AGRODEP
African Growth & Development Policy modeling consortium
FACILITATED BY IFPRI

AGRODEP Technical Note 06

February, 2013

Modeling commodity markets in stochastic contexts:

A practical guide using the RECS toolbox version 0.5

Christophe Gouel

INRA and CEPII (christophe.gouel@grignon.inra.fr)

Contents

1 Introduction

1.1 Background

Variants of rational expectations storage models are central to neoclassical studies of the behavior of markets for storable commodities (Williams and Wright, 1991, Wright, 2001). Simple versions are tested econometrically, for example in Deaton and Laroque (1992) and Cafiero et al. (2011). More complex models are used to analyze the effects of public interventions in commodity markets (Miranda and Helmberger, 1988, Gouel and Jean, 2012).

Like most dynamic stochastic problems, this model cannot be solved analytically. But contrary to most stochastic problems studied, it presents specific numerical difficulties related to the non-negativity constraint on storage. This feature prevents this model from being solved with popular software, such as Dynare,[1] which rely on perturbation methods and cannot handle occasionally binding constraints.

The lack of user-friendly softwares to solve storage models in the past may have represented a serious barrier to entry for research on these issues.[2] The RECS toolbox provides a modeling environment allowing economists to focus on the economic problem at hand, while abstracting from various issues related to the numerical implementation.

This paper describes the RECS toolbox and also several applications of this modeling framework to commodity markets related issues.

This document assumes basic knowledge of Matlab,[3] and of dynamic economic models (see Adda and Cooper, 2003, for an introduction). Storage models are presented in brief; for more information please refer to the original papers or to Williams and Wright (1991), which provide detailed descriptions of many of these models.

1.2 Storage modeling in the African context

Corresponding to the needs of AGRODEP members, in this section we emphasize the possible applications of this toolbox in the context of African countries.

Given recent events on world food markets, many countries are contemplating or have introduced policies to achieve greater independence from the world market and to protect their vulnerable populations from global food price spikes. Although these policies play a role in the development of many countries (see Rashid et al., 2008, for the case of Asian countries), they often prove quite costly and difficult to manage (Gilbert, 1996, Jayne and Jones, 1997). Because these policies are designed to affect the whole market, they are not amenable to assessment by randomized trials as in the case of policies targeting households. These price policies affect the whole economy and every policy experiment can be very costly. In this situation,

[1] `http://www.dynare.org/`.

[2] The solvers `remsolve` and `resolve` from Miranda and Fackler (2002) and Fackler (2005), respectively, provided a relatively user-friendly approach to solving such models.

[3] See `http://www.mathworks.fr/academia/student_center/tutorials/launchpad.html` for tutorials.

it is useful to be able to test and analyze stabilization policies without entailing any human or fiscal costs, using simulated economies that represent most of the important facts related to commodity price behavior. This points to the role of the RECS toolbox, which provides a simple environment in which policy can be designed (for applications, see Gouel, 2011, 2013, Gouel and Jean, 2012). It can be used to compare the costs and effectiveness of stabilization policies with other forms of interventions. For example, using RECS, Larson et al. (2012) compare the cost of a storage policy designed to protect consumers from high prices in Middle East and North African countries with safety nets that provide an equivalent protection. Sections 5.5 and 12.1 present two examples of public storage policies that illustrate how RECS can be used to analyze them.

The RECS toolbox can be useful to analyze international spillovers from domestic policies and the interactions among different markets. Most domestic agricultural price stabilization policies have an effect on partner countries. Historically, this was the case with the European CAP, which stabilized the European market for several products (e.g., dairy, wheat) through storage and trade interventions, and, in so doing, forced the rest of the world to shoulder more important adjustments. The recent turmoil in the rice market is a good example of what can happen when many countries, at the same moment, attempt to insulate their markets from events on the world market (Slayton, 2009). In this context, African countries are not innocent bystanders: they also implement policies that affect their and their partners' markets (Porteous, 2012). All these situations can be analyzed using RECS, for example, by reproducing the framework of Makki et al. (1996, 2001). The storage-trade model presented in Section 12.2 can be a starting point for this analysis, and extensions of it.

The rational expectations modeling allowed by RECS can be used also to analyze how the market reacts to weather shocks, which might allow better calibration of the information needed for public interventions. An analysis of reactions to news in a rational expectations storage model is provided for example in Osborne (2004) for the Ethiopian grain market.

The RECS toolbox is not limited to storage models. It is designed to solve small-scale rational expectations models, with a focus on models with occasionally binding constraints. This implies that it can be used also to understand household production, saving and storing behavior in the presence of transaction costs and market imperfections (see Park, 2006, for a model of Chinese rural households).

To summarize, RECS provides a user-friendly environment to develop small dynamic, stochastic models in which agents' expectations of other agents' behavior, markets and policies matter. These situations are pervasive in developing countries' food markets, where the importance of food, and the risks implied by weather and price volatility, compel households to engage in sophisticated risk coping strategies (Fafchamps, 2003).

1.3 RECS toolbox

RECS is a Matlab solver for dynamic, stochastic, rational expectations equilibrium models. RECS stands for "Rational Expectations Complementarity Solver". This name emphasizes that RECS has been developed

specifically to solve models that include complementarity equations, also known as models with occasionally binding constraints.

Unless stated otherwise, all files in the RECS toolbox are licensed using the Expat license, a permissive free software license:

```
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Please see the software license for more information.[4]

RECS source code and development version is available at https://github.com/christophe-gouel/recs. Bug should be reported at https://github.com/christophe-gouel/RECS/issues.

2 Installation

Download RECS Toolbox zip archives are available at http://code.google.com/p/recs/.

Why is this archive 12 MB? Much of this size is due to an executable for Windows. The executable file includes a complete Python distribution necessary to parse RECS model files.

Dependencies

- Matlab R2009b or later.

[4] https://raw.github.com/christophe-gouel/RECS/master/LICENSE.txt.

- CompEcon toolbox.[5] RECS depends on the CompEcon toolbox for many programs (especially with respect to interpolation). Please follow CompEcon installation instructions and *do not forget* to create the mex files if you want your models solved in a reasonable time.

Optional dependencies

- Path solver for Matlab.[6] Path is the reference solver for mixed complementarity problems (see Section 5). Its installation is highly recommended if difficult complementarity problems need to be solved.
- MATLAB Optimization Toolbox. The solver `fsolve` can be used to solve both the equilibrium equations and the rational expectations equilibrium.
- Sundials Toolbox,[7] which provides a compiled Newton-Krylov solver for solving the rational expectations equilibrium.
- Matlab Statistics Toolbox. Useful to simulate models in which shocks follow other distributions than the normal.

Installation instructions

1. Download the latest RECS archive at `http://code.google.com/p/recs/` and unzip it into a folder, called here `recsfolder` (avoid folder names that include spaces, even for parent folders).
2. Install the CompEcon toolbox:
   (a) Download the CompEcon toolbox archive at `http://www4.ncsu.edu/~pfackler/compecon/`;
   (b) Unzip the archive into a folder, called here `compeconfolder`;
   (c) Add CompEcon to the Matlab path:
   `addpath('compeconfolder/CEtools','compeconfolder/CEdemos');`
   (d) Type `mexall` in Matlab prompt to create all CompEcon mex files.
3. (optional) Install other dependencies.
4. Add the RECS folder to the Matlab path: `addpath('recsfolder')`.
5. On Windows, you are all set. On other architectures, you will have to install some Python packages. see instructions below.
6. You can test your installation by running RECS demonstration files by typing `recsdemos`. You can also access RECS documentation in Matlab by typing `doc`.

[5] `http://www4.ncsu.edu/~pfackler/compecon/`.
[6] `http://pages.cs.wisc.edu/~ferris/path.html`.
[7] `https://computation.llnl.gov/casc/sundials/main.html`.

Install on Linux Python 2.7.X is required. On Debian/Ubuntu, type in a terminal:

```
sudo apt-get install python-yaml python-scipy python-sympy
```

Install on Mac In this case, you are on your own. You have to install

- Python 2.7.X.[8] Python is preinstalled on Mac, but is usually too old to be useful.
- PyYaml.[9]
- SymPy.[10]
- SciPy.[11]

One solution might be to install a scientific Python distribution such as EPD.[12]

Let me know whether or not it works.

3 A RECS tutorial (model: STO1)

This tutorial example enables a quick overview of RECS features. The example is the competitive storage model presented in Wright and Williams (1982). It serves as the benchmark model from which all the other storage models presented in this paper will be derived.

3.1 The standard rational expectations storage model

There are three risk-neutral, representative agents: a storer, a producer and a consumer, and one commodity. The consumer is simply represented by its demand function, assumed here to be isoelastic, $D(P) = P^{\alpha}$, where α is the demand elasticity.

The storer's activity is to transfer a commodity from one period to the next. Storing the quantity S_t from period t to period $t+1$ entails a purchasing cost, $P_t S_t$, and a storage cost, kS_t, with k the unit physical cost of storage. In addition, a share δ of the commodity deteriorates during storage. The benefits in period t are the proceeds from the sale of previous stocks: $(1-\delta)P_t S_{t-1}$. The storer follows a storage rule that arbitrages intertemporal profit opportunities:[13]

$$S_t \geq 0 \quad \perp \quad (1-\delta)\beta \,\mathrm{E}_t\left(P_{t+1}\right) - P_t - k \leq 0, \tag{1}$$

[8] http://www.python.org/download/.
[9] http://pyyaml.org/wiki/PyYAML.
[10] http://sympy.org.
[11] http://www.scipy.org/Download.
[12] http://www.enthought.com/.
[13] Complementarity conditions in what follows are written using the "perp" notation ($\perp$). This means that the expressions on either side of the sign are orthogonal. If one equation holds with strict inequality, the other must have an equality (see Section 5 for details on complementarity problems).

where E_t denotes the mathematical expectations operator conditional on information available at time t and $\beta = 1/(1+r)$ is the discount factor. This equation means that inventories are null when the marginal cost of storage is not covered by expected marginal benefits; for positive inventories, the arbitrage equation holds with equality. So this is a situation of a stabilizing speculation, the storer buys when prices are low and when he rationally expects that they will be higher later.

The representative producer makes his productive choice one period before bringing output to market. He puts in production in period t a level H_t for period $t+1$, but a multiplicative disturbance affects final production (e.g., a weather disturbance). The producer chooses the production level by solving the following maximization of expected profit:

$$\max_{\{H_{t+i}\}_{i=0}^{\infty}} \mathrm{E}_t \left\{ \sum_{i=0}^{\infty} \beta^i \left[P_{t+i} \varepsilon_{t+i} H_{t+i-1} - \Psi\left(H_{t+i}\right) \right] \right\}, \tag{2}$$

where $\Psi(H_t)$ is the cost of planning the production H_t and $H_t \varepsilon_{t+1}$ is the realized production level. ε_{t+1} is the realization of an i.i.d. stochastic process of mean 1 exogenous to the producer. This problem gives the following Euler equation:

$$\beta \, \mathrm{E}_t \left(P_{t+1} \varepsilon_{t+1} \right) = \Psi'\left(H_t\right). \tag{3}$$

The marginal cost is assumed to be isoelastic and an increasing function of planned production, as increasing production requires increasing the use of less fertile lands, $\Psi'(H_t) = h H_t^{\mu}$, where h is a scale parameter and μ is the inverted supply elasticity.

At the beginning of each period, three predetermined variables define the state of the model: S_{t-1}, H_{t-1} and ε_t. These three variables can be combined in one state variable, availability (A_t), the sum of production, and carry-over:

$$A_t = H_{t-1} \varepsilon_t + (1-\delta) S_{t-1}. \tag{4}$$

Market equilibrium can be written as

$$A_t = D\left(P_t\right) + S_t. \tag{5}$$

From the above equations, we see that a storage model is defined by three response/control variables $\{S_t, H_t, P_t\}$, and one state variable A_t, corresponding to three equilibrium equations (1), (3), and (5), and one transition equation (4).

3.2 Its representation in RECS

The RECS representation of this problem is very similar to its algebraic representation and is as follows (available in file `sto1.yaml`):

```
# STO1.yaml Model definition file for the competitive storage model
# Copyright (C) 2011-2012 Christophe Gouel
# Licensed under the Expat license, see LICENSE.txt
```

```
declarations:

  states: [A]

  controls: [S, H, P]

  expectations: [EP, EPe]

  shocks: [e]

  parameters: [k, delta, r, h, mu, elastD, elastS]


equations:

  arbitrage:

    - P+k-EP*(1-delta)/(1+r)     |    0 <= S <= inf
    - EPe/(1+r) = h*H^mu         | -inf <= H <= inf
    - A = P^elastD+S             | -inf <= P <= inf

  transition:

    - A = (1-delta)*S(-1)+H(-1)*e

  expectation:

    - EP  = P(1)
    - EPe = P(1)*e


calibration:

  parameters:

    k      :  0.06
    delta  :  0.02
    r      :  0.03
    elastS :  0.2
```

```
  h      : 1/(1+r)
  mu     : 1/elastS
  elastD : -0.2

steady_state:

  A : 1
  S : 0
  H : 1
  P : 1
```

Observe that, this file, in addition to the definition of variables and equations, contains the values necessary to calibrate the model.

3.3 Solving the storage model under Matlab

Once the Yaml file of a model has been written, which can been done using the Matlab editor, solving the model involves following under Matlab these 4 steps:

i. Create the model structure in Matlab.

ii. Define the approximation space.

iii. Define a first-guess for the solver.

iv. Solve for the rational expectations equilibrium.

We now detail these steps for the case of our example model.

3.3.1 Create the model structure in Matlab

So far, Matlab does not know anything about the Yaml file and, even if it did, although the file is easy to read and write for human, it is meaningless for Matlab. So we have to tell Matlab to use the Yaml file, and convert it to a file format suitable for Matlab. This is done by the function `recsmodelinit`, which creates a structure containing the model, its shocks, parameters and steady-state as follows

```
Mu                = 1;
sigma             = 0.05;

model = recsmodelinit('sto1.yaml',struct('Mu',Mu,'Sigma',sigma^2,'order',7));
```

```
Deterministic steady state (equal to first guess)
 State variables:
1

 Response variables:
0 1 1

 Expectations variables:
1 1
```

In creating the model structure, this function call also leads to calculation of the model deterministic steady state using as a first guess the values provided in the Yaml file.

3.3.2 Define the approximation space

Having defined the model, we need to define the domain over which it will be approximated and the precision of the approximation. The approximation space is defined by the function `recsinterpinit`, which requires three inputs: the number of points on the grid of approximation, the lower bound, and the upper bound. In this case, we choose 30 points for the grid, and bounds are defined relative to the steady-state value: half of it for the lower bound and 80% above it for the upper bound.

```
[interp,s] = recsinterpinit(30,model.sss/2,model.sss*1.8);
```

This function creates the structure `interp`, which contains all the information related to the approximation space, and the variable `s`, which is the grid of state variables.

3.3.3 Define a first-guess for the solver

The rational expectations solver has to be initialized from a good starting point to ensure convergence to the rational expectations equilibrium. This can be done automatically by calculating the perfect foresight solution on the grid point corresponding to the stochastic problem. Using the two structures created previously, this is done by the function `recsFirstGuess`:

```
[interp,x] = recsFirstGuess(interp,model,s,model.sss,model.xss,5);
```

3.3.4 Solve for the rational expectations equilibrium

Now we have all the elements required to solve the model: model definition, approximation space, and first guess. All these elements have to be passed to the function `recsSolveREE`:

```
[interp,x] = recsSolveREE(interp,model,s,x);
```

```
Successive approximation
  Iteration Residual
       1 2.4E-001
       2 4.2E-002
       3 3.2E-002
       4 2.7E-002
       5 1.7E-002
       6 6.6E-003
       7 1.9E-003
       8 5.1E-004
       9 1.3E-004
      10 3.2E-005
      11 8.1E-006
      12 2.0E-006
      13 5.0E-007
      14 1.2E-007
      15 2.5E-008
      16 1.6E-015
```

It outputs `interp`, the interpolation structure that has been updated to contain the approximation of the policy rules, and `x` the response variables on the grid.

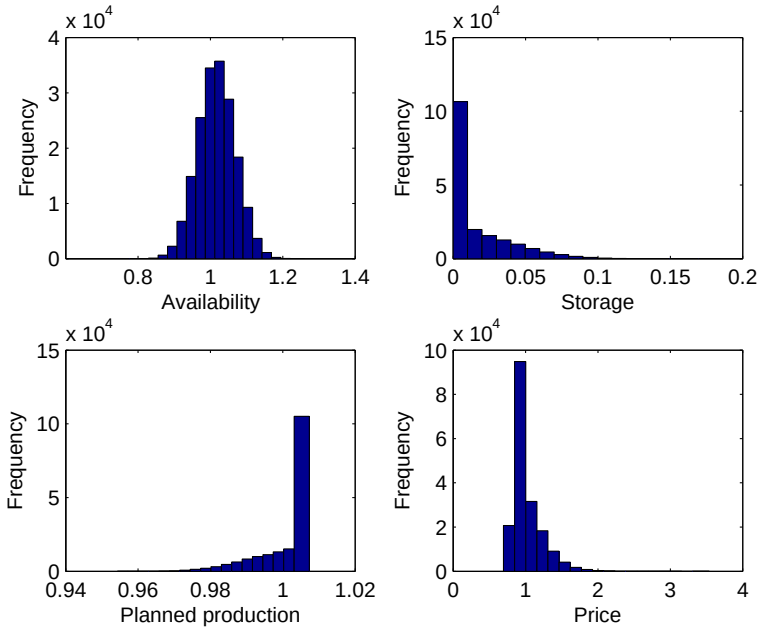
3.3.5 Analyze the results

Once the model is solved, it is possible to analyze its behavior. This is done below through a stochastic simulation that generates the main moments from the asymptotic distribution.

```
[~,~,~,~,stat] = recsSimul(model,interp,model.sss(ones(1000,1),:),200);
subplot(2,2,1)
xlabel('Availability')
ylabel('Frequency')
subplot(2,2,2)
xlabel('Storage')
ylabel('Frequency')
subplot(2,2,3)
xlabel('Planned production')
ylabel('Frequency')
subplot(2,2,4)
xlabel('Price')
ylabel('Frequency')
```

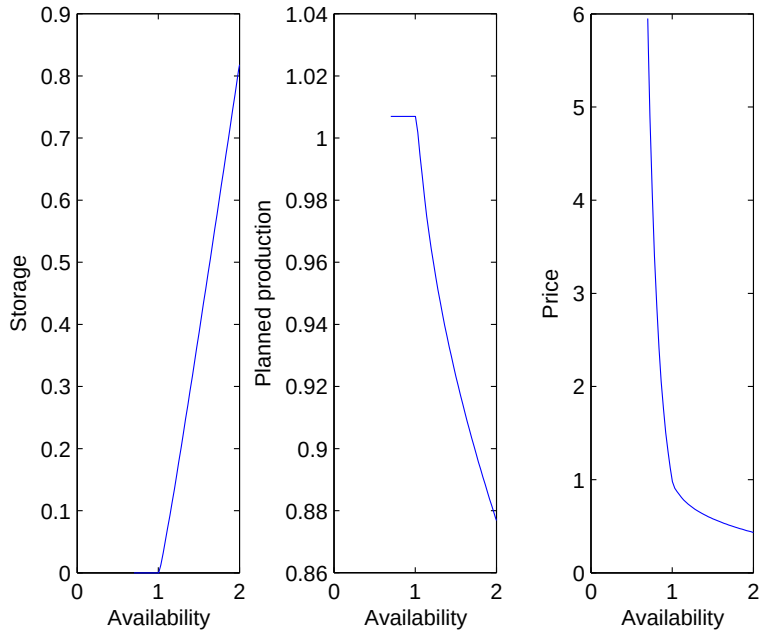
```
Statistics from simulated variables (excluding the first 20 observations):
```

```
Moments
    Mean      Std. Dev. Skewness  Kurtosis  Min       Max       %LB       %UB
    1.0167    0.0518    0.0190    2.9848    0.7767    1.2596

    0.0158    0.0221    1.6203    5.3890         0    0.1842   25.7104         0
    1.0010    0.0081   -1.4053    4.3248    0.9543    1.0074         0         0
    1.0147    0.2011    1.9070    8.1329    0.6953    3.5376         0         0

Correlation
    1.0000    0.8703   -0.8793   -0.9033
    0.8703    1.0000   -0.9982   -0.5825
   -0.8793   -0.9982    1.0000    0.5965
   -0.9033   -0.5825    0.5965    1.0000

Autocorrelation
    1         2         3         4         5
    0.2152    0.0438    0.0024   -0.0037   -0.0066

    0.2463    0.0536    0.0035   -0.0051   -0.0083
    0.2449    0.0532    0.0034   -0.0053   -0.0081
    0.1188    0.0188   -0.0009   -0.0033   -0.0043
```




```
figure
subplot(1,3,1)
```

```
plot(s,x(:,1))
xlabel('Availability')
ylabel('Storage')
subplot(1,3,2)
plot(s,x(:,2))
xlabel('Availability')
ylabel('Planned production')
subplot(1,3,3)
plot(s,x(:,3))
xlabel('Availability')
ylabel('Price')
```



4 Convention to define stochastic rational expectations problem in RECS

4.1 Rational expectation models

There are several ways to define rational expectations models. RECS adopts a controlled-process convention in which the values taken by control, or response, variables are decided at each period based on the values of state variables. The convention follows the framework proposed in Fackler (2005), and used also in Winschel and Krätzig (2010). A model can be defined by the following three equations, where time subscripts are implicit for current-period variables, and next-period variables are indicated with the $+$ subscript,

while previous-period variables are indicated with the $-$ subscript:

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f(s,x,z), \qquad \text{where } f : \mathbb{R}^{d+m+p} \to \mathbb{R}^m, \tag{6}$$

$$z = \mathrm{E}\left[h(s,x,e_+,s_+,x_+)\right], \qquad \text{where } h : \mathbb{R}^{d+m+q+d+m} \to \mathbb{R}^p, \tag{7}$$

$$s = g(s_-,x_-,e), \qquad \text{where } g : \mathbb{R}^{d+m+q} \to \mathbb{R}^d. \tag{8}$$

Variables have been partitioned into state variables, s, response variables, x, and shocks, e. Response variables can have lower and upper bounds, $\underline{x}$ and $\overline{x}$, which themselves can be functions of the state variables. Expectations variables, denoted by z, are also defined, because they are necessary for solving the model considering the implemented algorithms.

The first equation is the equilibrium equation. It characterizes the behavior of the response variables given state variables and expectations about the next period. For generality, it is expressed as a mixed complementarity problem (MCP, if you aren't familiar with the definition of MCP, see Section 5). In cases where response variables have no explicit lower and upper bounds, or have infinite ones, equation (6) simplifies to a traditional equation:

$$0 = f(s,x,z). \tag{9}$$

The second equation (7) defines the expectations. The last equation, (8), is the state transition equation, which defines how state variables are updated based on past responses, past states and contemporaneous shocks.

4.2 Restrictions imposed by the RECS convention

Distinction between state variables and other variables In many models, it is possible to simplify the state transition equation $s = g(s_-,x_-,e)$. For example, it is possible to have $s = e$ if some shocks are not serially correlated, or $s = x_-$ if the state is just a previous period response variables. In the latter case, one might be tempted to reduce the number of variables in the model by introducing the lag response variable directly in the equilibrium equation. This should not be done. A state variable corresponding to the lagged response variable or to the realized shock has to be created.

One consequence is that lags can only appear in state transition equations, (8), and in no other equations.

Lags and leads RECS only deals with lags and leads of one period. For a model with lags/leads of more periods, additional variables have to be included to reduce the number of periods.

In addition, leads can only appear in the equations defining expectations, (7). So no leads or lags should ever appear in the equilibrium equations.

Timing convention In RECS, the timing of each variable reflects when that variable is decided/determined. In particular, the RECS convention implies that state variables are determined by a transition equation that

includes shocks and so are always contemporaneous to shocks, even when shocks do not actually play a role in the transition equation. This convention implies that the timing of each variable depends on the way the model is written.

One illustration of the consequences of this convention is the timing of planned production in the competitive storage model presented as a tutorial (Section 3). Planned production, H_t, will only lead to actual production in $t+1$ and will be subject to shocks, ε_{t+1}, so it is tempting to use a $t+1$ indexing. However, since it is determined in t based on expectations of period $t+1$ price, it should be indexed t.

4.3 An example

As an example, consider the competitive storage model presented in tutorial. It is composed of four equations:

$$S_t : S_t \geq 0 \quad \perp \quad (1-\delta)\beta \,\mathrm{E}_t\left(P_{t+1}\right) - P_t - k \leq 0, \tag{10}$$

$$H_t : \beta \,\mathrm{E}_t\left(P_{t+1}\varepsilon_{t+1}\right) = \Psi'\left(H_t\right), \tag{11}$$

$$P_t : A_t = D\left(P_t\right) + S_t, \tag{12}$$

$$A_t : A_t = H_{t-1}\varepsilon_t + (1-\delta)S_{t-1}, \tag{13}$$

where S, H, P, and A respectively represent storage, planned production, price and availability.

There are three response variables: $x_t \equiv \{S_t, H_t, P_t\}$, and three corresponding equilibrium equations (10)–(12). These equations include two terms corresponding to expectations about period $t+1$: $z_t \equiv \{\mathrm{E}_t\left(P_{t+1}\right), \mathrm{E}_t\left(P_{t+1}\varepsilon_{t+1}\right)\}$.

There is one state variable, availability: $s_t \equiv \{A_t\}$, which is associated to the transition equation (13). It would have been perfectly legitimate to define the model with more state variables, such as production and stocks, since availability is the sum of both, and this would not prevent the solver from finding the solution, however, it is generally not a good idea. Since the solution methods implemented in RECS suffer from the curse of dimensionality, it is important where possible to combine predetermined variables to reduce the number of state variables.

Corresponding to RECS convention, this model is defined by the following three functions $\{f, h, g\}$:

$$f \equiv \left\{ \begin{array}{l} P_t + k - z_{1,t}(1-\delta)\beta \\ \beta z_{2,t} - \Psi'\left(H_t\right) \\ A_t - D\left(P_t\right) - S_t \end{array} \right\} \tag{14}$$

$$h \equiv \left\{ \begin{array}{l} P_{t+1} \\ P_{t+1}\varepsilon_{t+1} \end{array} \right\} \tag{15}$$

$$g \equiv \{H_{t-1}\varepsilon_t + (1-\delta)S_{t-1}\} \tag{16}$$

4.4 Solving a rational expectations model

What makes solving a rational expectations model complicated is that the equation (7), defining the expectations, is not a traditional algebraic equation. It is an equation that expresses the consistency between agents' expectations, their information set, and realized outcomes.

One way to bring this problem back to a traditional equation is to find an approximation or an algebraic representation of the expectations terms. For example, if it is possible to find an approximation of the relationship between expectations and current-period state variables (i.e., the parameterized expectations approach in den Haan and Marcet, 1990), the equilibrium equation can be simplified to

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f\left(s, x, \mathscr{Z}\left(s, c_z\right)\right), \tag{17}$$

where $\mathscr{Z}\left(s, c_z\right)$ approximates the expectations z as a function of s, and c_z are the coefficients characterizing this approximation. Using this approximation, equation (17) can be solved for x with any MCP solvers. So the crux of solving rational expectations model is to find a good way to approximate the expectations.

The RECS solver implements various methods to solve rational expectations models and to find approximation for the expectations terms. See Section 11 for a sketch of the algorithm.

5 Introduction to mixed complementarity problems

To be able to reliably solve models with occasionally binding constraints, all equilibrium equations should be represented in RECS as mixed complementarity problems (MCP). Here, we present a short introduction to this kind of problems. For more information, see Rutherford (1995), and Ferris and Pang (1997).

5.1 Definition of a mixed complementarity problem

Complementarity problems can be seen as extensions of square systems of nonlinear equations that incorporate a mixture of equations and inequalities. Many economic problems can be expressed as complementarity problems. An MCP is defined as follows (adapted from Munson, 2002):

Definition. (Mixed Complementarity Problem) Given a continuously differentiable function $F : \mathbb{R}^n \to \mathbb{R}^n$, and lower and upper bounds

$$l \in \{\mathbb{R} \cup \{-\infty\}\}^n,$$
$$u \in \{\mathbb{R} \cup \{+\infty\}\}^n.$$

The mixed complementarity problem $l \leq x \leq u \perp F(x)$ is to find a $x \in \mathbb{R}^n$ such that one of the following

holds for each $i \in \{1,\ldots,n\}$:

$$F_i(x) = 0 \text{ and } l_i \leq x_i \leq u_i, \tag{18}$$

$$F_i(x) > 0 \text{ and } x_i = l_i, \tag{19}$$

$$F_i(x) < 0 \text{ and } x_i = u_i. \tag{20}$$

To summarize, an MCP problem associates each variable, x_i, to a lower bound, l_i, an upper bound, u_i, and an equation, $F_i(x)$. The solution is such that if x_i is between its bounds then $F_i(x) = 0$. If x_i is equal to its lower (upper) bound then $F_i(x)$ is positive (negative).

This format encompasses several cases. In particular, it is easy to see that with infinite lower and upper bounds, solving an MCP problem is equivalent to solving a square system of nonlinear equations:

$$-\infty \leq x \leq +\infty \quad \perp \quad F(x) \quad \Leftrightarrow \quad F(x) = 0.$$

5.2 A simple example of mixed complementarity (model: CS1)

We consider here the traditional consumption/saving problem with borrowing constraint (Deaton, 1991). This problem is solved as a demonstration, see problem CS1 in the section Demos in the RECS documentation. A consumer with utility $\sum_{t=0}^{\infty} u(C_t)/(1+\delta)^t$ has a stochastic income, Y_t and has to choose each period how much to consume and how much to save. He is prevented from borrowing, but can save at the rate r. Without the borrowing constraint, his problem consists of choosing its consumption C_t such that it satisfies the standard Euler equation:

$$u'(C_t) = \frac{1+r}{1+\delta} \mathrm{E}_t \left[u'(C_{t+1}) \right].$$

The borrowing constraint implies the inequality $C_t \leq X_t$, where X_t is the cash on hand available in period t and defined as

$$X_t = (1+r)(X_{t-1} - C_{t-1}) + Y_t.$$

When the constraint is binding (i.e., $C_t = X_t$), the Euler equation no longer holds. The consumer would like to borrow but cannot, so its marginal utility of immediate consumption is higher than its discounted marginal utility over future consumption:

$$u'(C_t) \geq \frac{1+r}{1+\delta} \mathrm{E}_t \left[u'(C_{t+1}) \right].$$

Since the Euler equation holds when consumption is not constrained, this behavior can be summarized by the following complementarity equation

$$C_t \leq X_t \quad \perp \quad \frac{1+r}{1+\delta} \mathrm{E}_t \left[u'(C_{t+1}) \right] \leq u'(C_t).$$

5.3 When do complementarity problems arise?

In addition to encompassing nonlinear systems of equations, complementarity problems appear naturally in economics. Although not exhaustive, we present here few situations that lead to complementarity problems:

- **Karush-Kuhn-Tucker conditions** of a constrained nonlinear program. The first-order conditions of the following nonlinear programming problem $\min_x f(x)$ subject to $g(x) \leq 0$ and $l \leq x \leq u$ can be written as a system of complementarity equations:

$$l \leq x \leq u \quad \perp \quad f'(x) - \lambda g'(x), \tag{21}$$
$$\lambda \geq 0 \quad \perp \quad -g(x) \geq 0, \tag{22}$$

  where λ is the Lagrange multiplier on the first inequality constraint.

- Natural representation of **regime-switching behavior**. Let us consider two examples.
  - A variable y that is determined as a function of another variable x as long as y is superior to a lower bound a. Put simply: $y = \max[a, \lambda(x)]$. In MCP, we would write this as $y \geq a \perp y \geq \lambda(x)$.
  - A system of intervention prices in which a public stock is accumulated when prices decrease below the intervention price and sold when they exceed the intervention price (see also the demonstration problem in Section 12.1) can be represented as $S \geq 0 \perp P - P^I \geq 0$, where S, P and P^I, respectively, are the public storage, the price and the intervention price.

- **A Walrasian equilibrium** can be formulated as a complementarity problem (Mathiesen, 1987) by associating three sets of variables to three sets of inequalities: non-negative levels of activity are associated to zero-profit conditions, non-negative prices are associated to market clearance, and income levels are associated to income balance.

5.4 Conventions of notations adopted for representing complementarity problems

To solve complementarity problems, RECS uses several solvers. The convention adopted in most MCP solvers and used by RECS is the one used above in the MCP definition: superior or equal inequalities are associated with the lower bounds and inferior or equal inequalities are associated with the upper bounds. So, when defining your model, take care to respect this convention.

In addition, in Yaml files inequalities should not be written explicitly. As an example, let's consider these 5

MCP equations:

$$F(x) = 0, \tag{23}$$

$$y \geq a \quad \perp \quad G(y) \geq 0, \tag{24}$$

$$z \leq b \quad \perp \quad H(z) \leq 0, \tag{25}$$

$$a \leq k \leq b \quad \perp \quad J(k), \tag{26}$$

$$l \geq a \quad \perp \quad M(l) \leq 0. \tag{27}$$

They would be written in a Yaml file as

```
- F(x)=0 | -inf<=x<=inf
- G(y)   |    a<=y<=inf
- H(z)   | -inf<=z<=b
- J(k)   |    a<=k<=b
- -M(l)  |    a<=l<=inf
```

Note that it is necessary to associate lower and upper bounds with every variables, and the "perp" symbol ($\perp$) is substituted by the vertical bar (|), which is much easier to find on a keyboard. So if there are no finite bounds, one has to associate infinite bounds, as for equation (23). The last equation, (27), does not respect the convention that associates lower bounds on variables with superior or equal inequality for equations, so, when writing it in the Yaml file, the sign of the equation needs to be reversed.

5.5 Example: MCP representation of a price-band program (model: STO4)

Price-band policies constitute an excellent application for MCP modeling. Although this policy is simple to express in words, it needs some reformulation and attention to be expressed as an MCP problem.

This problem is analyzed in depth in Miranda and Helmberger (1988). This model is an extension of the competitive storage model in which government intervenes by defending a price band through purchase and sale of public storage. Here, we present an extension of Miranda and Helmberger (1988) in which there is a capacity constraint on the public stock level.

With a capacity constraint on public stock, $\overline{S^{G}}$, the behavior of a price-band obeys some simple principles. When the price is above the floor price, P^{F}, there is no accumulation of public stock, S^{G}:

$$P_t > P^{F} \Rightarrow \Delta S_t^{G} \leq 0. \tag{28}$$

Likewise, when the price is below the ceiling price, P^{C}, government does not sell stock:

$$P_t < P^{C} \Rightarrow \Delta S_t^{G} \geq 0. \tag{29}$$

When the capacity constraint is reached, the floor price is not defended and the price can decrease below it:

$$P_t < P^{\mathrm{F}} \Rightarrow \Delta S_t^{\mathrm{G}} = \overline{S}^{\mathrm{G}} - S_{t-1}^{\mathrm{G}}. \tag{30}$$

Finally, the ceiling price is not defended when the public stock is exhausted:

$$P_t > P^{\mathrm{C}} \Rightarrow \Delta S_t^{\mathrm{G}} = -S_{t-1}^{\mathrm{G}}. \tag{31}$$

To express these four conditions as complementarity equations, we need to introduce two variables: $\Delta S^{\mathrm{G}+}$ and $\Delta S^{\mathrm{G}-}$, which refer to increases and decreases in public stock. Both are positive and bounded from above. The increase in public stock is bounded from above by the remaining storage capacity, and the decrease in public stock by the level of existing stocks. To defend the price-band, public stocks are managed by the four conditions (28)–(31), which can be restated as two mixed complementarity equations:

$$0 \leq \Delta S_t^{\mathrm{G}+} \leq \overline{S}^{\mathrm{G}} - S_{t-1}^{\mathrm{G}} \quad \perp \quad P_t - P^{\mathrm{F}}, \tag{32}$$

$$0 \leq \Delta S_t^{\mathrm{G}-} \leq S_{t-1}^{\mathrm{G}} \quad \perp \quad P^{\mathrm{C}} - P_t. \tag{33}$$

Equation (32) means that public stocks increase to prevent the price from decreasing below the floor price, P^{F}. The floor is defended until public stocks reach the limit $\overline{S}^{\mathrm{G}}$. Equation (33) governs the decrease in public stocks. They decrease to prevent the price from rising above the ceiling price, P^{C}. The release of stocks is constrained by the existing level of stocks S_{t-1}^{G}.

Market equilibrium and public stock transition are then defined by[14]

$$A_t = D(P_t) + S_t + \Delta S_t^{\mathrm{G}+} - \Delta S_t^{\mathrm{G}-}, \tag{34}$$

$$S_t^{\mathrm{G}} = S_{t-1}^{\mathrm{G}} + \Delta S_t^{\mathrm{G}+} - \Delta S_t^{\mathrm{G}-}. \tag{35}$$

The recursive equilibrium under a price-band program is defined by the equilibrium equations (1), (3) and (32)–(34), and the transition equations (4) and (35).

6 Setting Up a Rational Expectations Problem

6.1 Structure of RECS model files

6.1.1 Structure of a rational expectation models

Before starting to write the model file, you need to organize your equations as described in Section 4. This means that you need to identify and separate in your model the following three group of equations:

Equilibrium equations

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f(s,x,z). \tag{36}$$

[14] Note that here, for simplicity, this model does not assume any depreciation during storage.

Expectations definition

$$z = \mathrm{E}\left[h(s,x,e_+,s_+,x_+)\right]. \tag{37}$$

Transition equations

$$s = g(s_-,x_-,e). \tag{38}$$

When defining your model, it is important to try to minimize the number of state variables. Currently, RECS relies for interpolation on grids constructed with tensor products, so the dimension of the problem increases exponentially with the number of state variables. This implies that RECS should be used only to solve small-scale models: more than three or four state variables may make the problems too large to handle. One way of reducing the problem size is, whenever possible (as in the competitive storage model), to sum together the predetermined variables that can be summed.

6.1.2 Structure of RECS model files

An RECS model can be written in a way that is quite similar to the original mathematical notations (as is proposed in most algebraic modeling languages). The model file that must be created is called a Yaml file (because it is written in YAML[15] and has the `.yaml` extension). A Yaml file is easily readable by humans, but not by Matlab. So the file needs to be processed for Matlab to be able to read it. This is done by the function `recsmodelinit`, which uses a Python preprocessor, `dolo-recs`, to do the job.

RECS Yaml files require three basic components, written successively:

1. `declarations` – The block `declarations` contains the declaration of all the variables, shocks and parameters that are used in the model. Inside this block, there are five sub-blocks: `states`, `controls`, `expectations`, `shocks`, and `parameters` with corresponding elements declarations.
2. `equations` – The block `equations` declares the model's equations.
3. `calibration` – The block `calibration` provides numerical values for parameters and a first-guess for the deterministic steady-state.

Yaml file structure To illustrate a complete Yaml file let us consider how to write the competitive storage model in Yaml (available in file `sto1.yaml`):

```
# STO1.yaml Model definition file for the competitive storage model
# Copyright (C) 2011-2012 Christophe Gouel
# Licensed under the Expat license, see LICENSE.txt

declarations:
```

[15] http://yaml.org.

```
  states: [A]

  controls: [S, H, P]

  expectations: [EP, EPe]

  shocks: [e]

  parameters: [k, delta, r, h, mu, elastD, elastS]


equations:

  arbitrage:

    - P+k-EP*(1-delta)/(1+r)     |    0 <= S <= inf
    - EPe/(1+r) = h*H^mu         | -inf <= H <= inf
    - A = P^elastD+S             | -inf <= P <= inf

  transition:

    - A = (1-delta)*S(-1)+H(-1)*e

  expectation:

    - EP  = P(1)
    - EPe = P(1)*e


calibration:

  parameters:

    k      :  0.06
    delta  :  0.02
    r      :  0.03
    elastS :  0.2
    h      : 1/(1+r)
    mu     : 1/elastS
```

```
  elastD : -0.2

steady_state:

  A : 1
  S : 0
  H : 1
  P : 1
```

Yaml files can be written with any text editor, including Matlab editor.

Yaml syntax conventions For the file to be readable, it is necessary to respect some syntax conventions:

- **Association equilibrium equations/control variables/bounds:** Each equilibrium equation must be associated with a control variable. It really matters for equations with complementarity constraints. If the equation is not a complementarity equation, the precise association with a control variable does not matter. The equation and its associated variable are separated by the symbol |. Control variables must be associated with bounds, which can be infinite.
- **Indentation:** Inside each block elements of the same scope have to be indented using the same number of spaces (do not use tabs). For the above example, inside the `declarations` block, `states`, `controls`, `expectations`, `shocks`, and `parameters` are indented to the same level.
- Each equation starts with a dash and a space ("`- `"), not to be confused with a minus sign. To avoid confusion, it is possible instead to use two points and a space ("`. .  `"), or two dashes and a space ("`--` ").
- **Comments:** Comments are introduced by the character `#`.
- **Lead/Lag:** Lead variables are indicated `X(1)` and lag variables `X(-1)`.
- **Timing convention:**
  - Transition equations are written by defining the new state variable at current time as a function of lagged response and state variables: $s_t = g(s_{t-1}, x_{t-1}, e_t)$.
  - Even if expectations are defined by $z_t = \mathrm{E}_t\left[h(s_t, x_t, e_{t+1}, s_{t+1}, x_{t+1})\right]$, when writing them in a Yaml file the shocks are not indicated with a lead. This can be seen in the example above where $P_{t+1}\varepsilon_{t+1}$ is written as `EPe = P(1)*e`.
- Do not use `lambda` as a variable/parameter name, this is a restricted word.
- Write equations in the same order as the order of variables declaration.
- Yaml files are case sensitive.

6.2 Defining the model structure

Now the model should be written in a Yaml file, however Matlab does not know anything about the Yaml file and, even if it did, although the file is easy for humans to read and write, it means nothing to Matlab. So we have to tell Matlab to use the Yaml file and to convert it to a format suitable for Matlab. Also, we have to provide additional information about the structure of shocks.

6.2.1 Convert the Yaml file and create the model structure

This task is done by the function `recsmodelinit` that calls a Python preprocessor, to convert the model described in a Yaml file to a file readable by Matlab and RECS programs. In the conversion, it calculates the analytic representation of all partial derivatives.

A simple call to `recsmodelinit` takes the following form:

```
model = recsmodelinit('file.yaml');
```

This call does two things:

- It converts `file.yaml` to `filemodel.m`, which contains the model definition in a Matlab readable form but also all the derivatives of the equations, plus some additional information such as the parameters values for calibration or a first guess for the steady state.
- It creates in Matlab workspace the structure `model` with two fields: the function name, `func` equal to `@filemodel`, and the parameters values, `params`, if these latter have been provided in the Yaml file.

6.2.2 Shocks with a Gaussian distribution

If your shocks follow a Gaussian distribution, you can also define their structure when calling `recsmodelinit`. It requires defining a structure with three fields characterizing the distribution mean, variance/covariance matrix, and order of approximation, with the call

```
model = recsmodelinit('file.yaml',...
                      struct('Mu',Mu,'Sigma',Sigma,'order',order));
```

Here `Mu` is a size-q vector of the distribution mean, `Sigma` is a q-by-q positive definite variance/covariance matrix, and `order` is a scalar or a size-q vector equal to the number of nodes for each shock variable.

This function call creates at least three additional fields in the model structure: `e` a matrix of the nodes for the shocks discretization; `w` the vector of associated probabilities; and `funrand` an anonymous function that can generate random numbers corresponding to the underlying distribution.

If a first-guess for the deterministic steady state has been provided, `recsmodelinit` attempts also to find the deterministic steady state of the problem. If it finds it, it is displayed on screen and output as three fields in the model structure: `sss`, `xss`, and `zss` for, respectively, the steady-state values of state, response and expectations variables.

6.2.3 An example

Let us consider the example of the competitive storage model. The complete function call in sto1.m is:

```
Mu                = 1;
sigma             = 0.05;

model = recsmodelinit('sto1.yaml',struct('Mu',Mu,'Sigma',sigma^2,'order',7));
```

```
Deterministic steady state (equal to first guess)
 State variables:
1

 Response variables:
0 1 1

 Expectations variables:
1 1
```

It is important to notice that variables names are not displayed here and will not be displayed in subsequent steps. Variable names are used only in the symbolic model definition in the Yaml file. Once the Yaml file has been processed, variables are merely ordered based on their original order in the Yaml file. In this case, it means that, in the steady-state results above, for the response variables the first number is storage, the second is planned production, and the third is price.

The model structure has the following fields:

```
disp(model)
```

```
       func: @sto1model
     params: [0.0600 0.0200 0.0300 0.9709 5 -0.2000 0.2000]
          e: [7x1 double]
          w: [7x1 double]
    funrand: @(nrep)Mu(ones(nrep,1),:)+randn(nrep,q)*R
        sss: 1
        xss: [0 1 1]
        zss: [1 1.0000]
```

6.3 Defining the interpolation structure

6.3.1 Define the interpolation structure

Following the previous two steps to completely define the problem, it remains only to define the domain over which it will be approximated and the precision of the approximation.

Create the interpolation structure This task is done by the function `recsinterpinit`, which requires at least three inputs: the number of points on the grid of approximation, the lower bounds and the upper bounds of the grid.

The structure of the call to `recsinterpinit` is then

```
[interp,s] = recsinterpinit(n,smin,smax,method);
```

The inputs are as follows: `n` designates the order of approximation (if it is a scalar, the same order is applied for all dimensions), `smin` and `smax` are size-d vectors of left and right endpoints of the state space, and the (optional) string `method` defines the interpolation method, either spline (`'spli'`, default), or Chebyshev polynomials (`'cheb'`).

This function call returns two variables: the structure `interp`, which defines the interpolation structure, and the matrix `s`, which represents the state variables on the grid.

6.3.2 An example

We now define the interpolation structure for the competitive storage example. Using the model structure defined in Section 6.2, we choose bounds for availability 50% below and 80% above the steady-state value (`model.sss`). Using 30 nodes and spline, the function call is:

```
[interp,s] = recsinterpinit(30,model.sss/2,model.sss*1.8);
```

The interpolation structure has the following fields:

```
disp(interp)
```

```
    fspace: [1x1 struct]
       Phi: [1x1 struct]
```

7 Steady state

In contrast to pertubation approaches, the deterministic steady state does not play an important role in finding the rational expectations equilibrium with collocation methods, and thus with RECS. However, finding the deterministic steady state proved very useful in the overall model building for

- Calibration;
- Definition of the approximation space;
- Calculation of a first guess using the corresponding perfect foresight problem;
- Checking model structure.

7.1 Steady state definition

The deterministic steady state is the state reached in the absence of shocks and ignoring future shocks. Following the convention adopted in RECS (see Section 4), the deterministic steady state is the set $\{s,x,z\}$ of state, response and expectations variables that solves the following system of equations

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f(s,x,z), \tag{39}$$

$$z = h(s,x,\mathrm{E}\,(e)\,,s,x), \tag{40}$$

$$s = g(s,x,\mathrm{E}\,(e)). \tag{41}$$

7.2 Finding the steady state with RECS

Automatically when initializing model structure When writing a model file (see Section 6.1, it is possible, at the end of the Yaml file in the `calibration` block, to define an initial guess for finding the steady state. When the model structure is created by `recsmodelinit`, if the definition of the shocks is provided to `recsmodelinit`, a Newton-type solver will attempt to find the steady state starting from the initial guess provided in the model file. If a steady state is found, it is displayed in Matlab command window.

Manually Otherwise, the steady state can be found manually by feeding the function `recsSS` with the model and an initial guess for the steady state.

Both approaches rely on a Newton-type solver to find the steady state.

7.3 Uses of the deterministic steady state with RECS

Calibration Compared to the stochastic rational expectations problem, the deterministic steady state is easy to find. It does not even require the definition of an interpolation structure. Since in practice it is often close to the long-run average values of the stochastic problem, the steady state is useful for calibrating the model so that, on the asymptotic distribution, the model can reproduce the desired long-run average (see Section 12.2).

Define the approximation space As the values of the state variables in the stochastic problem are often located around the deterministic steady state, the steady state serves as a good reference point around which the state space can be define. See Section 6.3 for more.

First guess calculation for the stochastic problem The rational expectations solver requires a good first guess to ensure convergence to the solution. RECS proposes to provide a first guess by calculating the perfect foresight problem corresponding to the stochastic problem. The perfect foresight problem assumes that the model converges in the long-run to its deterministic steady state. See next section for more.

Checking model structure Before solving the stochastic problem, even if there are no motives for using the steady state, it is good practice to find it in order to ensure that the model written in the Yaml file behaves as expected in this simple case.

8 First guess for the stochastic problem

The solution methods used in RECS to find the rational expectations equilibrium of a problem all rely on nonlinear equation solvers that allow convergence only if the starting point is not too far from the solution. Hence, the importance of a good first guess.

8.1 The perfect foresight problem as a first guess

RECS can attempt to calculate for you a first guess that is good in most situations. It does it by calculating the perfect foresight solution of the deterministic problem in which the shocks in the stochastic problem have been substituted by their expectations:

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f(s,x,z), \tag{42}$$

$$z = h(s,x,\mathrm{E}(e),s_+,x_+), \tag{43}$$

$$s = g(s_-,x_-,\mathrm{E}(e)). \tag{44}$$

The solver for perfect foresight problems (function `recsSolveDeterministicPb`) assumes that the problem converges to its deterministic steady state after T periods (by default T=50). The perfect foresight problem is solved for each grid point and the resultant behavior is used as a first guess for the stochastic problem. This is done by the following call:

```
[interp,x] = recsFirstGuess(interp,model,s,sss,xss);
```

This function updates the interpolation structure, `interp`, with the solution of the perfect foresight problem and outputs the value of the response variables on the grid, `x`.

Solving time As the perfect foresight problem is solved for each point of the grid and for T periods, this step can take some time. In many cases, it should be expected that it may take more time to find a first guess through the perfect foresight solution than to solve the stochastic problem from this first guess.

How good is this first guess? It depends on the model and the solution horizon. For most models, if the state space has been properly defined (i.e., neither too small nor too large), this first guess is good enough to allow the stochastic solver to converge.

The quality of the perfect foresight solution as a first guess depends also on the model's nonlinearity. For models with behavior close to linear, the first guess can be extremely good. For example, in the stochastic growth model the first guess leads to an initial deviation from rational expectations of 1E-5. So after a few iterations, the solver converges to the solution. However, in the stochastic growth model with irreversible investment, which is much more nonlinear, the residual when starting the stochastic solver from the first guess is 1E-2. This is sufficient to achieve convergence, but it is still far from the true solution.

8.2 User-provided first guess

It is not necessary to use the deterministic problem as a first guess. A first guess can be provided directly by the user. In this case there is no need to call a function, but only to define a n-by-m matrix `x` that provides the values of the m response variables for the n points on the grid of state variable. The matrix `x` should then be supplied to `recsSolveREE` (see next section for more information).

9 Solve the rational expectations problem

9.1 The function `recsSolveREE`

Once the model, the interpolation structure and a first guess solution have been defined, it is possible to attempt to find the rational expectations equilibrium of the model. This is achieved by the function `recsSolveREE`.

This is done by the following call

```
[interp,x] = recsSolveREE(interp,model,s,x,options);
```

This function call returns the interpolation structure `interp` with new or updated fields. Following a successful run of `recsSolveREE`, the fields `cx` and `cz` are created or updated in `interp`. They represent the response and expectations variables interpolation coefficients, respectively. These coefficients are sufficient to simulate the model. The second output is the matrix `x` of response variables on the grid.

The `options` structure defines the methods used to find the rational expectations equilibrium, as well as other aspects of the solution process.

9.2 An example

First guess from the corresponding deterministic problem If the first guess is generated by the function `recsFirstGuess`, `interp` already includes the fields of coefficients `cx` and `cz` corresponding to this first

guess, so it is not necessary to supply the value of the response variables on the grid. For the competitive storage model, finding the first guess and solving the model involves the following call:

```
[interp,x] = recsFirstGuess(interp,model,s,model.sss,model.xss,5);
[interp,x] = recsSolveREE(interp,model,s);
```

```
Successive approximation
  Iteration Residual
       1 2.4E-001
       2 4.2E-002
       3 3.2E-002
       4 2.7E-002
       5 1.7E-002
       6 6.6E-003
       7 1.9E-003
       8 5.1E-004
       9 1.3E-004
      10 3.2E-005
      11 8.1E-006
      12 2.0E-006
      13 5.0E-007
      14 1.2E-007
      15 2.5E-008
      16 1.7E-015
```

User-provided first guess If the first guess is provided by the user, there is no need to include the fields `cx` or `cz` in `interp`. The user needs only to provide an n-by-m matrix `x` that contains the values of the m response variables for the n points on the grid of state variable.

Our example is based on a simple first guess: storage is zero below steady-state availability and 70% of availability in excess of steady state, planned production is equal to its steady-state value and price is given by the inverse demand function assuming that demand is equal to availability

```
elastD     = model.params(6);
x          = [max(0,0.7*s-model.sss) repmat(model.xss(2),size(s,1),1) s.^(1/elastD)];
[interp,x] = recsSolveREE(interp,model,s,x);
```

```
Successive approximation
  Iteration Residual
       1 2.1E+000
       2 8.6E-001
       3 3.6E-001
```

```
 4 1.5E-001
 5 5.4E-002
 6 1.8E-002
 7 5.0E-003
 8 1.3E-003
 9 3.4E-004
10 8.5E-005
11 2.1E-005
12 5.3E-006
13 1.3E-006
14 3.2E-007
15 7.9E-008
16 2.5E-015
```

With this simple model, even a naive first guess allows the solver to find the solution, however it significantly decreases the precision of the first iteration.

10 Stochastic simulations

10.1 Running a simulation

Once the rational expectations equilibrium has been found, it is possible to simulate the model using the function `recsSimul` with the following call:

```
[ssim,xsim] = recsSimul(model,interp,s0,nper);
```

where `model` and `interp` have been defined previously as the model and the interpolation structure. `s0` is the initial state and `nper` is the number of simulation periods.

`s0` is not necessarily a unique vector of the state variables. It is possible to provide a matrix of initial states on the basis of which simulations can be run for `nper` periods. Even starting from a unique state, using this feature will speed up the simulations.

Shocks The shocks used in the simulation can be provided as the fifth input to the function:

```
[ssim,xsim] = recsSimul(model,interp,s0,nper,shocks);
```

However, by default, `recsSimul` uses the function `funrand` defined in the model structure (see Section 6.2) to draw random shocks. If this function is not provided random draws are made from the shock discretization, using the associated probabilities.

To reproduce previously run results, it is necessary to reset the random number generator using the Matlab function `reset`.

Asymptotic statistics If in the options the field `stat` is set to 1, or if five arguments are required as output of `recsSimul`, then some statistics over the asymptotic distribution are calculated. The first 20 observations are discarded. The statistics calculated are the mean, standard deviation, skewness, kurtosis, minimum, maximum, percentage of time spent at the lower and upper bounds, correlation matrix, and the five first-order autocorrelation coefficients. In addition, `recsSimul` draws the histograms of the variables distribution are drawn.

These statistics are available as a structure in the fifth output of `recsSimul`:

```
[ssim,xsim,esim,fsim,stat] = recsSimul(model,interp,s0,nper);
```

Since RECS does not retain the variable names, when displaying the statistics, the variables are organized as follows: first state variables, followed by response variables, both of which follow the order of their definition in the Yaml file.

10.2 Choice of simulation techniques

There are two main approaches to simulate the model once an approximated rational expectations equilibrium has been found.

Using approximated decision rule The first, and most common, method consists of simulating the model by applying the approximated decision rules recursively.

Starting from a known s_1, for $t = 1 : T$

$$x_t = \mathscr{X}(s_t, c_x) \tag{45}$$

Draw a shock realization e_{t+1} from its distribution and update the state variable:

$$s_{t+1} = g(s_t, x_t, e_{t+1}) \tag{46}$$

This is the default simulation method. It is chosen in the options structure by setting the field `simulmethod` to `interpolation`.

Solve equilibrium equations using approximated expectations The second main method uses the function being approximated (decisions rules, conditional expectations, or expectations function) to approximate next-period expectations and solve the equilibrium equations to find the current decisions (in a time-iteration approach). For example, if approximated expectations ($z \approx \mathscr{Z}(s, c_z)$) are used to replace conditional expectations in the equilibrium equation, the algorithm runs as follows:

Starting from a known s_1, for $t = 1 : T$, find x_t by solving the following MCP equation using as first guess $x_t = \mathscr{X}(s_t, c_x)$

$$\underline{x}(s_t) \leq x_t \leq \overline{x}(s_t) \quad \perp \quad f(s_t, x_t, \mathscr{Z}(s_t, c_z)) \tag{47}$$

Draw a shock realization e_{t+1} from its distribution and update the state variable:

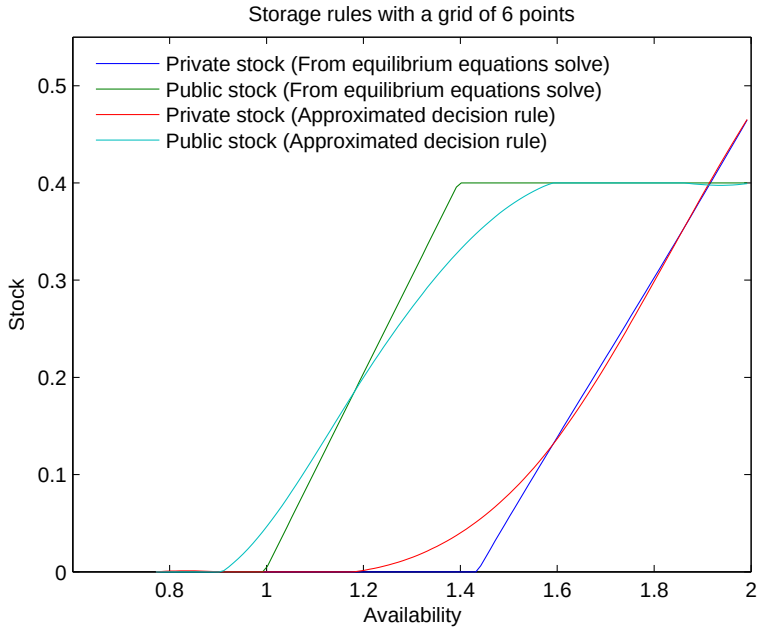
$$s_{t+1} = g(s_t, x_t, e_{t+1}) \tag{48}$$

This simulation method is chosen in the options structure by setting the field `simulmethod` to `solve`.

The two approaches differ in their speed and precision. Simulating a model through recursive application of approximated decision rules is very fast because it does not require solving nonlinear equations. The second approach is slower because each period requires that a system of complementarity equations be solved. However, its level of precision is much higher, because the approximation is used only for the expectations of next-period conditions. The precision gain is even more important when decision rules have kinks, which makes them difficult to approximate. Wright and Williams (1984), who first proposed the parameterized expectations algorithm, suggest using the second method to simulate the storage model.

When does this distinction matter? Most of the time, the default approach of recursively applying the approximated decision rules should be used. Simulating the model by solving the equations should be considered in the following two cases:

- If the model has been solved with low precision, using the approximated decision rules can lead to large errors, while simulating the model by solving the equilibrium equations using only the approximation in expectations will yield a more precise solution. This applies to the following example taken from Section 12.1: a price-floor policy backed by public storage. If the problem is approximated by a spline with a small grid of 6 points, it is not possible to expect a good approximation of the highly nonlinear storage rules. However, if the approximation is used only in expectations, it yields a very precise solution:



- If you are interested in the percentage of time spent in various regimes, solving the equilibrium equations is the only way to achieve a precise estimate. Even when solved with high precision, a simulation using the approximated decision rules provides limited precision regarding the percentage of time spent in each regime. Indeed, approximations using spline or Chebyshev polynomials fluctuate around the exact value between collocation nodes. So between two nodes at which the model should be at a bound, the approximated decision rule can yield a result very close to the bound without actually satisfying it. Below are the moments obtained from simulating the model in Section 12.1 with both methods and with decision rules approximated on 30 nodes. The moments are quite similar, but the approximated decision rules widely underestimate the time spent at the bounds:

```
Long-run statistics from equilibrium equations solve
Statistics from simulated variables (excluding the first 20 observations):
 Moments
    Mean      Std. Dev. Skewness  Kurtosis  Min       Max       %LB       %UB
    1.2349    0.1199   -0.2424    2.3425    0.8457    1.5736

    0.0007    0.0050    9.7021  115.6033         0    0.1133   96.4720         0
    1.0026    0.0049   -0.7963    6.7106    0.9667    1.0160         0         0
    1.0149    0.0523    3.5551   65.9633    0.7463    2.3121         0         0
    0.2369    0.1141   -0.3280    2.0886         0    0.4000    2.3400    8.0010

 Correlation
    1.0000    0.2742   -0.8393   -0.4939    0.9947
    0.2742    1.0000   -0.4928   -0.4930    0.1947
   -0.8393   -0.4928    1.0000    0.6688   -0.7992
   -0.4939   -0.4930    0.6688    1.0000   -0.4127
    0.9947    0.1947   -0.7992   -0.4127    1.0000

 Autocorrelation
    1         2         3         4         5
    0.8617    0.7473    0.6491    0.5629    0.4876

    0.1079    0.0352    0.0165    0.0054   -0.0009
    0.6710    0.4936    0.3824    0.3068    0.2551
       Inf       Inf       Inf       NaN      -Inf
    0.8712    0.7592    0.6618    0.5751    0.4981

Long-run statistics if simulated with approximated decision rules
Statistics from simulated variables (excluding the first 20 observations):
 Moments
```

```
   Mean      Std. Dev. Skewness  Kurtosis  Min       Max       %LB       %UB
   1.2346    0.1200   -0.2457    2.3441    0.8458    1.5726

   0.0008    0.0050    9.3978  112.0988         0    0.1125   48.8260         0
   1.0026    0.0049   -0.7267    6.6960    0.9669    1.0163         0         0
   1.0144    0.0511    4.3173   74.2896    0.7466    2.3105         0         0
   0.2364    0.1140   -0.3384    2.0941         0    0.4000    1.6080    4.5290

Correlation
   1.0000    0.3114   -0.8411   -0.4970    0.9948
   0.3114    1.0000   -0.5348   -0.5403    0.2300
  -0.8411   -0.5348    1.0000    0.6827   -0.7998
  -0.4970   -0.5403    0.6827    1.0000   -0.4164
   0.9948    0.2300   -0.7998   -0.4164    1.0000

Autocorrelation
   1         2         3         4         5
   0.8620    0.7476    0.6495    0.5633    0.4880

   0.1818    0.0850    0.0502    0.0315    0.0214
   0.6713    0.4938    0.3817    0.3067    0.2555
   0.2894    0.1596    0.1008    0.0676    0.0524
   0.8716    0.7597    0.6624    0.5757    0.4986
```

11 Sketch of the numerical algorithm

The numerical algorithms used in RECS are inspired by Miranda and Fackler (2002), Fackler (2005) and Miranda and Glauber (1995). These are all projection methods with a collocation approach. Several methods are implemented, but we will only present here the default one: the approximation of response variables behavior in a time iteration approach. As already explained, a model is characterized by three equations:

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f(s,x,z), \tag{49}$$

$$z = \mathrm{E}\left[h(s,x,e_+,s_+,x_+)\right], \tag{50}$$

$$s = g(s_-,x_-,e). \tag{51}$$

One way to solve this problem is to find a function that is a good approximation for the behavior of the

response variables. We consider an approximation of response variables,

$$x \approx \mathscr{X}(s, c_x), \tag{52}$$

where c_x are the parameters defining the spline or Chebyshev approximation. To calculate this approximation, we discretize the state space, and the approximation has to hold exactly for all points on the grid.

The expectations operator in equation (50) is approximated by a Gaussian quadrature, which defines a set of pairs $\{e_l, w_l\}$ in which e_l represents a possible realization of shocks and w_l is the associated probability. Using this discretization, and equations (50)–(52), we can express the equilibrium equation (49) as

$$\underline{x}(s) \leq x \leq \overline{x}(s) \quad \perp \quad f\left(s, x, \sum_l w_l h\left(s, x, e_l, g\left(s, x, e_l\right), \mathscr{X}\left(g\left(s, x, e_l\right), c_x\right)\right)\right). \tag{53}$$

For a given approximation, c_x, and a given s, equation (53) is a function of x and can be solved using a mixed complementarity solver.

Once all the above elements are defined, we can proceed to the algorithm, which runs as follows:

1. Initialize the approximation, $c_x^{(0)}$, based on a first-guess, $x^{(0)}$.
2. For each point of the grid of state variables, s_i, solve for x_i equation (53) using an MCP solver:

$$\underline{x}(s_i) \leq x_i \leq \overline{x}(s_i) \quad \perp \quad f\left(s_i, x_i, \sum_l w_l h\left(s_i, x_i, e_l, g\left(s_i, x_i, e_l\right), \mathscr{X}\left(g\left(s_i, x_i, e_l\right), c_x^{(n)}\right)\right)\right). \tag{54}$$

3. Update the approximation using the new values of response variables, $x = \mathscr{X}\left(s, c_x^{(n+1)}\right)$.
4. If $\|c_x^{(n+1)} - c_x^{(n)}\|_2 \geq \varepsilon$, where ε is the precision we want to achieve, then increment n to $n+1$ and go to step 2.

Once the rational expectations equilibrium is identified, the approximation of the decision rules is used to simulate the model.

This is only a sketch of the solution method. In fact, several methods are implemented. For example, instead of using the simple updating rule in step 3, a Newton, or inexact Newton, updating can be used when feasible.

12 Examples of storage models

This section presents some examples of rational expectations models and their implementation in RECS. For the simple competitive storage model, please see the tutorial in Section 3.

12.1 Competitive storage with floor-price backed by public storage (model: STO2)

This model is a simple extension of the competitive storage model in which government attempts to defend a floor price through public storage sale and purchase. The model presented here is close to one of the models presented in Wright and Williams (1988).

This policy is characterized by the following intervention rule, similar to that applying to price band in Section 5.5. All public stocks are sold if the price exceeds the intervention price:

$$P_t > P^{\mathrm{F}} \Rightarrow S_t^{\mathrm{G}} = 0. \tag{55}$$

Price can decrease below the floor if the capacity constraint $\overline{S}^{\mathrm{G}}$ is binding:

$$P_t < P^{\mathrm{F}} \Rightarrow S_t^{\mathrm{G}} = \overline{S}^{\mathrm{G}}. \tag{56}$$

Public stock accumulation occurs to defend the floor price:

$$P_t = P^{\mathrm{F}} \Rightarrow 0 \leq S_t^{\mathrm{G}} \leq \overline{S}^{\mathrm{G}}. \tag{57}$$

These three conditions can be written as one complementarity equation:

$$0 \leq S_t^{\mathrm{G}} \leq \overline{S}^{\mathrm{G}} \quad \perp \quad P_t - P^{\mathrm{F}}. \tag{58}$$

Using the same assumptions as in the competitive storage model presented in Section 3, the equations defining this model are for equilibrium equations

$$S_t : S_t \geq 0 \quad \perp \quad P_t + k - (1-\delta)\beta \mathrm{E}_t\left(P_{t+1}\right) \geq 0, \tag{59}$$

$$H_t : \beta \mathrm{E}_t\left(P_{t+1}\varepsilon_{t+1}\right) = h{H_t}^{\mu}, \tag{60}$$

$$P_t : A_t = {P_t}^{\alpha} + S_t^{\mathrm{S}} + S_t^{\mathrm{G}}, \tag{61}$$

$$S_t^{\mathrm{G}} : 0 \leq S_t^{\mathrm{G}} \leq \overline{S}^{\mathrm{G}} \quad \perp \quad P_t - P^{\mathrm{F}}, \tag{62}$$

and for the transition equation

$$A_t : A_t = (1-\delta)\left(S_{t-1} + S_{t-1}^{\mathrm{G}}\right) + H_{t-1}\varepsilon_t. \tag{63}$$

It should be noted that this model has only one state variable, availability (A_t), because the decision to accumulate public stock depends only on current availability, not on past public stock.

A simulation of this model involves exactly the same steps as in the competitive storage model, so we do not repeat them here; rather, we focus on welfare analysis issues. The reader can consult the file `sto2.m` in RECS demonstration files to see the model resolution.

Welfare analysis Given that this is a simple model, we use it to illustrate welfare analysis in this dynamic stochastic setting. Instantaneous social welfare, w_t, here, is the sum of consumer's surplus, producer's surplus, storer's surplus and public cost:

$$w_t = \int_{P_t}^{\bar{P}} D(p)\,\mathrm{d}p + P_t H_{t-1}\varepsilon_t - \Psi(H_t) + (1-\delta)P_t S_{t-1} - (k+P_t)S_t + (1-\delta)P_t S^{\mathrm{G}}_{t-1} - (k+P_t)S^{\mathrm{G}}_t. \quad (64)$$

Ignoring in consumer's surplus the term independent from policy choice and rearranging using (63), we have

$$w_t = -\frac{P_t^{1+\alpha}}{1+\alpha} + P_t A_t - \Psi(H_t) - (k+P_t)\left(S_t + S^{\mathrm{G}}_t\right). \quad (65)$$

Intertemporal welfare is given by $W_t = \sum_{i=0}^{\infty}\beta^i w_{t+i}$. It is clear that this infinite sum can be expressed as a recursive equation:

$$W_t : W_t = -\frac{P_t^{1+\alpha}}{1+\alpha} + P_t A_t - \Psi(H_t) - (k+P_t)\left(S_t + S^{\mathrm{G}}_t\right) + \beta\,\mathrm{E}_t\left(W_{t+1}\right). \quad (66)$$

Similarly consumer welfare could be calculated recursively as

$$W^C_t : W^C_t = -\frac{P_t^{1+\alpha}}{1+\alpha} + \beta\,\mathrm{E}_t\left(W^C_{t+1}\right). \quad (67)$$

Calculating the welfare of the other agents is not as simple because it generally involves additional state variables. Indeed, producer's welfare depends on past planned production as well as availability, and similarly, storer's welfare is a function of past private stock. So it is important to find a way to express these welfare changes as a function only of availability. Since the private storer operates in expectations at zero profit, its welfare can be reduced to the first-period benefit from holding stocks: $(1-\delta)P_0 S_{-1}$. Similarly, we can decompose producer's welfare between the first-period change and later changes: producer's welfare at time 0 is equal to $P_0 H_{-1}\varepsilon_0 + W^P_0$, where

$$W^P_t : W^P_t = \beta\,\mathrm{E}_t\left(P_{t+1}\varepsilon_{t+1}\right)H_t - \Psi(H_t) + \beta\,\mathrm{E}_t\left(W^P_t\right). \quad (68)$$

We assume that public stocks are zero when the policy starts, so government benefits or losses from public storage are

$$W^G_t : W^G_t = \left[(1-\delta)\beta\,\mathrm{E}_t\left(P_{t+1}\right) - k - P_t\right]S^{\mathrm{G}}_t + \beta\,\mathrm{E}_t\left(W^G_t\right). \quad (69)$$

Equations (66)–(69) can be included in the model definition file (available in file `sto2welf.yaml`):

```
# STO2WELF.yaml Model definition file for the competitive storage with floor-price backed by public storag
# Copyright (C) 2011-2012 Christophe Gouel
# Licensed under the Expat license, see LICENSE.txt

declarations:

  states: [A]
```

```
  controls: [S, H, P, Sg, W, Wc, Wp, Wg]

  expectations: [EP, EPe, EW, EWc, EWp, EWg]

  shocks: [e]

  parameters: [k, delta, r, h, mu, elastD, elastS, PF, Sgbar]


equations:

  arbitrage:

    - P+k-EP*(1-delta)/(1+r)                                                  |    0 <= S  <= inf
    - EPe/(1+r) = h*H^mu                                                      | -inf <= H  <= inf
    - A = P^elastD+S+Sg                                                       | -inf <= P  <= inf
    - P-PF                                                                    |    0 <= Sg <= Sgbar
    - W = -P^(1+elastD)/(1+elastD)+P*A-h*H^(1+mu)/(1+mu)-(P+k)*(S+Sg)+EW/(1+r) | -inf <= W  <= inf
    - Wc = -P^(1+elastD)/(1+elastD)+EWc/(1+r)                                 | -inf <= Wc <= inf
    - Wp = EPe*H/(1+r)-h*H^(1+mu)/(1+mu)+EWp/(1+r)                            | -inf <= Wp <= inf
    - Wg = Sg*(EP*(1-delta)/(1+r)-P-k)+EWg/(1+r)                              | -inf <= Wg <= inf

  transition:

    - A = (1-delta)*(S(-1)+Sg(-1))+H(-1)*e

  expectation:

    - EP  = P(1)
    - EPe = P(1)*e
    - EW  = W(1)
    - EWc = Wc(1)
    - EWp = Wp(1)
    - EWg = Wg(1)


calibration:

  parameters:

    k      :  0.06
    delta  :  0.02
    r      :  0.03
    elastS :  0.2
    h      : 1/(1+r)
    mu     : 1/elastS
    elastD : -0.2
    PF     :  1.02
    Sgbar  :  0.4
```

```
steady_state:

  A  : 1
  S  : 0
  H  : 1
  P  : 1
  Sg : Sgbar
  W  : (-P^(1+elastD)/(1+elastD)+P*A-h*H^(1+mu)/(1+mu)-(P+k)*(S+Sg))/(1-1/(1+r))
  Wc : (-P^(1+elastD)/(1+elastD))/(1-1/(1+r))
  Wp : (P*H/(1+r)-h*H^(1+mu)/(1+mu))/(1-1/(1+r))
  Wg : Sg*(EP*(1-delta)/(1+r)-P-k)/(1-1/(1+r))
```

To analyze the welfare effect of introducing this policy, we have to introduce the same welfare equations in the competitive storage model. The model without policy and the model with policy are solved in the Matlab file `sto2welf.m`:

```
function sto2welf
% STO2WELF Solves the floor-price model and calculates welfare changes

%% Pack model structures
Mu                = 1;
sigma             = 0.05;
model1 = recsmodelinit('sto1welf.yaml',struct('Mu',Mu,'Sigma',sigma^2,'order',7));
model2 = recsmodelinit('sto2welf.yaml',struct('Mu',Mu,'Sigma',sigma^2,'order',7));

%% Define approximation space
[interp1,s] = recsinterpinit(50,0.7,2);
interp2     = interp1;

%% Find a first guess through the perfect foresight solution
interp1 = recsFirstGuess(interp1,model1,s,model1.sss,model1.xss,5);
interp2 = recsFirstGuess(interp2,model2,s,model2.sss,model2.xss,5);

%% Solve for rational expectations (option: hide iterations)
[interp1,x1] = recsSolveREE(interp1,model1,s,[],struct('display',0));
[interp2,x2] = recsSolveREE(interp2,model2,s,[],struct('display',0));

%% Long-run simulation
[~,~,~,~,stat] = recsSimul(model1,interp1,ones(1E4,1),200);
[~,~,~,~,stat] = recsSimul(model2,interp2,ones(1E4,1),200);

%% Welfare change
```

```
sinit = [1 0.2];
[W1,W2] = WelfareCalculation([1 0.2]);
DW = W2-W1;
disp('Welfare change:')
disp(DW)
fid = fopen('./Published/sto2welf.tex','w');
fwrite(fid,latex([DW(2:end) DW(1)],'%0.4f'));
fclose(fid);

%% Plot social welfare against initial availability
n   = 100;
res = zeros(n,1);
H0  = linspace(0.8,1.7,n);
for i=1:n
  [W1,W2] = WelfareCalculation([H0(i) 0]);
  res(i) = W2(1)-W1(1);
end
figure
plot(H0,res)
xlabel('Initial availability')
ylabel('Change in social welfare')
set(gcf,'Color','w')
export_fig sto2welf.eps

function [Welf1,Welf2] = WelfareCalculation(sinit)
  % sinit is the decomposition of initial state between production and private
  % stock: [H*eps (1-delta)*S]

  %% Simulate the model for a given initial state variable to get welfare values
  [~,xsim1] = recsSimul(model1,interp1,sum(sinit),0);
  [~,xsim2] = recsSimul(model2,interp2,sum(sinit),0);

  P0   = [xsim1(3) xsim2(3)];
  beta = 1/(1+model1.params(3));

  %% Calculate welfare
  Welf1 = [xsim1(4:5) P0(1)*sinit(1)+xsim1(6) P0(1)*sinit(2)        0]*(1-beta);
  Welf2 = [xsim2(5:6) P0(2)*sinit(1)+xsim2(7) P0(2)*sinit(2) xsim2(8)]*(1-beta);
end
```

```
end
```

This generates two results: the welfare changes induced by the policy when it starts from a given initial availability, and the sensitivity analysis of social welfare to initial availability. Table 1 shows the welfare change if initial availability is 1.2 of which 1 comes from production. Given the absence of any market imperfections, the competitive equilibrium is optimal (Scheinkman and Schechtman, 1983) and the policy decreases social welfare. It decreases consumer's welfare because of the initial price increase due to stock buildup and because of the negative slope of the demand function, which implies that consumers prefer unstable to stable prices Waugh (1944). In contrast, producer and storer benefit from the policy because of the initial price increase.

Table 1. Welfare Results on Transitional Dynamics (as a percentage of the steady-state commodity budget)

Consumer	Producer	Storer	Government	Total
−0.0198	0.0237	0.0017	−0.0255	−0.0199

Figure 1 shows how social welfare is affected by initial availability. Welfare changes are constant for low availability, as nothing is stored. For a sufficiently high availability, the floor price starts to be binding and public storage kicks in so social welfare decreases. It decreases until an availability of 1.4, where public stock hits its capacity constraint, $\overline{S}^{G} = 0.4$.

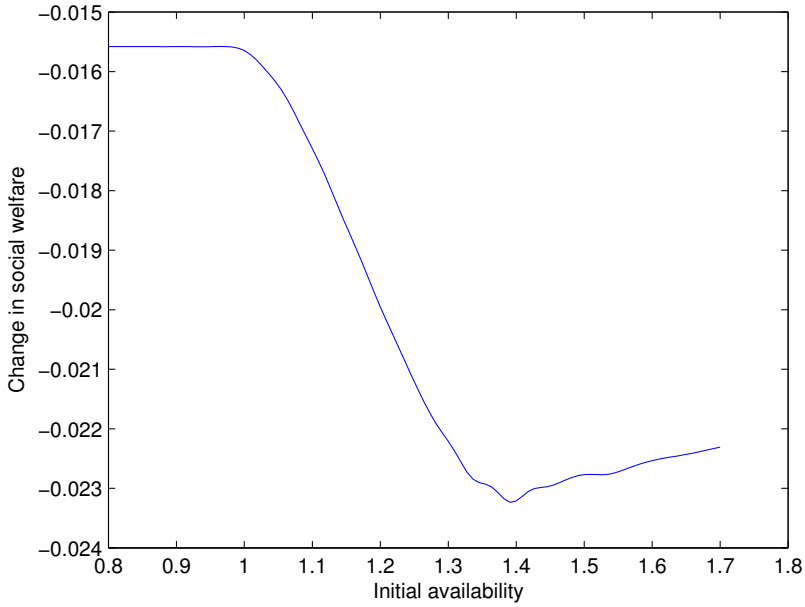


Figure 1. Sensitivity of social welfare to initial availability

12.2 Two-country storage-trade model (model: STO6)

In this section, we describe a model that is useful to analyze the effects of the interaction between one country's domestic food market and the rest of the world: a two-country storage-trade model. This model extends the competitive storage model to a situation with two locations, countries a and b indicated by the subscript $i \in \{a,b\}$.[16] It assumes that in each country there is a demand for final consumption, production, and competitive storage.

The countries are connected through trade, which occurs when the price difference covers the trade costs. Trade is decided by the following spatial arbitrage condition:

$$X_{it} \geq 0 \quad \perp \quad P_{it} + \theta \geq P_{jt}, \text{ for } j \neq i, \tag{70}$$

where X_{it} is the export from region i, and θ is the per-unit transaction cost, inclusive of transport costs. This implies that the prices in country i fall within a moving band that is defined by country-j price and trade costs:

$$P_{jt} - \theta \leq P_{it} \leq P_{jt} + \theta, \text{ for } j \neq i. \tag{71}$$

When adjusted for the subscripts i and j, the model's equations are similar to the single country case, except for market equilibrium, which now accounts for trade. So the equations defining this model are for equilibrium equations, for $i \in \{a,b\}$ and $j \neq i$:

$$S_{it} : S_{it} \geq 0 \quad \perp \quad P_{it} + k - (1-\delta)\beta \mathrm{E}_t\left(P_{it+1}\right) \geq 0, \tag{72}$$

$$H_{it} : \beta \mathrm{E}_t\left(P_{it+1}\varepsilon_{it+1}\right) = h_i {H_{it}}^{\mu}, \tag{73}$$

$$P_{it} : A_{it} + X_{jt} = \gamma_i {P_{it}}^{\alpha} + S_{it} + X_{it}, \tag{74}$$

$$X_{it} : X_{it} \geq 0 \quad \perp \quad P_{it} + \theta \geq P_{jt}, \tag{75}$$

where h_i and γ_i are country-specific scale parameters for supply and demand. The transition equations are, for $i \in \{a,b\}$:

$$A_{it} : A_{it} = (1-\delta) S_{it-1} + H_{it-1}\varepsilon_{it}. \tag{76}$$

Calibration on real data While the previous models were calibrated on arbitrary data, here we show how to calibrate a model on real data. For this, we follow Larson et al. (2012) who calibrate a similar model for the wheat market between the Middle East and North Africa (MENA) region and the rest of the world (RoW).[17]

Storage models in the current literature are not calibrated based on the time-series behavior of the corresponding variables as is common for DSGE models. In addition, there is no procedure in the RECS toolbox

[16] The trade-and-storage model was first developed in Williams and Wright (1991, Ch. 9), and was subsequently analyzed in Miranda and Glauber (1995), and Makki et al. (1996).

[17] In this case, the MENA region includes Algeria, Bahrain, Egypt, Iran, Iraq, Jordan, Kuwait, Lebanon, Libya, Morocco, Oman, Saudi Arabia, Syria, Tunisia, United Arab Emirates, and Yemen.

to make these calibrations automatically. Consequently, the calibration we present here aims at ensuring that a benchmark equilibrium is reproduced at the deterministic steady state, and remaining behavioral parameters are set based on literature survey.

The calibration focuses on the steady state because it can be solved easily without having to solve for the stochastic rational expectations equilibrium, and since in practice it is close to the long-run mean, calibrating the steady state on average observed values should reproduce them as the long-run average of the model variables. To help the calibration, note that we have no storage at steady state since prices are constant: $S_i^* = 0$, where $*$ indicates variables at their steady-state values. Since $\mathrm{E}\,\varepsilon_i = 1$, we have also $A_i^* = H_i^*$. As can be seen in Table 2, the MENA region is always importing wheat, which implies that

$$X_{\mathrm{RoW}}^* > 0, \tag{77}$$

$$X_{\mathrm{MENA}}^* = 0, \tag{78}$$

$$P_{\mathrm{MENA}}^* = P_{\mathrm{RoW}}^* + \theta. \tag{79}$$

It is important to note that a storage model represents a stationary equilibrium, but the corresponding data are not stationary. So, to calibrate the model, we start by removing the underlying trend. The detrended data are presented in Table 2. Since there is no stock at the steady state, quantities have to be adjusted to be consistent with a steady-state equilibrium. The steady-state MENA price is defined by assuming that the price difference between the regions reflects transport costs, which is set at \$35.55/ton based on a recent survey (World Bank and FAO, 2012). Choosing a reference equilibrium point is not sufficient to assign a value to every parameter. To pick up parameters that cannot be identified at the steady state (elasticities, storage costs, discount rate), we rely on the literature. We are guided by the literature on commodity price dynamics, which shows that observed price volatility is consistent with relatively low demand elasticities (Roberts and Schlenker, 2009, Cafiero et al., 2011), thus, we assume demand elasticity to be equal to -0.12, which is toward the lower end of commonly used elasticities. We consider a supply elasticity of 0.2, in line with commonly used supply elasticities for wheat.[18] Interest rates and per unit storage charges are assumed to be the same for each region at 5% per annum and \$22.40 per ton per annum, respectively. The storage costs are based on the findings from a recent World Bank study on wheat markets in MENA (World Bank and FAO, 2012).

Using these behavioral parameters and target values at the steady state, we can calculate h_i and γ_i using equation (73) and (74).

This model and its calibration are written in the Yaml file `sto6MENA.yaml`:

```
# STO6MENA.yaml Model definition file for a two-country storage-trade model with supply reaction (calibrat
# Copyright (C) 2011-2012 Christophe Gouel
# Licensed under the Expat license, see LICENSE.txt

# Country a is MENA, and b is RoW
```

[18] See, for example, FAPRI's elasticity database, available on the internet at http://www.fapri.iastate.edu/tools/elasticity.aspx.

Table 2. Model calibration

	MENA		RoW
Calibration target at steady state			
Consumption (million ton)	75.3		592.9
Production (million ton)	40.2		628
Price ($/ton)	211.55		176
Parameter			
α		-0.12	
β		0.9524	
γ_i	143.2		1102.7
δ		0	
std (ε_i)	0.07		0.03
θ		35.55	
μ		5	
h_i	1.92E$-$6		1.72E$-$12
k		22.4	

Notes: Consumption, production and price in RoW targets are determined as 2011 trend values after applying a Holdrick-Prescott filter (smoothing parameter of 400) to the underlying data (USDA, 2011, for consumption and production and World Bank, 2011, for price). RoW consumption is adjusted to ensure global market equilibrium. MENA price target is defined by adding transport cost to RoW price.

```
declarations:

  states: [Aa, Ab]

  controls: [Sa, Sb, Ha, Hb, Pa, Pb, Xa, Xb]

  expectations: [EPa, EPb, EPea, EPeb]

  shocks: [ea, eb]

  parameters: [k, r, delta, ha, hb, mu, elastS, theta, elastD, gammaa, gammab, beta]


equations:

  arbitrage:

    .. Pa+k-EPa*(1-delta)/(1+r)        |    0 <= Sa <= inf
    .. Pb+k-EPb*(1-delta)/(1+r)        |    0 <= Sb <= inf
    .. EPea/(1+r) = ha*Ha^mu           | -inf <= Ha <= inf
    .. EPeb/(1+r) = hb*Hb^mu           | -inf <= Hb <= inf
    .. Aa+Xb = gammaa*Pa^elastD+Sa+Xa  | -inf <= Pa <= inf
    .. Ab+Xa = gammab*Pb^elastD+Sb+Xb  | -inf <= Pb <= inf
```

```
    .. Pa-Pb+theta                      |    0 <= Xa <= inf
    .. Pb-Pa+theta                      |    0 <= Xb <= inf

  transition:

    .. Aa = (1-delta)*Sa(-1)+Ha(-1)*ea
    .. Ab = (1-delta)*Sb(-1)+Hb(-1)*eb

  expectation:

    .. EPa   = Pa(1)
    .. EPb   = Pb(1)
    .. EPea  = Pa(1)*ea
    .. EPeb  = Pb(1)*eb


calibration:

  parameters:

    k      : 22.4
    r      :  0.05
    beta   : 1/(1+r)
    delta  :  0
    elastS :  0.2
    mu     : 1/elastS
    elastD : -0.12
    theta  : 35.55
    gammaa : (Aa+Xb-Sa-Xa)/Pa^elastD
    gammab : (Ab+Xa-Sb-Xb)/Pb^elastD
    ha     : Pa*beta/Ha^mu
    hb     : Pb*beta/Hb^mu

  steady_state:

    Aa  : Ha
    Ab  : Hb
    Sa  : 0
    Sb  : 0
    Ha  : 40.2
    Hb  : 628
    Pa  : 211.55
    Pb  : 176
    Xa  : 0
    Xb  : Hb-592.9
```

It can be seen that the values entered for the parameters and first guess of the steady state may each be

function of the other, which facilitates model calibration.

The model is solved using the same commands as before, found in `sto6MENA.m`.

References

Adda, J. and Cooper, R. (2003). *Dynamic Economics. Quantitative Method and Applications*. Cambridge, Massachusetts: The MIT Press.

Cafiero, C., Bobenrieth, E. S. A., Bobenrieth, J. R. A. and Wright, B. D. (2011). The empirical relevance of the competitive storage model. *Journal of Econometrics*, 162(1), 44–54.

Deaton, A. (1991). Saving and liquidity constraints. *Econometrica*, 59(5), 1221–1248.

Deaton, A. and Laroque, G. (1992). On the behaviour of commodity prices. *Review of Economic Studies*, 59(1), 1–23.

Fackler, P. L. (2005). A MATLAB solver for nonlinear rational expectations models. *Computational Economics*, 26(2), 173–181.

Fafchamps, M. (2003). *Rural Poverty, Risk and Development*. Edward Elgar Publishing.

Ferris, M. and Pang, J. (1997). Engineering and economic applications of complementarity problems. *Siam Review*, 39(4), 669–713.

Gilbert, C. L. (1996). International commodity agreements: An obituary notice. *World Development*, 24(1), 1–19.

Gouel, C. (2011). Rules versus discretion in food storage policies, mimeo.

Gouel, C. (2013). Optimal food price stabilisation policy. *European Economic Review*, 57, 118–134.

Gouel, C. and Jean, S. (2012). *Optimal Food Price Stabilization in a Small Open Developing Country*. Policy Research Working Paper 5943, The World Bank, URL: `http://go.worldbank.org/23I9ACR4R0`.

den Haan, W. J. and Marcet, A. (1990). Solving the stochastic growth model by parameterizing expectations. *Journal of Business & Economic Statistics*, 8(1), 31–34.

Jayne, T. S. and Jones, S. (1997). Food marketing and pricing policy in Eastern and Southern Africa: A survey. *World Development*, 25(9), 1505–1527.

Larson, D. F., Lampietti, J., Gouel, C., Cafiero, C. and Roberts, J. (2012). *Food security and storage in the Middle East and North Africa*. Policy Research Working Paper Series 6031, The World Bank.

Makki, S. S., Tweeten, L. G. and Miranda, M. J. (1996). Wheat storage and trade in an efficient global market. *American Journal of Agricultural Economics*, 78(4), 879–890.

Makki, S. S., Tweeten, L. G. and Miranda, M. J. (2001). Storage-trade interactions under uncertainty-implications for food security. *Journal of Policy Modeling*, 23(2), 127–140.

Mathiesen, L. (1987). An algorithm based on a sequence of linear complementarity problems applied to a walrasian equilibrium model: An example. *Mathematical Programming*, 37, 1–18.

Miranda, M. J. and Fackler, P. L. (2002). *Applied Computational Economics and Finance*. Cambridge: MIT Press.

Miranda, M. J. and Glauber, J. W. (1995). Solving stochastic models of competitive storage and trade by Chebychev collocation methods. *Agricultural and Resource Economics Review*, 24(1), 70–77.

Miranda, M. J. and Helmberger, P. G. (1988). The effects of commodity price stabilization programs. *The American Economic Review*, 78(1), 46–58.

Munson, T. S. (2002). *Algorithms and Environments for Complementarity*. Ph.D. thesis, University of Wisonsin—Madison.

Osborne, T. (2004). Market news in commodity price theory: Application to the Ethiopian grain market. *The Review of Economic Studies*, 71(1), 133–164.

Park, A. (2006). Risk and household grain management in developing countries. *Economic Journal*, 116(514), 1088–1115.

Porteous, O. C. (2012). Empirical effects of short-term export bans: The case of african maize, mimeo.

Rashid, S., Gulati, A. and Cummings, R. W. (eds.) (2008). *From Parastatals to Private Trade: Lessons from Asian Agriculture*. Baltimore, Md.; Washington D.C.: Johns Hopkins University Press ; International Food Policy Research Institute.

Roberts, M. J. and Schlenker, W. (2009). World supply and demand of food commodity calories. *American Journal of Agricultural Economics*, 91(5), 1235–1242.

Rutherford, T. F. (1995). Extension of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control*, 19(8), 1299–1324.

Scheinkman, J. A. and Schechtman, J. (1983). A simple competitive model with production and storage. *The Review of Economic Studies*, 50(3), 427–441.

Slayton, T. (2009). *Rice Crisis Forensics: How Asian Governments Carelessly Set the World Rice Market on Fire*. Working Paper 163, Center for Global Development.

Waugh, F. V. (1944). Does the consumer benefit from price instability? *The Quarterly Journal of Economics*, 58(4), 602–614.

Williams, J. C. and Wright, B. D. (1991). *Storage and Commodity Markets*. New York: Cambridge University Press.

Winschel, V. and Krätzig, M. (2010). Solving, estimating, and selecting nonlinear dynamic models without the curse of dimensionality. *Econometrica*, 78(2), 803–821.

World Bank and FAO (2012). *The Grain Chain: Food Security and Managing Wheat Imports in Arab Countries*. Washington, DC.: World Bank.

Wright, B. D. (2001). Storage and price stabilization. In B. L. Gardner and G. C. Rausser (eds.) *Marketing, Distribution and Consumers*, volume 1B, part 2 of *Handbook of Agricultural Economics*, chapter 14, (pp. 817–861). Amsterdam: Elsevier.

Wright, B. D. and Williams, J. C. (1982). The economic role of commodity storage. *The Economic Journal*, 92(367), 596–614.

Wright, B. D. and Williams, J. C. (1984). The welfare effects of the introduction of storage. *The Quarterly Journal of Economics*, 99(1), 169–192.

Wright, B. D. and Williams, J. C. (1988). The incidence of market-stabilising price support schemes. *The Economic Journal*, 98(393), 1183–1198.