliable lower bounds on the storage value.

References

[1] A. Eydeland and W. Krzysztof. *Energy and Power Risk Management: New Developments in Modeling, Pricing and Hedging.* Wiley Finance. Wiley, 2003.

[2] J. Gray and P. Khandelwal. Realistic natural gas storage models ii: Trading strategies. *Commodities Now*, pages 86–91, September 2004.

[3] IBM. Ibm ilog cplex optimization studiocplex users manual. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.studio.help/pdf/usrcplex.pdf`. Accessed 05-April-2018.

[4] S. Maragos. Valuation of the operational flexibility of natural gas storage reservoirs. In E. I. Ronn, editor, *Real Options and Energy Management*, chapter 14, pages 431–456. Risk Waters Group Ltd., 2002.

[5] T. Walther. A scip constraint handler for piecewise linear functions. Master's thesis, Technische Universitaet Berlin, 2014.

A Details on numerical results

A.1 Problem 1: Constant daily limits

```
Intrinsic Storage Valuation
------------------------------------------------------------------------------------------------
Problem parameters:
------------------------------------------------------------------------------------------------
Facility minimum inventory : GJ 0
Facility total capacity    : GJ 2000000
Facility I/W fuel costs    : Percent 0.5
Storage initial inventory  : GJ 0
Storage final inventory    : No condition
Futures contract size      : GJ 2500
Futures bid-ask spread     : USD/GJ 0
Time until first contract  : Days 30
Interest rate              : Percent 0
------------------------------------------------------------------------------------------------
Optimal schedule:
------------------------------------------------------------------------------------------------
```

Month	Long	Short	Bid	Ask	Capital	Inj.	Wit.	Inj.Lim.	Wit.Lim.	Volume
-					0					0
1	143	0	4.484	4.484	-1603030	357500	0	358200	0	355712
2	143	0	4.477	4.477	-3203558	357500	0	358200	355712	711425
3	143	0	4.548	4.548	-4829468	357500	0	358200	600000	1067137
4	143	0	4.597	4.597	-6472895	357500	0	358200	600000	1422850
5	143	0	4.584	4.584	-8111675	357500	0	358200	600000	1778562
6	88	0	4.660	4.660	-9136875	220000	0	221437	600000	1997462
7	1	0	4.851	4.851	-9149003	2500	0	2537	600000	1999950
8	0	79	5.101	5.101	-8141555	0	198492	50	600000	1801457
9	0	238	5.228	5.228	-5030895	0	597989	198542	600000	1203467
10	0	238	5.210	5.210	-1930945	0	597989	358200	600000	605477
11	0	238	5.127	5.127	1119620	0	597989	358200	600000	7487
12	143	0	4.890	4.890	-628555	357500	0	358200	7487	363200
13	143	0	4.898	4.898	-2379590	357500	0	358200	363200	718912
14	143	0	4.931	4.931	-4142423	357500	0	358200	600000	1074625
15	143	0	4.954	4.954	-5913477	357500	0	358200	600000	1430337
16	143	0	4.979	4.979	-7693470	357500	0	358200	600000	1786050
17	85	0	4.980	4.980	-8751720	212500	0	213949	600000	1997487
18	0	0	5.050	5.050	-8751720	0	0	2512	600000	1997487
19	0	0	5.296	5.296	-8751720	0	0	2512	600000	1997487
20	0	81	5.535	5.535	-7630883	0	203517	2512	600000	1793970
21	0	238	5.673	5.673	-4255448	0	597989	206029	600000	1195980
22	0	238	5.645	5.645	-896672	0	597989	358200	600000	597990
23	0	238	5.560	5.560	2411527	0	597989	358200	597989	0
24	0	0	5.220	5.220	2411527	0	0	358200	0	0

```
------------------------------------------------------------------------------------------------
Intrinsic storage value: USD 2411527.5
------------------------------------------------------------------------------------------------
```

Table 4. Details on optimal schedule for problem 1.

A.2 Problem 2: Simple daily ratchets

Intrinsic Storage Valuation

Problem parameters:

```
Facility minimum inventory : GJ 0
Facility total capacity    : GJ 2000000
Facility I/W fuel costs    : Percent 0.5
Storage initial inventory  : GJ 0
Storage final inventory    : No condition
Futures contract size      : GJ 2500
Futures bid-ask spread     : USD/GJ 0
Time until first contract  : Days 30
Interest rate              : Percent 0
```

Optimal schedule:

Month	Long	Short	Bid	Ask	Capital	Inj.	Wit.	Inj.Lim.	Wit.Lim.	Volume
-					0					0
1	167	0	4.484	4.484	-1872070	417500	0	417900	0	415412
2	167	0	4.477	4.477	-3741218	417500	0	417900	415412	830825
3	139	0	4.548	4.548	-5321648	347500	0	348821	480000	1176587
4	119	0	4.597	4.597	-6689255	297500	0	298500	542529	1472600
5	119	0	4.584	4.584	-8052995	297500	0	298500	641200	1768612
6	92	0	4.660	4.660	-9124795	230000	0	231387	720000	1997462
7	1	0	4.851	4.851	-9136923	2500	0	2537	720000	1999950
8	0	129	5.101	5.101	-7491850	0	324120	50	720000	1675829
9	0	282	5.228	5.228	-3806110	0	708542	298500	708943	967286
10	0	191	5.210	5.210	-1318335	0	479899	309832	480000	487387
11	0	191	5.127	5.127	1129807	0	479899	417900	480000	7487
12	167	0	4.890	4.890	-911768	417500	0	417900	7487	422900
13	167	0	4.898	4.898	-2956683	417500	0	417900	422900	838312
14	138	0	4.931	4.931	-4657878	345000	0	346682	480000	1181587
15	119	0	4.954	4.954	-6131693	297500	0	298500	544195	1477600
16	119	0	4.979	4.979	-7612945	297500	0	298500	642866	1773612
17	90	0	4.980	4.980	-8733445	225000	0	226387	720000	1997487
18	0	0	5.050	5.050	-8733445	0	0	2512	720000	1997487
19	0	0	5.296	5.296	-8733445	0	0	2512	720000	1997487
20	0	133	5.535	5.535	-6893058	0	334170	2512	720000	1663316
21	0	280	5.673	5.673	-2921958	0	703517	298500	704772	959799
22	0	191	5.645	5.645	-226470	0	479899	311971	480000	479899
23	0	191	5.560	5.560	2428430	0	479899	417900	479899	0
24	0	0	5.220	5.220	2428430	0	0	417899	0	0

Intrinsic storage value: USD 2428430.0

Table 5. Details on optimal schedule for problem 2.

A.3 Problem 3: Complex daily ratchets

Intrinsic Storage Valuation

Problem parameters:

Facility minimum inventory : GJ 0
Facility total capacity : GJ 2000000
Facility I/W fuel costs : Percent 1
Storage initial inventory : GJ 0
Storage final inventory : No condition
Futures contract size : GJ 1000
Futures bid-ask spread : USD/GJ 0.002
Time until first contract : Days 30
Interest rate : Percent 2

Optimal schedule:

Month	Long	Short	Bid	Ask	Capital	Inj.	Wit.	Inj.Lim.	Wit.Lim.	Volume
-					0					0
1	415	0	4.476	4.478	-1858175	415000	0	415800	0	410850
2	415	0	4.461	4.463	-3710361	415000	0	415800	352948	821700
3	399	0	4.524	4.526	-5516360	399000	0	399328	500628	1216710
4	355	0	4.565	4.567	-7137804	355000	0	355877	562998	1568160
5	275	0	4.545	4.547	-8388215	275000	0	275028	583883	1840410
6	150	0	4.613	4.615	-9080409	150000	0	150528	597495	1988910
7	11	0	4.794	4.796	-9133162	11000	0	11090	600000	1999800
8	0	593	5.032	5.034	-6148918	0	598989	200	600000	1400810
9	0	569	5.149	5.151	-3219034	0	574747	321343	575515	826062
10	0	496	5.123	5.125	-678075	0	501010	399016	501317	325052
11	0	296	5.033	5.035	811658	0	298989	415800	299325	26062
12	415	0	4.792	4.794	-1177915	415000	0	415799	26062	436912
13	415	0	4.792	4.794	-3167424	415000	0	415800	369237	847762
14	397	0	4.816	4.818	-5080270	397000	0	397466	504743	1240792
15	352	0	4.831	4.833	-6781366	352000	0	352172	566801	1589272
16	268	0	4.847	4.849	-8080886	268000	0	269270	584938	1854592
17	143	0	4.840	4.842	-8773271	143000	0	143436	598204	1996162
18	3	0	4.900	4.902	-8787976	3000	0	3837	600000	1999132
19	0	0	5.130	5.132	-8787976	0	0	867	600000	1999132
20	0	594	5.353	5.355	-5608548	0	600000	867	600000	1399132
21	0	569	5.477	5.479	-2492184	0	574747	321989	575431	824385
22	0	496	5.441	5.443	206454	0	501010	399136	501052	323375
23	0	295	5.350	5.352	1784686	0	297979	415800	298276	25395
24	0	25	5.014	5.016	1910045	0	25252	415800	25395	142

Intrinsic storage value: USD 1910044.5

Table 6. Details on optimal schedule for problem 3.

B Matlab implementation

B.1 Installing CPLEX for Matlab

Follow the instructions on `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.cplex.help/CPLEX/MATLAB/topics/gs_install.html`.

B.2 Main script

Listing 1. Main script calculating the intrinsic value of a natural gas storage facility, file `intrinsicValuation.m`

```

% adding CPLEX installation folder to MATLAB path
baseDir = 'C:\Program Files\IBM\ILOG\CPLEX_Studio128\cplex';
addpath([baseDir,'\matlab\x64_win64']);
addpath([baseDir,'\examples\src\matlab']);

% setting the default text interpreterc for plots
set(0,'DefaultTextInterpreter','latex');

% loading problem parameters
contract = loadContractParams;
facility = loadFacilityParams;

% calculate and approximate monthly limits
facility = calculateMonthlyRates(facility);
facility = approximateMonthlyRates('inj',facility);
facility = approximateMonthlyRates('wit',facility);

% plotting daily and monthly limits
plotDailyLimits(facility)
plotMonthlyLimits(facility,1);

% setting CPLEX options
options = cplexoptimset('cplex');
options.Display                  = 'on';
options.timelimit                = 10;
options.mip.tolerances.mipgap = 1e-7;
options.emphasis.mip             = 3;
% emphasis on feasibilty        (= 1)
% resp. optimality              (= 2)
% resp. best bound improv.      (= 3)
```

```
% resp. balanced     (default = 0)

% Heuristic for creating initial feasible soultion
% sometimes highly beneficial, sometimes the opposite
%x0 = createInitialSolution(facility,contract,1);
x0 = []; % no initial solution

% creating CPLEX problem structure
problem = createCplexProblem(facility,contract,x0,options);

% solving the MILP
[x,fval,exitflag,output] = cplexmilp(problem);
checkConstraints(x,problem,1e-6)

% neting of months with simultaneous injection and withdrawal
[xn,fvaln] = getNettedSolution(x,facility,contract);
checkConstraints(xn,problem,1e-6)

% plotting & printing results
plotResults(xn,facility,contract);
printResult(xn,fvaln,facility,contract,'schedule.txt');
```

B.3 Loading the problem parameters

Listing 2. Loads the problem parameters of the storage facility, the daily ratchets on injection and withdrawal can be set here, file `loadFacilityParams.m`

```

function facility = loadFacilityParams()

facility.vMin = 0;    % Minimum gas volume of storage facility (in GJ)
facility.vMax = 2e6; % Maximum gas volume of storage facility (in GJ)
facility.fuelCost = 0.5; % injection-witdrawal fuel cost (in percent)
                         % due to pump usage

% daily injection resp. withdrawal limits as ratchets:
% piecewise constant limits (in GJ/day)
% depending on the working gas volume (in GJ)

% Problem 1: constant daily limits
% facility.dI = [0   , 2e6 ;    % piecewise constant interpolation
%                 12e3, 0   ];  % via interp1(...,'previous')
% facility.dW = [0   , 2e6 ;    % piecewise constant interpolation
```

```
%                 0   , 20e3];  % via interp1(...,'next')

% Problem 2: simple daily ratchets
facility.dI = [0   , 1e6 , 2e6 ;    % piecewise constant interpolation
              14e3, 10e3, 0   ];  % via interp1(...,'previous')
facility.dW = [0   , 1e6 , 2e6 ;    % piecewise constant interpolation
               0   , 16e3, 24e3];  % via interp1(...,'next')

% Problem 3: complex daily ratchets
% facility.dI = [0    , 1.0e6, 1.4e6, 1.7e6, 1.9e6, 2.0e6;
%                 14e3,  13e3,  11e3,   8e3,   4e3, 0    ];
% facility.dW = [0    , 0.1e6, 0.3e6, 0.7e6, 1.3e6, 2.0e6;
%                 0    ,   6e3,  12e3,  16e3,  19e3,  20e3];

end
```

Listing 3. Loads the problem parameters of the storage contract, the futures prices can be manipulated here, file `loadContractParams.m`

```

function contract = loadContractParams()

% Hint: Comment field 'vEnd' out if no constraint on end volume shall
% be imposed. In this case any remaining gas at contract end is lost.

contract.leadTime = 30; % Time (in days) from valuation date until
                        % delivery start of first futures contract
contract.vStart = 0;    % Initial gas volume of storage facility (GJ)
                        % with storage.vMin <= vStart <= storage.vMax
% contract.vEnd = 0;  % Terminal gas volume of storage facility (GJ)
%                     % with storage.vMin <= vEnd <= storage.vMax

% Monthly futures prices at valuation date (USD/GJ)
% paid at beginning of month,
% gas delivery from beginning to end of months
contract.futuresPrices = [4.484, 4.477,4.548, 4.597, 4.584, ...
    4.660, ...
  4.851, 5.101, 5.228, 5.210, 5.127, 4.890, 4.898, 4.931, 4.954, ...
  4.979, 4.98,5.050, 5.296, 5.535, 5.673, 5.645, 5.560, 5.220];

contract.contractSize = 2500; % Futures contract size (in GJ)
contract.discRate = 0;  % Interest rate for discounting (in percent)
contract.spread   = 0;  % Bid-ask spread on futures prices (USD/GJ)
```

```

end
```

B.4 Calculating the support points

Listing 4. Calculates the exact monthly limits on injection and withdrawal based on the daily limits, file `calculateMonthlyRates.m`

```

function facility = calculateMonthlyRates(facility)
% calculate monthly injection resp. withdrawal limits
% as function of the working gas volume

% unpack required storage attributes
dI         = facility.dI;
dW         = facility.dW;
vMin       = facility.vMin;
vMax       = facility.vMax;
lossFactor = (1-facility.fuelCost/100);

% discrete query points for the working gas volume
[vq,~] = getStorageGrid(facility);

vI = vq; % injected volume (starting with current working gas volume)
vW = vq; % withdrawn volume (starting with current working gas ...
    volume)

for i = 1:30 % assume 30 days per month
  vI = vI + lossFactor*interp1(dI(1,:), dI(2,:), vI, 'previous'); ...
      % with loss factor!
  vW = vW - interp1(dW(1,:), dW(2,:), vW, 'next');
  vI = min(vI, vMax); % cap at max volume
  vW = max(vW, vMin); % floor at min volume
end

facility.mIex = vI - vq; % exact monthly injection limit
facility.mWex = vq - vW; % exact monthly withdrawal limit

end
```

Listing 5. Calculates the support points of the continuous piecewise linear approximation of the monthly limits on injection and withdrawal, file `approximateMonthlyRates.m`

```
function facility = approximateMonthlyRates(iwSwitch,facility)

rateFacility = createRateFacility(iwSwitch,facility);
vMin         = rateFacility.vMin;
vMax         = rateFacility.vMax;
vGrid        = rateFacility.vGrid;
vDelta       = rateFacility.vDelta;
mRates       = rateFacility.mRex;

coefficients = getLinearCoefficients(rateFacility);
slopes       = coefficients.slopes;
intercepts   = coefficients.intercepts;

% constants
eps = 1e-10;
tol = eps*max(abs(vMin),abs(vMax));

s = [];
t = [];
w = [];
nodes = [];
start = nan;
targets = [];
edgeMap = [];

for ixCoeff = 1:length(slopes) % iterating through coefficients

  intersects = getIntersections(ixCoeff); % [x;y] desc. along x
  ixPreviousNode = nan;

  for ixSects = 1:size(intersects,2)
    ixNode = insertNode(intersects(:,ixSects));
    if ~isnan(ixPreviousNode)
      weight = getEdgeWeight(ixNode,ixPreviousNode,ixCoeff);%reverse!
      insertEdge(ixPreviousNode,ixNode,weight);
      edgeMap(ixPreviousNode,ixNode) = ixCoeff;
    end
    ixPreviousNode = ixNode;
  end

end

```

```
G = digraph(s,t,w);
targets = unique(targets);
[TR,D] = shortestpathtree(G,start,targets,'OutputForm','cell');

[~,ixPath] = min(D);
spCell = TR(ixPath);
sp = spCell{1}; % shortest path

% drop nodes lying within a graph segment
coeffIndices = [];
for ixNode = 1:length(sp)-1
  coeffIndices = [coeffIndices,edgeMap(sp(ixNode),sp(ixNode+1))];
end
if length(coeffIndices) >= 2
  ix = [1,coeffIndices(1:end-1)~=coeffIndices(2:end),1];
  sp = sp(ix==1);
end

supportPoints = fliplr(nodes(:,sp));
if strcmp(iwSwitch,'inj')
  facility.mI = supportPoints;
elseif strcmp(iwSwitch,'wit')
  facility.mW = [-fliplr(supportPoints(1,:));
                 fliplr(supportPoints(2,:))];
else
  error('Error.\nFirst argument must be ''inj'' or ''wit''.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nested functions
function intersections = getIntersections(i)
  x = (intercepts-intercepts(i))./(slopes(i)-slopes);
  x = x(~isinf(x));
  x = x(vMin<x & x<vMax);
  x = [vMin,x,vMax];
  x = fliplr(uniquetol(x,eps)); % sorted desc!
  y = slopes(i)*x + intercepts(i);
  intersections = [x;y];
end

function ixNode = insertNode(newNode)
  if isempty(nodes)
    nodes = newNode;
```

```
    ixNode = 1;
  else
    ixNode = find(sum(abs(nodes-newNode),1) < tol,1,'first');
    if isempty(ixNode)
      nodes = [nodes,newNode];
      ixNode = size(nodes,2);
    end
  end
  % check if newNode is the start node
  if isnan(start) && sum(abs(newNode-[vMax;0])) < tol
    start = ixNode;
  end
  % check if newNode is a target node
  if newNode(1)==vMin
    targets = [targets,ixNode];
  end
end

function insertEdge(source,target,weight)
  s = [s,source];
  t = [t,target];
  w = [w,weight];
end

function weight = getEdgeWeight(ixNode1,ixNode2,ixCoeff)
  slope = slopes(ixCoeff);
  intercept = intercepts(ixCoeff);
  x1 = nodes(1,ixNode1);
  x2 = nodes(1,ixNode2);
  x = vGrid(x1<=vGrid & vGrid<=x2);
  ya = slope*x + intercept;
  ye = mRates(round(1+(x-vMin)/vDelta));
  % weight = standardized sum of squared residuals
  weight = sum((ya-ye).^2)/max(abs(vMin),abs(vMax));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

Listing 6. Helper function needed by `approximateMonthlyRates.m`, reverses direction of withdrawal rates so that they can be treated like injection rates, file `createRateFacility.m`

```

```

```
function rateFacility = createRateFacility(iwSwitch, facility)

[vGrid, vDelta] = getStorageGrid(facility);

if strcmp(iwSwitch,'inj')

  rateFacility.dR     = facility.dI;
  rateFacility.mRex   = facility.mIex;
  rateFacility.c      = (1-facility.fuelCost/100); % loss factor
  rateFacility.vMin   = facility.vMin;
  rateFacility.vMax   = facility.vMax;
  rateFacility.vGrid  = vGrid;
  rateFacility.vDelta = vDelta;

elseif strcmp(iwSwitch,'wit')

  rateFacility.dR     = [-fliplr(facility.dW(1,:)); ...
                          fliplr(facility.dW(2,:))];
  rateFacility.mRex   = fliplr(facility.mWex);
  rateFacility.c      = 1;
  rateFacility.vMin   = -facility.vMax;
  rateFacility.vMax   = -facility.vMin;
  rateFacility.vGrid  = -fliplr(vGrid);
  rateFacility.vDelta = vDelta;

else

  error('Error.\nFirst argument must be ''inj'' or ''wit''.')

end

end
```

Listing 7. Helper function needed by `approximateMonthlyRates.m`, calculates the slope & intercept coefficients of the approximating piecewise linear functions, file `getLinearCoefficients.m`

```

function coefficients = getLinearCoefficients(rateFacility)

baseNodes   = rateFacility.dR(1,:);
dLimits     = rateFacility.dR(2,:);
mRates      = rateFacility.mRex;
```

```
vGrid      = rateFacility.vGrid;
vDelta     = rateFacility.vDelta;
vMin       = rateFacility.vMin;
startPoint = rateFacility.vMax;

preSupportNodes = getPreSupportNodes(rateFacility);

currentBaseNode = preSupportNodes(1);
ixBaseNode = 1;
ixDiff = 0;
slopes = [];
intercepts = [];

for ixNode = 2:length(preSupportNodes)

  ixDiff = ixDiff+1;
  currentNode = preSupportNodes(ixNode);

  if currentNode ~= currentBaseNode

    num    = dLimits(ixBaseNode+ixDiff-1)-dLimits(ixBaseNode);
    denom  = dLimits(ixBaseNode);
    slopes = [slopes,num/denom];

    startNode = preSupportNodes(ixNode-1);
    endNode   = currentNode;
    x = vGrid(startNode<=vGrid & vGrid < endNode);
    y = mRates(round(1+(x-vMin)/vDelta));

    ft = fittype(sprintf('(%f/%f)*x+b',num,denom));
    f = fit(x',y',ft,'StartPoint',startPoint);
    intercepts = [intercepts,f.b];

    if ~isempty(find(currentNode == baseNodes,1,'first'))
      % currentNode is a new base node
      ixBaseNode = find(currentNode == baseNodes,1,'first');
      currentBaseNode = currentNode;
      ixDiff = 0;
    end
  end
end

coefficients.slopes = slopes;
```

```
coefficients.intercepts = intercepts;

end
```

Listing 8. Helper function needed by `getLinearCoefficients.m`, finds all pre-support nodes which partition the storage volume into intervals on which one approximates the exact monthly rates linearly with potential discontinuities at the interval ends, file `getPreSupportNodes.m`

```
function nodes = getPreSupportNodes(rateFacility)
% returns all pre-support nodes for the continuous piecewise linear
% approximation of the monthly inj./wit. limits

dNodes = rateFacility.dR(1,:);
n = length(dNodes);

nodes = [];
for ixStart = 1:n
  node = dNodes(ixStart); % daily nodes are monthly pre-support nodes
  ixEnd = ixStart;
  while ~isnan(node)
    nodes = [nodes,node];
    ixEnd = ixEnd+1;
    node = calculatePreSupportNode(rateFacility,ixStart,ixEnd);
  end
end

end
```

Listing 9. Helper function needed by `getPreSupportNodes.m`, searches for pre-support node in given bounds, file `calculatePreSupportNode.m`

```
function node = calculatePreSupportNode(rateFacility,ixStart,ixEnd)
% searches for pre-support node
% in [dNodes(ixStart), dNodes(ixStart+1)[
% = min node within interval
% where daily limit switches to dLimit(ixEnd)
% within first 30 days
%
% rateFacility: facility rate params for injection resp. withdrawal
% ixStart     : index of start node
```

```
% ixEnd        : index of terminal node, required: ixEnd>=ixStart+1

% constants
eps  = 1e-2; % x<y -> x<=y-eps for (Aineq,bineq) constraints
days = 30;    % days per months

dNodes  = rateFacility.dR(1,:);
dLimits = rateFacility.dR(2,:);
c       = rateFacility.c;

% index-out-of-bound check
% needed for while loop in invoking method getSupportNodes()
if ixEnd > length(dNodes)
  node = nan;
  return
end

n = ixEnd-ixStart; % number of day count variables

Age = -[ones(n,1), c*tril(repmat(dLimits(ixStart:ixEnd-1),n,1))];
Al  = -Age;
bge = -dNodes(ixStart+1:ixEnd)';
bl  = dNodes(ixStart+1:ixEnd)' + c*dLimits(ixStart:ixEnd-1)' -eps;
A   = [Age;Al;[0,ones(1,n)]];
b   = [bge;bl;days-1];

lb = [dNodes(ixStart);ones(n,1)];
ub = [dNodes(ixStart+1)-eps;(days-1)*ones(n,1)];

f = [1,zeros(1,n)];
intcon = 2:n+1;

options = optimoptions('intlinprog','Display','none');
res = intlinprog(f,intcon,A,b,[],[],lb,ub,options);

if isempty(res)
  node = nan;
  return
end

node = res(1);

% correction for last descent
```

```
if ixEnd==length(dNodes) ...
   && node-dLimits(ixStart)>=dNodes(ixStart)
  node = node-dLimits(ixStart);
end

end
```

B.5 Finding an initial feasible solution

Listing 10. Heuristic to calculate an initial solution of the storage problem, file `createInitialSolution.m`

```

function solution = createInitialSolution(facility,contract,loops)

problem = createCplexProblem(facility,contract);

% unpack struct fields
vStart     = contract.vStart;
fuelRate   = (1-facility.fuelCost/100);
D          = contract.contractSize;
fp         = contract.futuresPrices;
discRate   = contract.discRate;
dT         = contract.leadTime;
spread     = contract.spread;

n   = length(fp);

discFactors = exp(-discRate/100*(dT+(0:n-1)*30)/360);
discBid = (fp - spread/2).*discFactors;
discAsk = (fp + spread/2).*discFactors;
spreads = triu(-discAsk'+discBid);
spreads(spreads<0) = 0;

rows = 1:n;
cols = 1:n;

Ax =     fuelRate *D*tril(ones(n),-1);
Ay = (1/fuelRate)*D*tril(ones(n),-1);

% initialize solution
x = zeros(n,1);
y = zeros(n,1);
```

```
loops = max(n, loops);
for loop = 1:loops
  amendSolution;
end

solution = completeSolution([x;y],facility,contract,1e-9);
% end of outer function


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nested function
function amendSolution

[~,argmax] = max(spreads(:));
[row,col] = ind2sub(size(spreads),argmax);

vols = vStart + Ax*x - Ay*y; % storage volume at each month beginning
vol = vols(rows(row));
[iLimit, ~] = getMonthlyLimit(facility,vol);
% maximum number of contracts that can be injected
xs = floor(iLimit/D);
dx = zeros(n,1);
dx(rows(row)) = xs;

vols = vols + Ax*dx; % updated storage volume at each month beginning
vol = vols(cols(col));
[~, wLimit] = getMonthlyLimit(facility,vol);
% maximum number of contracts that can be withdrawn
ys = floor(fuelRate*min(wLimit,fuelRate*D*xs)/D);
dy = zeros(n,1);
dy(cols(col)) = ys;

solution = completeSolution([x;y],facility,contract,1e-9);
if checkConstraints(solution,problem,1e-6) == 0 && netCashFlow >= 0
  x = x+dx;
  y = y+dy;
end

spreads(row,:) = [];
spreads(:,col) = [];
rows(row) = [];
cols(col) = [];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% inner nested function
function net = netCashFlow
  fx = fp(rows(row));
  fy = fp(cols(col));
  net = D*(-xs*fx + ys*fy);
  vols = vols + Ay*dy;
  [~, wLimit] = getMonthlyLimit(facility,vols(n));
  if vols(n) <= wLimit
    dVn = D*(fuelRate*xs - (1/fuelRate)*ys);
    net = net + dVn*fp(n);
  end
end
% end of inner nested function
end
% end of nested function

end
```

Listing 11. Helper function needed by `createInitialSolution.m`, returns monthly injection and withdrawal limits given the current storage gas level, file `getMonthlyLimit.m`

```

function [iLimit,wLimit] = getMonthlyLimit(storage,gasLevel)
% returns monthly injection and withdrawal limits based on the
% current storage gas level

iNodes  = storage.mI(1,:);
iLimits = storage.mI(2,:);
wNodes  = storage.mW(1,:);
wLimits = storage.mW(2,:);

ixI = sum(gasLevel >= iNodes,2);
ixW = sum(gasLevel >= wNodes,2);

% add fake nodes
iNodes  = [iNodes, iNodes(end)+1];
iLimits = [iLimits, iLimits(end)+1];
wNodes  = [wNodes, wNodes(end)+1];
wLimits = [wLimits, wLimits(end)+1];

iLambda = (gasLevel-iNodes(ixI))/(iNodes(ixI+1)-iNodes(ixI));
```

```
wLambda = (gasLevel-wNodes(ixW))/(wNodes(ixW+1)-wNodes(ixW));

iLimit = iLimits(ixI) + iLambda*(iLimits(ixI+1)-iLimits(ixI));
wLimit = wLimits(ixW) + wLambda*(wLimits(ixW+1)-wLimits(ixW));

end
```

B.6 Creating the CPLEX problem structure

Listing 12. Creates a CPLEX MILP structure based on the loaded problem parameters, file `createCplexProblem.m`

```
function problem = createCplexProblem(facility,contract,x0,options)

% unpack struct fields
vMin       = facility.vMin;
vMax       = facility.vMax;
vStart     = contract.vStart;
mI         = facility.mI;
mW         = facility.mW;
fuelRate   = (1-facility.fuelCost/100);
D          = contract.contractSize;
fp         = contract.futuresPrices;
discRate   = contract.discRate;
dT         = contract.leadTime;
spread     = contract.spread;

n  = length(fp);
m1 = size(mI,2);
m2 = size(mW,2);

discFactors = exp(-discRate/100*(dT+(0:n-1)*30)/360);
discBid = (fp - spread/2).*discFactors;
discAsk = (fp + spread/2).*discFactors;

% objective function
% decision variables i=1:n represent numbers
% of long contracts per month i
% decision variables i=n+1:2n represent numbers
% of short contracts per month i-n
f = D*[discAsk,-discBid,zeros(1,n*(m1+m2))];

```

```
% numeric type of decision variable at position i
% -> cytpe(i) with 'I' = Integer, 'C' = continuous
ctype = [repmat('I',1,2*n),repmat('C',1,n*(m1+m2))];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preparation for volume equality constraints:
% Storage volume at beginning of each month must be represented as
% convex combination of its two neighbouring injection resp.
% withdrawal nodes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Ax  = D*   fuelRate *tril(ones(n),-1);
Ay  = D*(1/fuelRate)*tril(ones(n),-1);
Axy = [Ax,-Ay;
       Ax,-Ay];

iNodes = mI(1,:); % injection nodes (facility levels) for mI limits
wNodes = mW(1,:); % withdrawal nodes (facility levels) for mW limits

iNodesBlocks = repmat({iNodes},1,n);
iNodesDiag   = blkdiag(iNodesBlocks{:});
wNodesBlocks = repmat({wNodes},1,n);
wNodesDiag   = blkdiag(wNodesBlocks{:});

Aln = [iNodesDiag            ,zeros(size(wNodesDiag));
       zeros(size(iNodesDiag)),wNodesDiag            ];

iOnesBlocks = repmat({ones(1,m1)},1,n);
iOnesDiag   = blkdiag(iOnesBlocks{:});
wOnesBlocks = repmat({ones(1,m2)},1,n);
wOnesDiag   = blkdiag(wOnesBlocks{:});

Al1 = [iOnesDiag             ,zeros(size(wOnesDiag));
       zeros(size(iOnesDiag)),wOnesDiag             ];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Formulation of volume equality constraints:
% Storage volume at beginning of each month = convex combination
% of its two adjacent injection resp. withdrawal nodes.
% The lambda multipliers of the convex combinations are given by the
% decision variables 2n+1:2n+n(m1+m2) in the form of special ordered
```

```
% sets of type 2 (n sets of length m1 and n sets of length m2),
% each summing up to 1. This ensures the required convexity &
% adjacency property.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Aeq = [-Axy            ,Aln;
       zeros(size(Axy)),All];

beq = [vStart*ones(2*n,1);
              ones(2*n,1)];

% special ordered sets of type 2
% SOS2 = set of integer or continuous variables with at most two
% nonzero variables. If two variables are nonzero they must be
% adjacent in the set
sos2 = [];
for i = 1:n
  sosI.type = '2';
  sosI.ind  = n*2 + ((i-1)*m1+1:i*m1)';
  sosI.wt   = iNodes';
  sos2      = [sos2; sosI];
end
for i = 1:n
  sosW.type = '2';
  sosW.ind  = n*(2+m1) + ((i-1)*m2+1:i*m2)';
  sosW.wt   = wNodes';
  sos2      = [sos2; sosW];
end

% If vEnd is specified in the facility contract, add equality
% contraint for the facility volume at contract end
if isfield(contract,'vEnd')
  vEnd = contract.vEnd;
  Aeq  = [Aeq;D*[   fuelRate*ones(1,n),...
                 -1/fuelRate*ones(1,n),...
                 zeros(1,n*(m1+m2))    ]];
  beq  = [beq;vEnd-vStart];
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inequality constraints:
% Monthly i/w volumes <= monthly i/w limts
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Bx  = D             *diag(ones(1,n));
By  = D*(1/fuelRate)*diag(ones(1,n));
Bxy = [Bx             ,zeros(size(By));
       zeros(size(Bx)),By             ];

iLimits = mI(2,:); % limit nodes for monthly injections
wLimits = mW(2,:); % limit nodes for monthly withdrawals

iLimitsBlocks = repmat({iLimits},1,n);
iLimitsDiag   = blkdiag(iLimitsBlocks{:});
wLimitsBlocks = repmat({wLimits},1,n);
wLimitsDiag   = blkdiag(wLimitsBlocks{:});

Bl = [iLimitsDiag             ,zeros(size(wLimitsDiag));
      zeros(size(iLimitsDiag)),wLimitsDiag             ];

A = [Bxy,-Bl];
b = zeros(2*n,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% lower for all decision variables
lb = zeros(n*(2+m1+m2),1);

% create problem
problem.f       = f;
problem.ctype   = ctype;
problem.Aineq   = A;
problem.bineq   = b;
problem.Aeq     = Aeq;
problem.beq     = beq;
problem.sos     = sos2;
problem.lb      = lb;
problem.ub      = [];
problem.x0      = []; % default: no starting point
problem.options = cplexoptimset('MaxTime',120); % default options

if nargin>=3 && ~isempty(x0)
  problem.x0 = x0;
end

```

```
if nargin>=4 && ~isempty(options)
  problem.options = options;
end

end
```

B.7 Plotting and printing

Listing 13. Plots the daily limits on injection and withdrawal, file `plotDailyLimits.m`

```

function plotDailyLimits(facility)

% unpack required storage attributes
dI      = facility.dI;
dW      = facility.dW;

% query points along working gas volume
[vq,vDelta] = getStorageGrid(facility);

figure('DefaultAxesFontSize',18)
hold on;

h(1) = plot(nan,nan,'r-');
h(2) = plot(nan,nan,'b-');

dIq = interp1(dI(1,:),dI(2,:),vq, 'previous');
p1 = plot(dI(1,:),dI(2,:),'ro');
p2 = plot(vq,dIq,'r:.');

dWq = interp1(dW(1,:),dW(2,:),vq, 'next');
p3 = plot(dW(1,:),dW(2,:),'bo');
p4 = plot(vq,dWq,'b:.');

hold off;

% plot setting
box on;
xlabel('Storage volume (GJ)','fontsize', 18);
ylabel('Daily I/W limit (GJ)','fontsize', 18);
ylim([0 max([dI(2,:),dW(2,:)])+vDelta]);
[~, hobj] = legend(h,{'daily $I(V)$', 'daily $W(V)$'},...
  'orientation', 'vertical', 'location', 'south');
```

```
lineobj = findobj(hobj,'type','line');
set(lineobj,'LineWidth',1.25);
textobj = findobj(hobj,'type','text');
set(textobj, 'interpreter', 'latex', 'fontsize', 15);

end
```

Listing 14. Plots the exact monthly limits on injection and withdrawal, optionally with highlighted support points of the continuous piecewise linear approximation, file `plotMonthlyLimits.m`

```

function plotMonthlyLimits(facility,withSupport)

% unpack required storage attributes
mIex = facility.mIex;
mWex = facility.mWex;
if withSupport
  mI    = facility.mI; % approx.
  mW    = facility.mW; % approx.
end

% query points along working gas volume
[vq,~] = getStorageGrid(facility);

figure('DefaultAxesFontSize',18)
hold on;

h(1) = plot(nan,nan,'r-');
h(2) = plot(nan,nan,'b-');

if withSupport
  h(3) = plot(nan,nan,'mo');
  plot(mI(1,:),mI(2,:),'mo-',... % approx. inj.
     'MarkerSize',8,'LineWidth',1.25);
  plot(mW(1,:),mW(2,:),'mo-',... % approx. wit.
     'MarkerSize',8,'LineWidth',1.25);
end

plot(vq,mIex,'r.'); % exact inj.
plot(vq,mWex,'b.'); % exact wit.

hold off;
```

```
% plot setting
text = {'monthly $I(V)$', 'monthly $W(V)$'};
if withSupport
  text{3} = 'support points';
end
box on;
xlabel('Storage volume (GJ)','fontsize', 18);
ylabel('Monthly I/W limit (GJ)','fontsize', 18);
[~, hobj] = legend(h,text,...
  'orientation','vertical','location','south');
lineobj = findobj(hobj,'type','line');
set(lineobj,'LineWidth',1.25);
textobj = findobj(hobj,'type','text');
set(textobj, 'interpreter', 'latex', 'fontsize', 15);

end
```

Listing 15. Plots the resulting injection and withdrawal schedule including monthly limits and storage gas level, file `plotResults.m`

```

function plotResults(solution,facility,contract)

% unpack struct fields
mI        = facility.mI;
mW        = facility.mW;
vStart    = contract.vStart;
vMax      = facility.vMax;
fuelRate = (1-facility.fuelCost/100);
D         = contract.contractSize;
fp        = contract.futuresPrices;

n  = length(fp);
m1 = size(mI,2);
m2 = size(mW,2);

% split decision variables
x = solution(1:n);                  %number of long futures contracts
y = solution(n+1:2*n);              %number of short futures contracts
lambdaI = solution(2*n+1:2*n+n*m1);%lambda multipliers for injections
lambdaW = solution(2*n+n*m1+1:2*n+n*m1+n*m2);  % ... and withdrawals

```

```
Ax = D*    fuelRate *tril(ones(n));
Ay = D*(1/fuelRate)*tril(ones(n));
vols = vStart + Ax*x - Ay*y;  % storage volumes at beginning of month

injLimitBlocks = repmat({mI(2,:)},1,n);
injLimitDiag   = blkdiag(injLimitBlocks{:});
witLimitBlocks = repmat({mW(2,:)},1,n);
witLimitDiag   = blkdiag(witLimitBlocks{:});

% monthly i/w limits based on storage volume at beginning of month
injMax = injLimitDiag*lambdaI;
witMax = witLimitDiag*lambdaW;

months  = 1:n;
months1 = 1:n+1;


figure('DefaultAxesFontSize',14)
hold on;

% plot monthly inj. volumes & limits
bar(months+0.5, D*x, 0.8, 'FaceColor', [0 0 .5])
stairs(months1, [injMax', injMax(end)], 'r:', 'LineWidth', 1.5);

% plot monthly withdrawals volumes & limits
bar(months+0.5, -(1/fuelRate)*D*y, 0.8, 'FaceColor', [0 .5 0])
stairs(months1, -[witMax', witMax(end)], 'b:', 'LineWidth', 1.5);

% plot storage volumes (at beginning of each month)
plot(months1, [vStart, vols'], 'ko-', 'MarkerSize', 5);

hold off;

% plot settings
box on
xlim([0.5 25.5]); xticks([1,5:5:25]);
xlabel('Months','fontsize', 14);
ylabel('Gas volume (GJ)','fontsize', 14);
[~, hobj] = legend({'Injection', 'Inj. limit', 'Withdrawal', ...
  'With. limit', 'Volume'}, ...
  'orientation', 'vertical','location','northeastoutside');
textobj = findobj(hobj, 'type', 'text');
set(textobj, 'interpreter', 'latex', 'fontsize', 11);
```

```
set(gcf,'units','centimeter','Position',[10, 5, 20, 10]);

end
```

Listing 16. Prints full details on the resulting injection and withdrawal schedule, file `printResult.m`

```

function printResult(solution,fval,facility,contract,filename)

% unpack struct fields
mI          = facility.mI;
mW          = facility.mW;
vStart      = contract.vStart;
if isfield(contract,'vEnd')
  vEndStr   = sprintf('GJ %i',contract.vEnd);
else
  vEndStr   = 'No condition';
end
vMin        = facility.vMin;
vMax        = facility.vMax;
fuelCost    = facility.fuelCost;
lossFactor = (1-facility.fuelCost/100);
D           = contract.contractSize;
fp          = contract.futuresPrices;
discRate    = contract.discRate;
dT          = contract.leadTime;
spread      = contract.spread;

n   = length(fp);
m1 = size(mI,2);
m2 = size(mW,2);

disc = exp(-discRate/100*(dT+(0:n-1)*30)/360);
a = (fp - spread/2).*disc;
b = (fp + spread/2).*disc;

lng      = solution(1:n);
sht      = solution(n+1:2*n);
lambdaI = solution(2*n+1:2*n+n*m1);
lambdaW = solution(2*n+n*m1+1:2*n+n*m1+n*m2);

inj = D*lng;
```

```
wit = (1/lossFactor)*D*sht;

% cummalitive cashflow at beginning of each month
cap = vStart+D*(-tril(repmat(a,n,1))*lng+tril(repmat(b,n,1))*sht);

% storage volumes at beginning of month
vol = vStart + lossFactor*tril(ones(n))*inj - tril(ones(n))*wit;

injLimitBlocks = repmat({mI(2,:)},1,n);
injLimitDiag   = blkdiag(injLimitBlocks{:});
witLimitBlocks = repmat({mW(2,:)},1,n);
witLimitDiag   = blkdiag(witLimitBlocks{:});

% monthly i/w limits based on storage volume at beginning of month
iLt = injLimitDiag*lambdaI;
wLt = witLimitDiag*lambdaW;


% build txt string
newline = '\n';
space   = '   ';
width   = 93;
mline   = pad('',width+2,'left','-');

txt = 'Intrinsic Storage Valuation ';
txt = [txt newline mline];

txt = [txt newline 'Problem parameters:'];
txt = [txt newline mline];
txt = [txt newline...
       pad('Facility minimum inventory',26,'right') ' : '...
       sprintf('GJ %i',vMin)];
txt = [txt newline...
       pad('Facility total capacity'   ,26,'right') ' : '...
       sprintf('GJ %i',vMax)];
txt = [txt newline...
       pad('Facility I/W fuel costs'   ,26,'right') ' : '...
       sprintf('Percent %g',fuelCost)];
txt = [txt newline...
       pad('Storage initial inventory' ,26,'right') ' : '...
       sprintf('GJ %i',vStart)];
txt = [txt newline...
       pad('Storage final inventory'   ,26,'right') ' : '...
```

```
        sprintf('%s',vEndStr)];
txt = [txt newline...
        pad('Futures contract size'     ,26,'right') ' : '...
        sprintf('GJ %i',D)];
txt = [txt newline...
        pad('Futures bid-ask spread'    ,26,'right') ' : '...
        sprintf('USD/GJ %g',spread)];
txt = [txt newline...
        pad('Time until first contract' ,26,'right') ' : '...
        sprintf('Days %i',dT)];
txt = [txt newline...
        pad('Interest rate'             ,26,'right') ' : '...
        sprintf('Percent %g',discRate)];
txt = [txt newline mline];

txt = [txt newline 'Optimal schedule:'];
txt = [txt newline mline];

% header line
txt = [txt newline...
        pad('Month'   ,5,'left') '  ' ...
        pad('Long'    ,4,'left') '  ' ...
        pad('Short'   ,5,'left') space...
        pad('Bid'     ,5,'left') space...
        pad('Ask'     ,5,'left') space...
        pad('Capital' ,8,'left') space...
        pad('Inj.'    ,6,'left') space...
        pad('Wit.'    ,6,'left') '  ' ...
        pad('Inj.Lim.',8,'left') '  ' ...
        pad('Wit.Lim.',8,'left') space...
        pad('Volume'  ,7,'left') ];

% month 0 line
txt = [txt newline...
        pad('-'                          ,5,'right') '  ' ...
        pad(''                           ,4,'left' ) '  ' ...
        pad(''                           ,5,'left' ) space...
        pad(''                           ,5,'left' ) space...
        pad(''                           ,5,'left' ) space...
        pad(sprintf('%i',floor(vStart)),8,'left' ) space...
        pad(''                           ,6,'left' ) space...
        pad(''                           ,6,'left' ) '  ' ...
        pad(''                           ,8,'left' ) '  ' ...
```

```
       pad(''                              ,8,'left' ) space...
       pad('0'                             ,7,'left' ) ];

% lines for months 1 to n
for i = 1:n
  month    = pad(sprintf('%i'   ,i)            ,5,'right');
  long     = pad(sprintf('%i'   ,round(lng(i))),4,'left' );
  short    = pad(sprintf('%i'   ,round(sht(i))),5,'left' );
  bid      = pad(sprintf('%4.3f',b(i))         ,5,'left' );
  ask      = pad(sprintf('%4.3f',a(i))         ,5,'left' );
  capital  = pad(sprintf('%i'   ,round(cap(i))),8,'left' );
  inject   = pad(sprintf('%i'   ,floor(inj(i))),6,'left' );
  withdraw = pad(sprintf('%i'   ,floor(wit(i))),6,'left' );
  injLimit = pad(sprintf('%i'   ,floor(iLt(i))),8,'left' );
  witLimit = pad(sprintf('%i'   ,floor(wLt(i))),8,'left' );
  volume   = pad(sprintf('%i'   ,floor(vol(i))),7,'left' );
  txt = [txt newline...
         month '  ' long '  ' short space bid space ask space...
         capital space inject space withdraw '  ' injLimit '  '...
         witLimit space volume];
end
txt = [txt newline mline];
txt = [txt newline...
       sprintf('Intrinsic storage value: USD %.1f',-fval)];
txt = [txt newline mline];


fid = fopen(filename,'wt');
fprintf(fid,txt);
fclose(fid);

end
```

Listing 17. Plots the futures mid, ask and bid prices, file `plotFuturesCurve.m`

```

function plotFuturesCurve(contract)

% unpack struct fields
fp         = contract.futuresPrices;
discRate   = contract.discRate;
dT         = contract.leadTime;
spread     = contract.spread;
```

```
n  = length(fp);
months = 1:n;

discFactors = exp(-discRate/100*(dT+(0:n-1)*30)/360);
discBid     = (fp - spread/2).*discFactors;
discAsk     = (fp + spread/2).*discFactors;

figure('DefaultAxesFontSize',18)
plot(months,fp,'o-','Color',[0 0.5 0]);
box on;
grid on;
% ylim([4.3 5.8]);
xlabel('Months','fontsize', 18);
ylabel('Futures prices (USD/GJ)','fontsize', 18);
[~, hobj] = legend('mid',...
  'orientation','vertical','location','northwest');
lineobj = findobj(hobj,'type','line');
set(lineobj,'LineWidth',1.25);
textobj = findobj(hobj,'type','text');
set(textobj, 'interpreter', 'latex', 'fontsize', 15);

figure('DefaultAxesFontSize',18)
hold on;
plot(months,discAsk,'ro-');
plot(months,discBid,'bo-');
hold off;
box on;
grid on;
% ylim([4.3 5.4]);
xlabel('Months','fontsize', 18);
ylabel('Futures prices (USD/GJ)','fontsize', 18);
[~, hobj] = legend({'ask','bid'},...
  'orientation','vertical','location','northwest');
lineobj = findobj(hobj,'type','line');
set(lineobj,'LineWidth',1.25);
textobj = findobj(hobj,'type','text');
set(textobj, 'interpreter', 'latex', 'fontsize', 15);

end
```

B.8 Miscellaneous

Listing 18. Helper function to calculate the greatest gas delta volume to accurately represent daily limits, also returns corresponding volume grid, file `getStorageGrid.m`

```

function [vGrid,vDelta] = getStorageGrid(facility)

% greatest gas delta volume to accurately represent daily limits
vDelta = double(gcd(sym(...
  [facility.dI(1,:),facility.dW(1,:),...
   facility.dI(2,:),facility.dW(2,:)])));

% grid of gas volumes
vGrid = facility.vMin:vDelta:facility.vMax;

end
```

Listing 19. Helper function to net months with simultaneous injection and withdrawal in the CPLEX solution, file `getNettedSolution.m`

```

function [netted,fval] = ...
    getNettedSolution(solution,facility,contract)

n = length(contract.futuresPrices);
Axy = [solution(1:n),solution(n+1:2*n)];
[~, argmax] = max(Axy,[],2);

for i = 1:n
  j = argmax(i);
  Axy(i,j)   = round(Axy(i,j)-Axy(i,3-j));
  Axy(i,3-j) = 0;
end

x = Axy(:,1);
y = Axy(:,2);
netted = completeSolution([x;y],facility,contract,1e-9);

problem = createCplexProblem(facility,contract);
fval = problem.f*netted;

end
```

Listing 20. Helper function to set the lambda decision variables of a given partial solution (short and long variables only) of the MILP, file `completeSolution.m`

```

function solution = completeSolution(partial,facility,contract,eps)

% unpack struct fields
iNodes    = facility.mI(1,:); % nodes (storage vols) for inj. limits
wNodes    = facility.mW(1,:); % nodes (storage vols) for wit. limits
vStart    = contract.vStart;
fuelRate = (1-facility.fuelCost/100);
D         = contract.contractSize;

n  = length(partial)/2;
x  = partial(1:n);
y  = partial(n+1:2*n);

Ax   = D*    fuelRate *tril(ones(n),-1);
Ay   = D*(1/fuelRate)*tril(ones(n),-1);
vols = vStart + Ax*x - Ay*y;

lambdaI = createLambdas(vols,iNodes,eps);
lambdaW = createLambdas(vols,wNodes,eps);

solution = [x;y;lambdaI;lambdaW];

end


function lambda = createLambdas(vols,nodes,eps)
% ensured:
% vMin <= vols  <= vMax
% vMin <= nodes <= vMax
% nodes(1)  = vMin, nodes(end)  = vMax

n = length(vols);
m = length(nodes);

ixNode = sum(vols >= nodes,2);

% add dummy node for delta calculation
% to avoid index-out-of-bounds error
nodes = [nodes, 2*nodes(end)];
delta = (vols-nodes(ixNode)')./(nodes(ixNode+1)'-nodes(ixNode)');
```

```

lambda = zeros(n*m, 1);
ix = m*(0:n-1)' + ixNode; % ixNode to 1D index
ix1 = ix(ixNode<m);
lambda(ix1) = 1-delta(ixNode<m);
lambda(ix1+1) = delta(ixNode<m);
ix2 = ix(ixNode==m);
lambda(ix2) = 1;

% correction of floating point errors
lambda(abs(lambda-1)<eps) = 1;
lambda(abs(lambda-0)<eps) = 0;

end
```

Listing 21. Helper function to double-check if a given MILP solution really satisfies all constraints within a given tolerance, file `checkConstraints.m`

```

function isFeasible = checkConstraints(x,problem,eps)

% unpack problem fields
ctype = problem.ctype;
A     = problem.Aineq;
b     = problem.bineq;
Aeq   = problem.Aeq;
beq   = problem.beq;
lb    = problem.lb;
ub    = problem.ub;
sos2  = problem.sos;

indices = 1:length(x);
intcon = indices(ctype=='I');


if ~isempty(A) && sum(A*x<=b+eps) < length(b)
  isFeasible = 1;
  return
end

if ~isempty(Aeq) && sum(abs(Aeq*x-beq)<=eps) < length(beq)
  isFeasible = 2;
  return
```

```
end

if ~isempty(lb) && sum(lb-eps<=x) < length(x)
  isFeasible = 3;
  return
end

if ~isempty(ub) && sum(x<=ub+eps) < length(x)
  isFeasible = 4;
  return
end

if ~isempty(intcon) && sum( ...
    abs(x(intcon)-floor(x(intcon)))<=eps ...
    | abs(x(intcon)-ceil(x(intcon)))<=eps) ...
    < length(x(intcon))
  isFeasible = 5;
  return
end

for i = 1:length(sos2)
  sosT = sos2(i);
  ix = find(abs(x(sosT.ind))>=eps);
  if length(ix) > 2
    isFeasible = 6;
    return
  end
  if length(ix) == 2 && ix(1)+1~=ix(2)
    isFeasible = 7;
    return
  end
end

isFeasible = 0;

end
```