YADA Manual — Computational Details

Anders Warne

March 11, 2009

Abstract: YADA (Yet Another Dsge Application) is a Matlab program for Bayesian estimation and evaluation of Dynamic Stochastic General Equilibrium and vector autoregressive models. This paper provides the mathematical details for the various functions used by the software. First, the different prior distributions that are supported, the state-space representation and the Kalman filter used to evaluate the log-likelihood are presented. Since YADA uses the Anderson-Moore algorithm (AiM) to solve log-linearized DSGE models, the matrix specification of the DSGE model is provided and linked to the state-space model. Various tools for evaluating an estimated DSGE model are provided, including impulse response functions, forecast error variance decompositions, historical forecast error and observed variable decompositions. This paper also provides information about the various types of input that YADA requires and how these inputs should be prepared. Furthermore, it discusses how the posterior mode is computed, including how the original model parameters are transformed internally to facilitate the posterior mode estimation. Next, the paper provides some details on the algorithm used for sampling from the posterior distribution: the random walk Metropolis algorithm. In order to conduct inference based on the draws from the MCMC sampler, tools for evaluating convergence are considered next. We are here concerned both with simple graphical tools, as well as formal tools for single and parallel chains. Different methods for estimating the marginal likelihood are considered thereafter. Such estimates may be used to evaluate posterior probabilities for different DSGE models. Estimation of a VAR model with a prior on the steady state parameters is also examined. The main concerns are: prior hyperparameters, posterior mode estimation, posterior sampling via the Gibbs sampler, and marginal likelihood calculation (when the full prior is proper). Finally, forecasting issues, such as the unconditional and conditional predictive distributions, are examined. At the end of each section, the main YADA functions for the different tasks are presented.

 I have received valuable comments and suggestions by the other members of the NAWM project Kai Christoffel, Günter Coenen, Juha Kilponen (Soumen Pankki), Igor Vetlov, and Pascal Jacquinot, as well as our consultant from Sveriges Riksbank, Malin Adolfson. A special thanks goes to Mattias Villani for his patience when trying to answer all my questions on Bayesian analysis. I have also benefitted greatly from discussions with Frank Schorfheide. And last but not least, I am grateful to Magnus Jonsson and Stefan Laséen at Sveriges Riksbank and to Peter Welz for helping me track down a number of unpleasant bugs and to improve the generality of the YADA code.

Contents

1. Introduction

YADA is a Matlab program for Bayesian estimation of and inference in Dynamic Stochastic General Equilibrium (DSGE) and Vector Autoregressive (VAR) models. DSGE models are micro-founded optimization-based models that have become very popular in macroeconomics over the past 25 years. The most recent generation of DSGE models is not just attractive from a theoretical perspective, but is also showing great promise in areas such as forecasting and quantitative policy analysis; see, e.g., Adolfson, Laséen, Lindé, and Villani (2007a), Christiano, Eichenbaum, and Evans (2005), Smets and Wouters (2003, 2005), and An and Schorfheide (2007).

YADA is developed in connection with the New Area-Wide Model (NAWM) project at the ECB; cf. Christoffel, Coenen, and Warne (2008). The software relies on code made available to the NAWM project by colleagues at both central bank institutions and the academic world. In particular, it relies to some extent on the code written by the group of researchers at Sveriges Riksbank that have developed the Riksbank DSGE model (Ramses). This group includes Malin Adolfson, Stefan Laséen, Jesper Lindé, and Mattias Villani. YADA also relies on a Matlab version of the Anderson-Moore algorithm for solving linear rational expectations models (AiM) and writing them in state-space form; see, e.g., Anderson (1999) or Zagaglia (2005).

In contrast with other software that can estimate DSGE models, YADA has a Graphical User Inference (GUI) from which all actions and settings are controlled. The current document does not give much information about the GUI. Instead it primarily focuses on the mathematical details of the functions needed to calculate, for instance, the log-likelihood function. The instances when this documents refers to the GUI are always linked to functions that need certain data from the GUI. The help file in the YADA distribution covers the GUI functionality.

This document is structured as follows. In the next section, we start with Bayes theorem, giving the notation for the prior density, the conditional and the marginal density of the data, as well as the posterior density. We then turn to the density functions that can be used in YADA for the prior distribution of the DSGE model parameters. Parametric definitions are provided and some of their properties are stated.

This leads us into the actual calculation of the likelihood using the Kalman filter. Notation and main equations of this tool are given in Section 3. Since the computation of the Kalman filter for a particular value of the parameters requires that we can solve the DSGE model for that value, Section 4 provides an overview of the matrix representation of linear rational expectations models and how these can be analysed with the Anderson-Moore algorithm.

In Section 5 we turn to various tools for analysing the properties of a DSGE model. Such tools include, for example, impulse responses and forecast error variance decompositions. The next steps are taken in Section 6, where the reading of observed data into YADA, the setup of the prior, the construction of the files for the measurement equations, additional parameters, and the AiM model file are covered. Once these tasks have been achieved, YADA is ready for estimation of the posterior mode and the inverse Hessian at the mode.

Since some of the parameters can have bounded support, e.g., a standard deviation of a shock only taking on positive values, the optimization problem typically involves inequality restrictions. Having to take such restrictions into account can slow down the optimization time considerably. A natural way to avoid this computational issue is to transform these parameters such that the support of the transformed parameters is the real line. In that way we shift from a constrained optimization problem to an unconstrained one. The specific transformations that YADA can apply are discussed in Section 7 as well as how this affects the estimation of the posterior mode, a topic which is thereafter covered in Section 8.

Once the posterior mode and the Hessian at the mode has been calculated, we can construct draws from the posterior distribution using the random walk Metropolis algorithm. This issue is discussed in section 9. Given a sample of such draws it is important to address the question if the posterior sampler has converged. In Section 10 we deal with simple graphical tools as well as formal statistical tools for assessing convergence in a single chain and in parallel chains.

When we are satisfied that our posterior sampler has converged, we may turn to other issues regarding Bayesian inference. In Section 11 we examine the problem of computing the marginal likelihood of the DSGE model. This object can be used for relative model comparisons and potentially also for comparisons with alternative models, such as Bayesian VARs.

The next topic of the document is Bayesian VARs. In particular, YADA supports VAR models for forecasting purposes. The types of prior that may be used, computation of posterior mode, posterior sampling with the Gibbs sampler, and the computation of the marginal likelihood are all given some attention in Section 12. One specific feature of the Bayesian VAR models that YADA support is that the steady state parameters are modelled explicitly.

Finally, certain out-of-sample forecasting issues are discussed in Section 13 for both the DSGE/state-space models and Bayesian VARs. Both unconditional and conditional forecasting are considered as well as a means for checking if the conditional forecasts are subject to the famous Lucas (1976) critique or not. Since the document concerns the implementation of various mathematical issues in a computer program, all sections end with a part that discusses some details of the main functions that are made use of by YADA.

2. Prior and Posterior Distributions

2.1. *Bayes Theorem*

Let y_t denote the observed variables, a vector of dimension n. Furthermore, the sample is given by $t = 1, \ldots, T$ and we collect the data into the $n \times T$ matrix Y. For simplicity we here neglect any exogenous or predetermined variables as they do not matter for the exposition.

The density function for a random matrix Y conditional on θ is given by $p(Y|\theta)$, where θ is a vector of parameters. The joint prior distribution of θ is denoted by $p(\theta)$. From Bayes theorem we then know that the posterior distribution of θ, denoted by $p(\theta|Y)$, is related to these functions through

$$p(\theta|Y) = \frac{p(Y|\theta)p(\theta)}{p(Y)}, \tag{2.1}$$

where $p(Y)$ is the marginal density of the data, defined from

$$p(Y) = \int_{\theta\in\Theta} p(Y|\theta)p(\theta)d\theta, \tag{2.2}$$

with Θ being the support of θ. Since $p(Y)$ is a constant we know that the posterior density of θ is proportional to the product $p(Y|\theta)p(\theta)$. Hence, if we can characterize the distribution of this product we would know the posterior distribution of θ. For complex models like those belonging to the DSGE family this characterization is usually not possible. Methods based on Markov Chain Monte Carlo (MCMC) theory can instead be applied to generate draws from the posterior.

Still, without having to resort to such often time consuming calculations it should be noted that the mode of the posterior density can be found by maximizing the product $p(Y|\theta)p(\theta)$. Since this product is usually highly complex, analytical approaches to maximization are ruled out from the start. Instead the posterior mode, denoted by $\tilde{\theta}$, can be estimated using numerical methods. In the Section 2.2 we provide the individual density functions for the elements of θ that YADA supports. Through the independence assumption, the joint prior $p(\theta)$ is simply the product of these individual (and marginal) prior densities. The computation of the likelihood function for any given value of θ is thereafter discussed in Section 3.

2.2. *Density Functions for the Prior Distributions*

In the Bayesian DSGE framework it is usually assumed that the parameters to be estimated, denoted here by θ, are a priori independent. For parameters that have support $\mathbb{R}$, the prior distribution is typically Gaussian. Parameters that instead have support $\mathbb{R}_+$ tend to have either gamma or inverse gamma prior distributions, while parameters with support (c, d), where $d > c$ and both are finite, are usually assumed to have beta prior distributions; see, e.g., An and Schorfheide (2007). In some cases, e.g., Adolfson et al. (2007a), the distribution may be left

truncated normal for a certain parameter. Below the density functions of these distributions as well as of the uniform distribution are given. In addition, YADA can support a number of additional distributions through parameter transformation functions. These include the Weibull and Snedecor (better known as the F or Fisher) distributions. First, however, the gamma and beta functions are presented as these often appear in the integration constants.

2.2.1. *The Gamma and Beta Functions*

The *gamma* function is defined by the following integral identity:

$$\Gamma(a) = \int_0^\infty x^{a-1} \exp(-x) dx, \quad a > 0. \tag{2.3}$$

In many cases integer or half integer values of a are used. Here it is useful to know that $\Gamma(1) = 1$, while $\Gamma(1/2) = \sqrt{\pi}$. Integration by parts of (2.3) gives for $a > 1$ that $\Gamma(a) = (a-1)\Gamma(a-1)$.

The beta function $\beta(a,b)$ for $a, b > 0$ is defined as:

$$\beta(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1} dx. \tag{2.4}$$

It is related to the gamma function through the following relationship:

$$\beta(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}. \tag{2.5}$$

Matlab contains two useful functions for dealing with the gamma function. One is `gamma` which works well for relatively small values of a. The other is `gammaln` which calculates the natural logarithm of the gamma function and works well also with large values of a. Similarly, for the beta function Matlab provides the functions `beta` and `betaln`.

2.2.2. *Gamma, χ^2, Exponential, Erlang and Weibull Distributions*

A random variable $z > 0$ has a gamma distribution with shape parameter $a > 0$ and scale parameter $b > 0$, denoted by $z \sim G(a,b)$ if and only if its pdf is given by

$$p_G(z|a,b) = \frac{1}{\Gamma(a)b^a} z^{a-1} \exp\left(\frac{-z}{b}\right). \tag{2.6}$$

It is worthwhile to keep in mind that if $z \sim G(a,b)$ and $y = z/b$, then $y \sim G(a,1)$. Furthermore, $E[z] = ab$, while $E[(z-ab)^2] = ab^2$; see, e.g., Bauwens, Lubrano, and Richard (1999) or Zellner (1971). If $a > 1$, then the pdf has a unique mode at $z = b(a-1)$.

Letting μ_Γ and σ_Γ^2 denote the mean and the variance, respectively, we can directly see that

$$a = \frac{\mu_\Gamma^2}{\sigma_\Gamma^2}, \quad b = \frac{\sigma_\Gamma^2}{\mu_\Gamma}. \tag{2.7}$$

In practise, most economists (and econometricians) are probably more comfortable formulating a prior in terms of the mean and the standard deviation, than in terms of a and b. The mode can, when it exists, also be expressed in terms of the mean and the variance parameters. Letting $\tilde{\mu}_\Gamma$ be the mode, equation (2.7) gives us

$$\tilde{\mu}_\Gamma = \mu_\Gamma - \frac{\sigma_\Gamma^2}{\mu_\Gamma},$$

and the mode therefore exists when $\mu_\Gamma^2 > \sigma_\Gamma^2$, i.e., when the mean is greater than the standard deviation.

One special case of the gamma distribution is the $\chi^2(q)$. Specifically, if $z \sim \chi^2(q)$ then this is equivalent to stating that $z \sim G(q/2, 2)$, with mean q and variance $2q$. The mode of this distribution exists and is unique when $q \geq 3$ and is equal to $q-2$.

Another special case is the *exponential distribution*. This is obtained by letting $a = 1$ in (2.6). With $z \sim \mathcal{E}(b) \equiv G(1,b)$, we find that the mean is equal to $\mu_\mathcal{E} = b$ and the variance is $\sigma_\mathcal{E}^2 = b^2$. Similarly, the *Erlang* distribution is the special case of the gamma when a is an integer. For this

case we have that $\Gamma(a) = (a-1)!$, where ! is the factorial function. The parameterization of the Erlang density is usually written in terms of $\lambda = 1/b$, a rate parameter.

YADA can also support the *Weibull distribution* through the gamma prior and the so called *file with parameters to update*; see Section 6.4. Specifically, if $x \sim G(1,1) \equiv \mathcal{E}(1)$ and $x = (z/a)^b$ with $a, b > 0$, then z has a Weibull distribution with scale parameter a and shape parameter b, i.e. $z \sim W(a,b)$. In YADA one would specify a prior for the random variable x and compute $z = ax^{1/b}$ in the file with parameters to update. The density function is now

$$p_W(z|a,b) = \frac{b}{a^b} z^{b-1} \exp\left(-\left[\frac{z}{a}\right]^b\right), \qquad z > 0.$$

The mean of the Weibull distribution is $\mu_W = a\Gamma(1 + (1/b))$, the variance is $\sigma_W^2 = a^2[\Gamma(1 + (2/b)) - (\Gamma(1 + (1/b)))^2]$, whereas the mode exists and is given by $\tilde{\mu}_W = a((b-1)/b)^{(1/b)}$ when $b > 1$.

2.2.3. *Inverse Gamma Distribution*

The definition of the inverse gamma distribution given below is taken from Zellner (1971).[1] A random variable $z > 0$ has an inverse gamma distribution with parameters $a > 0$ and $b > 0$, denoted by $z \sim IG(a,b)$, if and only if its pdf is given by

$$p_{IG}(z|a,b) = \frac{2}{\Gamma(a) b^a} z^{-(2a+1)} \exp\left(\frac{-1}{bz^2}\right). \tag{2.8}$$

This pdf has a unique mode at $z = (2/(b(2a+1)))^{1/2}$.

This pdf is commonly used as a prior for a standard deviation. Letting $\sigma = z$, $a = q/2$, and $b = 2/qs^2$, we get

$$p_{IG}(\sigma|s,q) = \frac{2}{\Gamma(q/2)} \left(\frac{qs^2}{2}\right)^{q/2} \sigma^{-(q+1)} \exp\left(\frac{-qs^2}{2\sigma^2}\right), \tag{2.9}$$

where $s, q > 0$. The parameter q is an integer (degrees of freedom) while s is a location parameter. This pdf has a unique mode at $\tilde{\mu}_{IG} = s(q/(q+1))^{1/2}$. Hence, the mode is below s for finite q and increasing towards s with q.

The moments of this distribution exists when q is sufficiently large. For example, if $q \geq 2$, then the mean is $\mu_{IG} = (\Gamma((q-1)/2)/\Gamma(q/2))(q/2)^{1/2}s$, while if $q \geq 3$ then the variance is given by $\sigma_{IG}^2 = qs^2/(q-2) - \mu_{IG}^2$; see Zellner (1971) for details.

Another parameterization of the inverse gamma distribution is used in the software developed by Adolfson et al. (2007a). Letting $a = d/2$ and $b = 2/c$, the pdf in (2.8) can be written as:

$$p_{IG}(z|c,d) = \frac{2}{\Gamma(d/2)} \left(\frac{c}{2}\right)^{d/2} z^{-(d+1)} \exp\left(\frac{-c}{2z^2}\right).$$

The mode of this parameterization is found by setting $z = (c/(d+1))^{1/2}$. With $c = qs^2$ and $d = q$ this parameterization is equal to that in equation (2.9) with $z = \sigma$.

2.2.4. *Beta and Snedecor (F) Distributions*

A random variable $c < x < d$ has a beta distribution with parameters $a > 0$, $b > 0$, c and d, denoted by $x \sim B(a,b,c,d)$ if and only if its pdf is given by

$$p_B(x|a,b,c,d) = \frac{1}{(d-c)\beta(a,b)} \left(\frac{x-c}{d-c}\right)^{a-1} \left(1 - \frac{x-c}{d-c}\right)^{b-1}. \tag{2.10}$$

The standardized beta distribution can directly be determined from (2.10) by defining the random variable $z = (x-c)/(d-c)$. Hence, $0 < z < 1$ has a beta distribution with parameters

[1] Bauwens, Lubrano, and Richard (1999) refer to the inverse gamma distribution as the inverse gamma-1 distribution. The inverse gamma-2 distribution is then defined for a variable $x = z^2$, where z follows an inverse gamma-1 distribution.

$a > 0$ and $b > 0$, denoted by $z \sim B(a,b)$ if and only if its pdf is given by

$$p_{SB}(z|a,b) = \frac{1}{\beta(a,b)} z^{a-1}(1-z)^{b-1}. \tag{2.11}$$

For $a,b > 1$, the mode of (2.11) is given by $z = (a-1)/(a+b-2)$. Zellner (1971) provides general expressions for the moments of the beta pdf in (2.11). For example, the mean of the standardized beta is $a/(a+b)$, while the variance is $ab/((a+b)^2(a+b+1))$.

Let μ_{SB} and σ^2_{SB} be the mean and the variance, respectively, of the standardized beta distribution. Some algebra later we find that the a and b parameters can be expressed as:

$$\begin{aligned} a &= \frac{\mu_{SB}}{\sigma^2_{SB}} \left[\mu_{SB}\big(1-\mu_{SB}\big) - \sigma^2_{SB}\right], \\ b &= \frac{\big(1-\mu_{SB}\big)}{\mu_{SB}} a. \end{aligned} \tag{2.12}$$

From these expressions we see that a and b are defined from μ_{SB} and σ^2_{SB} when $\mu_{SB}\big(1-\mu_{SB}\big) > \sigma^2_{SB} > 0$ with $0 < \mu_{SB} < 1$.

Letting μ_B and σ^2_B be the mean and the variance of $x \sim B(a,b,c,d)$, it is straightforward to show that:

$$\begin{aligned} \mu_B &= c + \big(d-c\big)\mu_{SB}, \\ \sigma^2_B &= \big(d-c\big)^2 \sigma^2_{SB}. \end{aligned} \tag{2.13}$$

This means that we can express a and b as functions of μ_B, σ_B, c, and d:

$$\begin{aligned} a &= \frac{(\mu_B - c)}{(d-c)\sigma^2_B} \left[\big(\mu_B - c\big)\big(d-\mu_B\big) - \sigma^2_B\right], \\ b &= \frac{\big(d-\mu_B\big)}{\big(\mu_B - c\big)} a. \end{aligned} \tag{2.14}$$

The conditions that $a > 0$ and $b > 0$ means that $c < \mu_B < d$, while $(\mu_B - c)(d - \mu_B) > \sigma^2_B$.

The beta distribution is related to the gamma distribution in a particular way. Suppose $x \sim G(a,1)$ while $y \sim G(b,1)$. As shown by, e.g., Bauwens, Lubrano, and Richard (1999, Theorem A.3), the random variable $z = x/(x+y) \sim B(a,b)$.

The beta distribution is also related to the *Snedecor* or F (Fisher) distribution. For example, suppose that $x \sim B(a/2, b/2)$ with a,b being positive integers. Then $z = bx/(a(1-x))$ can be shown to have an $F(a,b)$ distribution; cf. Bernardo and Smith (2000, Chapter 3). That is,

$$p_F(z|a,b) = \frac{a^{(a/2)} b^{(b/2)}}{\beta(a/2, b/2)} z^{(a/2)-1} \big(b + az\big)^{(a+b)/2}, \qquad z > 0.$$

The mean of this distribution exists if $b > 2$ and is then $\mu_F = b/(b-2)$. The mode exists and is unique with $\tilde{\mu}_F = (a-2)b/(a(b+2))$ when $a > 2$. Finally, if $b > 4$ then the variance exists and is given by $\sigma^2_F = 2b^2(a+b-2)/(a(b-4)(b-2)^2)$.

Although YADA does not directly support the F distribution, the combination of the beta prior and the *file with parameters to update* (see Section 6.4) makes it possible to indirectly support this as a prior.

2.2.5. *Normal and Log-Normal Distributions*

For completeness, the Gaussian density function is also provided. Specifically, a random variable z is Gaussian with parameters $\mu \in \mathbb{R}$, $\sigma > 0$, denoted by $z \sim N(\mu, \sigma^2)$, if and only if its pdf is given by

$$p_N(z|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right). \tag{2.15}$$

The mode of this density is, of course, $z = \mu$, while the mean is also equal to μ and the variance is σ^2.

YADA does not directly support the log-normal as a prior distribution. Nevertheless, log-normal priors can be used by combining the normal distribution with the *file with parameters to update*; cf. Section 6.4. That is, the normal prior is specified for the random variable x, while $z = \exp(x)$ is given in the file with parameters to update.

The density function of the log-normal distribution is

$$p_{LN}(z|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} z^{-1} \exp\left(-\frac{1}{2\sigma^2}\left(\ln z - \mu\right)^2\right), \qquad z > 0.$$

The mean of the log-normal distribution is $\mu_{LN} = \exp(\mu + (\sigma^2/2))$, the variance is $\sigma^2_{LN} = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$, while the mode is $\tilde{\mu}_{LN} = \exp(\mu - \sigma^2)$; see Gelman, Carlin, Stern, and Rubin (2004).

2.2.6. *Left Truncated Normal Distribution*

The left truncated normal distribution can be defined from (2.15) by introducing a lower bound c. This means that a random variable $z \geq c$ is left truncated normal with parameters $\mu \in \mathbb{R}$, $\sigma > 0$, and finite c, denoted by $z \sim LTN(\mu, \sigma^2, c)$, if and only if its pdf is

$$p_{LTN}(z|\mu,\sigma,c) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)\left(1 - \Phi((c-\mu)/\sigma)\right)^{-1}, \tag{2.16}$$

where

$$\begin{aligned} \Phi(a) &= \begin{cases} (1 + \kappa(a/\sqrt{2}))/2 & \text{if } a > 0 \\ (1 - \kappa(-a/\sqrt{2}))/2 & \text{otherwise,} \end{cases} \\ \kappa(b) &= \frac{2}{\sqrt{\pi}} \int_0^b \exp\left(-x^2\right) dx. \end{aligned} \tag{2.17}$$

Hence, the left truncated normal density is given by the normal density divided by 1 minus the cumulative normal distribution up to the point of left truncation, i.e., $(c-\mu)/\sigma$. The function $\kappa(b)$ is often called the error function. In Matlab, its name is `erf`.

As long as $\mu \geq c$, the mode is given by $z = \mu$, while $\mu < c$ means that the mode is $z = c$.

2.2.7. *Uniform Distribution*

A random variable z is said to have a uniform distribution with parameters a and b with $b > a$, denoted by $z \sim U(a,b)$ if and only if its pdf is given by

$$p_U(z|a,b) = \frac{1}{b-a}. \tag{2.18}$$

The mean and the variance of this distribution are:

$$\begin{aligned} \mu_U &= \frac{a+b}{2}, \\ \sigma^2_U &= \frac{(b-a)^2}{12}. \end{aligned}$$

The beta distribution is equivalent to a uniform distribution with lower bound c and upper bound d when $\mu_B = (c+d)/2$ and $\sigma^2_B = (d-c)^2/12$; see, also, Bauwens, Lubrano, and Richard (1999) for additional properties of the uniform distribution.

2.3. *YADA Code*

The density functions presented above are all written in natural logarithm form in YADA. The main reason for this is keeping the scale manageable. For example, the exponential function in Matlab, like any other computer software available today, cannot deal with large numbers. If one attempts to calculate e^{700} one obtains `exp(700) = 1.0142e+304`, while `exp(720) = Inf`. Furthermore, and as discussed in Section 2.2.1, the gamma and beta functions return infinite values for large input values, while the natural logarithm of these functions return finite values also for large (but finite) input values.

2.3.1. `logGammaPDF`

The function `logGammaPDF` calculates $\ln p_G(z|a,b)$ in (2.6). Required inputs are `z`, `a`, `b`, while the output is `lnG`. Notice that all inputs can be vectors, but that the function does not check that the dimensions match. This is instead handled internally in the files calling or setting up the input vectors for `logGammaPDF`.

2.3.2. `logInverseGammaPDF`

The function `logInverseGammaPDF` calculates $\ln p_{IG}(\sigma|s,q)$ in (2.9). Required inputs are `sigma`, `s`, `q`, while the output is `lnIG`. Notice that all inputs can be vectors, but that the function does not check that the dimensions match. This is instead handled internally in the files calling or setting up the input vectors for `logInverseGammaPDF`.

2.3.3. `logBetaPDF`

The function `logBetaPDF` calculates $\ln p_B(z|a,b,c,d)$ in (2.10). Required inputs are `z`, `a`, `b`, `c`, and `d`, while the output is `lnB`. Notice that all inputs can be vectors, but that the function does not check that the dimensions match. This is instead handled internally in the files calling or setting up the input vectors for `logBetaPDF`.

2.3.4. `logNormalPDF`

The function `logNormalPDF` calculates $\ln p_N(z|\mu,\sigma)$ in (2.15). Required inputs are `z`, `mu`, `sigma`, while the output is `lnN`. Notice that all inputs can be vectors, but that the function does not check that the dimensions match. This is instead handled internally in the files calling or setting up the input vectors for `logNormalPDF`.

2.3.5. `logLTNormalPDF`

The function `logLTNormalPDF` calculates $\ln p_{LTN}(z|\mu,\sigma,c)$ in (2.16). Required inputs are `z`, `mu`, `sigma`, `c`, while the output is `lnLTN`. This function calls the `PhiFunction` described below. Notice that the function does not check if $z \geq c$ holds. This is instead handled by the functions that call `logLTNormalPDF`.

2.3.6. `logUniformPDF`

The function `logUniformPDF` calculates $\ln P_U(z|a,b)$ in (2.18), i.e., the log height of the uniform density, i.e., $-\ln(b-a)$. The required inputs are `a` and `b`, where the former is the lower bound and the latter the upper bound of the uniformly distributed random vector z.

2.3.7. `PhiFunction`

The function `PhiFunction` evaluates the expression for $\Phi(a)$ in (2.17). The required input is the vector `a`, while the output is `PhiValue`, a vector with real numbers between 0 and 1.

2.3.8. *Discussion*

YADA needs input from the user regarding the type of prior to use for each parameter it should estimate. In the case of the beta and normal distributions the parameters needed as input are assumed to be the mean and the standard deviation. If you wish to have a general beta prior you need to provide the upper and lower bounds as well or YADA will set these to 1 and 0, respectively. For the gamma and the left truncated normal distribution, the parameters to assign values for are given by μ, σ, and a lower bound c. For the gamma distribution this the first two parameters are the mean and the standard deviation, while for the left truncated normal they are defined in equation (2.16).

Similarly, for the inverse gamma distribution the parameters to select values for are s, q, and a lower bound c. The s parameter is, as mentioned above, a location parameter and q is a degrees of freedom parameter that takes on integer values. The location parameter s can, e.g., be selected such that the prior has a desired mode. Relative to equation (2.9) we are now dealing with the location parameter $(s-c)$ and the random variable $(\sigma - c)$ since the

density is expressed for a random variable that is positive. The mode for this parameterization is $\tilde{\mu}_{IG} = s(q/(q+1))^{1/2} + c(1-(q/(q+1))^{1/2})$.

Finally, for the uniform distribution the lower and the upper bound are required.

For all distributions but the beta and the gamma, there is no need for internally transforming the distribution parameters. For these two distributions, however, transformations into the a and b parameters are needed, using the mean and the standard deviation as input. YADA has two functions that deal with these transformations, `MomentToParamGammaPDF` and `MomentToParamStdbetaPDF`, respectively. Both functions take vectors as input as provide vectors as output. The formulas used are found in equations (2.7) and (2.12) above. Since YADA supports a lower bound that can be different from zero for the gamma prior, the mean minus the lower bound is used as input for the transformation function `MomentToParamGammaPDF`. Similarly, the mean and the standard deviation of the standardized beta distribution are computed from the mean and the standard deviation as well as the upper and lower bounds of the general beta distribution. Recall that these relations are $\mu_{SB} = (\mu_B - c)/(d-c)$ and $\sigma_{SB} = \sigma_B/(d-c)$.

3. The Kalman Filter

The solution to a log-linearized DSGE model can be written as a VAR model, where some of the variables may be unobserved. This suggests that any approach to estimation of its parameters is closely linked to estimation of state-space models. In the case of DSGE models, the solution of the model is represented by the state equation, while the measurement equation provides the link from the state (or model) variables into the observable variables, the steady-state of the DSGE model, and possible measurement errors.

The Kalman filter was originally developed by Kalman (1960) and Kalman and Bucy (1961) to estimate state-space models. It can be applied to compute the sample log-likelihood function of the data for a given set of parameter values, i.e., $\ln p(Y|\theta)$. It can also be used for forecasting the observed variables in the model as well as all the state variables. The notation used in this section follows the notation in Hamilton (1994, Chapter 13) closely, where details on the derivation of the filter are also found.

3.1. *The State-Space Representation*

Let y_t denote an $n \times 1$ dimensional vector of variables that are observed at date t. The *measurement* (or observation) equation for y is given by:

$$y_t = A'x_t + H'\xi_t + w_t. \tag{3.1}$$

The vector x_t is $k \times 1$ dimensional and contains only deterministic variables. The vector ξ_t is $r \times 1$ dimensional and is known as the state vector and contains possibly unobserved variables. The term w_t is white noise and is called the measurement error.

The *state* equation of the dynamics of y is given by:

$$\xi_t = F\xi_{t-1} + v_t, \tag{3.2}$$

where F is the state transition matrix. The term v_t is white noise and it is assumed that v_t and w_τ are uncorrelated for all t and τ, with

$$E[v_t v_\tau'] = \begin{cases} Q \text{ for } t = \tau, \\ 0 \text{ otherwise}, \end{cases}$$

while

$$E[w_t w_\tau'] = \begin{cases} R \text{ for } t = \tau, \\ 0 \text{ otherwise}. \end{cases}$$

The parameter matrices are given by A $(k \times n)$, H $(r \times n)$, F $(r \times r)$, Q $(r \times r)$, and R $(n \times n)$. These matrices are known once we provide a value for θ. To initialize the process described by (3.1) and (3.2), it is assumed that ξ_1 is uncorrelated with any realizations of v_t or w_t.

3.2. *Forecasting y_t and ξ_t*

Let $\mathcal{Y}_t = \{y_t, y_{t-1}, \ldots, y_1, x_t, x_{t-1}, \ldots, x_1\}$ denote the set of observations up to and including period t. The Kalman filter provides a method for computing optimal forecasts of y_t conditional on its past values and on the vector x_t as well as the associated mean squared error matrix.[2] These forecasts and their mean squared errors can then be used to compute the value of the log-likelihood function for y. Given the state-space representation in (3.1) and (3.2), it can directly be seen that the calculation of such forecasts requires forecasts of the state vector ξ_t conditional on the observed variables.

Let $\xi_{t+1|t}$ denote the linear projection of ξ_{t+1} on $\mathcal{Y}_t$. The Kalman filter calculates these forecasts recursively, generating $\xi_{1|0}$, $\xi_{2|1}$, and so on. Associated with each of these forecasts is a mean squared error matrix, represented by

$$P_{t+1|t} = E\left[(\xi_{t+1} - \xi_{t+1|t})(\xi_{t+1} - \xi_{t+1|t})'|\mathcal{Y}_t\right].$$

To start up the algorithm we may let:

$$\xi_{1|0} = E[\xi_1].$$

In a log-linearized version of a DSGE model, ξ_t measures the state variables as deviations from steady state. Hence, a natural candidate for the expected value of ξ_1 is 0 (the variables are initially in steady state). Note, however, that this candidate is *natural* when ξ_t is covariance stationary, i.e., if all eigenvalues of F are inside the unit circle.

Let $y_{t|t-1}$ be the linear projection of y_t on $\mathcal{Y}_{t-1}$ and x_t. From the measurement equation (3.1) and the assumption about w_t we have that:

$$y_{t|t-1} = A'x_t + H'\xi_{t|t-1}. \tag{3.3}$$

It can be shown (see, Hamilton, 1994, p. 379) that:

$$E\left[(y_t - y_{t|t-1})(y_t - y_{t|t-1})'|\mathcal{Y}_{t-1}\right] = H'P_{t|t-1}H + R. \tag{3.4}$$

To compute the forecasts and mean squared errors for y, we thus need to know the sequence of forecasts and mean squared error matrices of ξ_t. This means that we also need a value for $P_{1|0}$ to start up the filter.

If ξ_t is covariance stationary, the unconditional covariance matrix $E[\xi_t\xi_t'] = \Sigma$ exists. From the state equation (3.2) we find that:

$$\Sigma = F\Sigma F' + Q. \tag{3.5}$$

The solution to (3.5) is given by

$$\text{vec}(\Sigma) = \left[I_{r^2} - \left(F \otimes F\right)\right]^{-1}\text{vec}(Q), \tag{3.6}$$

where vec is the column stacking operator, and $\otimes$ the Kronecker product. One candidate for $P_{1|0}$ is therefore Σ.

3.3. *Updating ξ_t*

The reason for needing an update of ξ_t can be directly seen from the state equation (3.2). That is, projecting both sides on $\mathcal{Y}_t$ and a constant gives us:

$$\xi_{t+1|t} = F\xi_{t|t}. \tag{3.7}$$

The updated or filtered value of ξ_t relative to its forecasted value is now given by:

$$\xi_{t|t} = \xi_{t|t-1} + P_{t|t-1}H\left[H'P_{t|t-1}H + R\right]^{-1}\left(y_t - y_{t|t-1}\right). \tag{3.8}$$

Substituting (3.8) into (3.7), gives us:

$$\xi_{t+1|t} = F\xi_{t|t-1} + K_t\left(y_t - y_{t|t-1}\right), \tag{3.9}$$

where

$$K_t = FP_{t|t-1}H\left[H'P_{t|t-1}H + R\right]^{-1}, \tag{3.10}$$

[2] The forecasts are optimal in a mean squared error sense among any functions of $(x_t, \mathcal{Y}_{t-1})$.

is known as the *Kalman gain matrix*.

To complete the filter, it remains to calculate $P_{t+1|t}$. There are several ways to determine this matrix, but for DSGE models it appears that the fastest approach is based on the Kalman gain matrix. This updating formula states that

$$P_{t+1|t} = \left(F - K_t H'\right) P_{t|t-1} \left(F - K_t H'\right)' + K_t R K_t' + Q. \tag{3.11}$$

3.4. *Summary of the Kalman Filter*

The Kalman filter makes it possible to calculate the forecast of y_t conditional on $\mathbf{y}_{t-1}$ in equation (3.3) as well as the mean squared error matrix in equation (3.4) for all t using equations (3.9), (3.10), and (3.11) given a set of initial values for $\xi_{1|0}$ and $P_{1|0}$, e.g., $\xi_{1|0} = 0$ and $P_{1|0} = \Sigma$. If ξ_t is covariance stationary, these initial values are natural candidates since the state vector ξ is initialized at its steady state mean and covariance.

However, if r is large then the calculation of Σ in (3.6) may be too cumbersome, especially if this has to be performed frequently (e.g., during the stage of drawing from the posterior). In such cases, it may be better to make use of the Doubling Algorithm. Equation (3.5) is a Lyapunov equation, i.e., a special case of the Sylvester equation. Letting $\gamma_0 = Q$ and $\alpha_0 = F$ we can express the iterations

$$\gamma_k = \gamma_{k-1} + \alpha_{k-1} \gamma_{k-1} \alpha_{k-1}', \quad k = 1, 2, \ldots \tag{3.12}$$

where

$$\alpha_k = \alpha_{k-1} \alpha_{k-1}.$$

The specification in (3.12) is equivalent to expressing:

$$\gamma_k = \sum_{j=0}^{2^k - 1} F^j Q F'^j.$$

From this relation we can see that $\lim_{k \to \infty} \gamma_k = \Sigma$. Moreover, each iteration doubles the number of terms in the sum and we expect the algorithm to converge quickly since $||\alpha_k||$ should be close to zero also for relatively small k when all the eigenvalues of F lie inside the unit circle.

Alternatively, it may be better to let $P_{1|0} = cI_r$ for some constant c and only use values of $y_{t|t-1}$ and its mean squared error matrix for some time period t_m and onwards, where $t_m > 1$. The YADA code takes both these alternatives to using equation (3.6) into account.

3.5. *The Likelihood Function*

The sample log-likelihood function for $y_T, y_{T-1}, \ldots, y_1$ can be expressed as:

$$\ln L\left(\mathbf{y}_T; \theta\right) = \sum_{t=1}^{T} \ln p\left(y_t | x_t, \mathbf{y}_{t-1}; \theta\right), \tag{3.13}$$

by the usual factorization and assumption regarding x_t. To compute the right hand side for a given θ, we need to make some distributional assumptions regarding ξ_1, v_t, and w_t.

In YADA it is assumed that ξ_1, v_t, and w_t are multivariate Gaussian with Q and R being positive semi-definite. This means that:

$$\begin{aligned} \ln p\left(y_t | x_t, \mathbf{y}_{t-1}; \theta\right) = &- \frac{n}{2} \ln(2\pi) - \frac{1}{2} \ln \left| H' P_{t|t-1} H + R \right| + \\ &- \frac{1}{2} \left(y_t - y_{t|t-1}\right)' \left[H' P_{t|t-1} H + R\right]^{-1} \left(y_t - y_{t|t-1}\right). \end{aligned} \tag{3.14}$$

The value of the sample log-likelihood can thus be calculated directly from the 1-step ahead forecasts of y_t and the mean squared error matrix of this forecast.

3.6. *Smoothing of ξ_t*

Since many of the elements of the state vector are given structural interpretations in the DSGE framework, it can be of interest to use as much information as possible to predict this vector. That is, we are concerned with the smooth predictions $\xi_{t|T}$. In equation (3.8) we already have

such a prediction for $t = T$. It thus remains to determine the backward looking predictions for $t < T$.

Hamilton (1994, Chapter 13.6) shows that the smooth predictions of the state vector are given by:

$$\xi_{t|T} = \xi_{t|t} + J_t\big(\xi_{t+1|T} - \xi_{t+1|t}\big), \quad t = 1, \ldots, T-1, \tag{3.15}$$

where the Kalman smoothing matrix J_t is given by

$$\begin{aligned} J_t &= P_{t|t}F'P_{t+1|t}^{-1}, \\ P_{t|t} &= P_{t|t-1} - P_{t|t-1}H\left[H'P_{t|t-1}H + R\right]^{-1}H'P_{t|t-1}. \end{aligned}$$

The mean squared error of the smooth prediction $\xi_{t|T}$ is now:

$$P_{t|T} = P_{t|t} + J_t\big(P_{t+1|T} - P_{t+1|t}\big)J_t'. \tag{3.16}$$

To calculate $\xi_{t|T}$ and $P_{t|T}$ one therefore starts in period $t = T-1$ and then iterates backwards until $t = 1$.

The smoothing expression in (3.15) requires that $P_{t+1|t}$ is a full rank matrix. This assumption is violated if, for instance, $R = 0$ (no measurement errors) and $\text{rank}(Q) < r$ with $P_{1|0} = \Sigma$. The Kalman filter itself is still valid provided that $H'P_{t+1|t}H$ has full rank n. In this case we may use an eigenvalue decomposition such that $P_{t+1|t} = S\Lambda S'$, where S is $r \times s$, $r > s$, Λ is a diagonal full rank $s \times s$ matrix, and $S'S = I_s$. Replacing J_t with

$$J_t^* = P_{t|t}F'S(S'P_{t+1|t}S)^{-1}S',$$

the smoothing algorithm remains otherwise intact.

In fact, smoothing allows us to estimate the shocks to the state equations using as much information as possible. In particular, by projecting both sides of equation (3.2) on the data until period T we find that

$$\nu_{t|T} = \xi_{t|T} - F\xi_{t-1|T}, \quad t = 2, \ldots, T. \tag{3.17}$$

To estimate ν_1 we would need an estimate of $\xi_{0|T}$. A crude approach would be to let $\xi_{0|T} = 0$, i.e., the state variables are assumed to be in steady state at time 0. Alternatively, we could use part of the sample to "train" the Kalman filter and simply let $\xi_{0|T}$ be the last smoothed estimate of the state variables from the training sample. In YADA, $\xi_{0|T} = 0$ unless you have selected a training sample for the state vector. In that case, the latter approach is used.

Similarly, the measurement errors can be estimated using all the sample information once the smoothed state variables have been computed. In this case, we turn to the measurement equation (3.1) and simply let the smoothed estimate of the measurement error be

$$w_{t|T} = y_t - A'x_t - H'\xi_{t|T}, \quad t = 1, \ldots, T. \tag{3.18}$$

If the covariance matrix $R = 0$, then $w_{t|T} = 0$ by construction. Notice that the estimation of the measurement errors does not require any special treatment of $t = 1$.

The measurement equation can also be extended to the case when the measurement matrix H is time varying. With H replaced by H_t in (3.1), the filtering, updating, and smoothing equations are otherwise unaffected as long as H_t is treated as known. YADA is well equipped to deal with a time-varying measurement matrix; see Sections 3.9 and 6.2 for details.

3.7. *Multistep Forecasting*

The calculation of h-steps ahead forecasts of y is now straightforward. From the state equation (3.2) we know that for any $h \geq 1$

$$\xi_{t+h|t} = F\xi_{t+h-1|t}. \tag{3.19}$$

The h-steps ahead forecast of y is therefore:

$$y_{t+h|t} = A'x_{t+h} + H'\xi_{t+h|t}. \tag{3.20}$$

The mean squared error matrix for the $\xi_{t+h|t}$ forecast is simply:

$$\begin{aligned} P_{t+h|t} =& F^h P_{t|t} (F')^h + \sum_{i=0}^{h-1} F^i Q (F')^i \\ =& F P_{t+h-1|t} F' + Q. \end{aligned} \tag{3.21}$$

Finally, the mean squared error matrix for the $y_{t+h|t}$ forecast is:

$$E\left[(y_{t+h} - y_{t+h|t})(y_{t+h} - y_{t+h|t})' | \mathcal{Y}_t\right] = H' P_{t+h|t} H + R. \tag{3.22}$$

The h-steps ahead forecasts of y and the mean squared error matrix can thus be built iteratively from the forecasts of the state variables and their mean squared error matrix.

3.8. *Covariance Properties of the Observed and the State Variables*

It is sometimes of interest to compare the covariances of the observed variables based on the model to the actual data or some other model. It is straightforward to determine these moments provided that the state vector is stationary, i.e., that the eigenvalues of F are inside the unit circle. The unconditional covariance matrix for the state vector is then given by Σ, which satisfies equation (3.5).

From the state equation (3.2) we can rewrite the first order VAR as follows for any non-negative integer h:

$$\xi_t = F^h \xi_{t-h} + \sum_{i=0}^{h-1} F^i v_{t-i}. \tag{3.23}$$

From this it immediately follows that the autocovariance function of the state variables, given the parameter values, based on the state-space model is:

$$E\left[\xi_t \xi_{t-h}'\right] = F^h \Sigma. \tag{3.24}$$

The autocovariance function of the observed variables can now be calculated from the measurement equation (3.1). With $E[y_t] = A' x_t$, it follows that for any non-negative integer

$$E\left[\left(y_t - A' x_t\right)\left(y_{t-h} - A' x_{t-h}\right)'\right] = \begin{cases} H' \Sigma H + R, & \text{if } h = 0, \\ H' F^h \Sigma H, & \text{otherwise.} \end{cases} \tag{3.25}$$

From equations (3.24) and (3.25) it can be seen that the autocovariances tend to zero as h increases. For given parameter values we may, e.g., compare these autocovariances to those obtained directly from the data.

3.9. *YADA Code*

3.9.1. `KalmanFilter & KalmanFilterHt`

The function `KalmanFilter` in YADA computes the value of the log-likelihood function in (3.13) for a given set of parameter values. It requires a $n \times T$ matrix $Y = [y_1 \cdots y_T]$ with the observed variables, a $k \times T$ matrix $X = [x_1 \cdots x_T]$ with exogenous variables, and parameter matrices A, H, F, Q, and R. The F and Q matrices are constructed based on the output from the solution to the DSGE model, while the A, H, and R matrices are specified in a user-defined function that determines the measurement equation; cf. Section 6.2. Moreover, the vector with initial state values $\xi_{1|0}$ is needed. This input is denoted by `KsiInit` and is by default the zero vector.

Furthermore, `KalmanFilter` requires input on the variable `initP`. If this variable is 1, then the function calculates an initial value for the matrix $P_{1|0}$ as described in equation (3.5). If this variable is set to 2, then the doubling algorithm is used to calculate an approximation of Σ (see `DoublingAlgorithmLyapunov` below). Next, the input variables `MaxIter` and `Tolerance` are accepted and are used by the doubling algorithm function. The input variable `StartPeriod` is used to start the sample at period $t_m \geq 1$. The default value of this parameter is 1, i.e., not to skip any observations. Moreover, the boolean variable `AllowUnitRoot` is needed to determine

if undefined unit roots are accepted in the state equation or not. Finally, if `initP` is 3, then $P_{1|0} = cI_r$, where $c > 0$ needs to be specified; its default value is 100.

The function `KalmanFilterHt` takes exactly the same input variables as `KalmanFilter`. While the input variable H is $r \times n$ for the latter function, it is now $r \times n \times T$ for the former function. This means that `KalmanFilterHt` allows for a time-varying measurement matrix.

As output, `KalmanFilter` (`KalmanFilterHt`) provides `lnL`, the value of the log-likelihood function in (3.13), where the summation is taken from t_m until T. Furthermore, output is optionally provided for $y_{t|t-1}$, $H'P_{t|t-1}H + R$ (or $H_t'P_{t|t-1}H_t + R$ when the measurement matrix is time-varying), $\xi_{t|t-1}$, $P_{t|t-1}$, $\ln p(y_t|x_t, \mathcal{Y}_{t-1}; \theta)$ from t_m until T, etc. The dimensions of the outputs are:

`lnL`: scalar containing the value of the log-likelihood function in (3.13).
`status`: boolean variable being 0 if all the eigenvalues of F are inside the unit circle, and 1 otherwise. In the latter case, `KalmanFilter` (`KalmanFilterHt`) sets `initP` to 3.
`lnLt`: $1 \times (T - t_m + 1)$ vector $[\ln p(y_{t_m}|x_{t_m}, \mathcal{Y}_{t_m-1}; \theta) \cdots \ln p(y_T|x_T, \mathcal{Y}_{T-1}; \theta)]$. [Optional]
`Yhat`: $n \times (T - t_m + 1)$ matrix $[y_{t_m|t_m-1} \cdots y_{T|T-1}]$. [Optional]
`MSEY`: $n \times n \times (T - t_m + 1)$ 3 dimensional matrix where $\texttt{MSEY}(:, :, t - t_m + 1) = H'P_{t|t-1}H + R$. [Optional]
`Ksitt1`: $r \times (T - t_m + 1)$ matrix $[\xi_{t_m|t_m-1} \cdots \xi_{T|T-1}]$. [Optional]
`Ptt1`: $r \times r \times (T - t_m + 1)$ 3 dimensional matrix where $\texttt{Ptt1}(:, :, t - t_m + 1) = P_{t|t-1}$. [Optional]

The inputs are given by `Y`, `X`, `A`, `H`, `F`, `Q`, `R`, `KsiInit`, `initP`, `MaxIter`, `Tolerance`, `StartPeriod`, and `c`. All inputs are required by the function. The integer `MaxIter` is the maximum number of iterations that the doubling algorithm can use when `initP` is 2. In this case, the parameter `Tolerance`, i.e., the tolerance value for the algorithm, is also used.

3.9.2. `UnitRootKalmanFilter & UnitRootKalmanFilterHt`

The function `UnitRootKalmanFilter` (`UnitRootKalmanFilterHt`) takes all the input variables that `KalmanFilter` (`KalmanFilterHt`) accepts. In addition, this unit-root consistent version of the Kalman filter needs to know the location of the stationary state variables. This input vector is given by `StationaryPos`. Using this information the function sets up an initial value for the rows and columns of $P_{1|0}$ using the algorithm determined through `initP`. If this integer is 1 or 2, then the rows and columns of F and Q determined by `StationaryPos` are used. The remaining entries of the $P_{1|0}$ are set to zero if off-diagonal and to c if diagonal.

The output variables from `UnitRootKalmanFilter` (`UnitRootKalmanFilterHt`) are identical to those from `KalmanFilter` (`KalmanFilterHt`).

3.9.3. `StateSmoother & StateSmootherHt`

The function `StateSmoother` (`StateSmootherHt`) computes $\xi_{t|t}$, $\xi_{t|T}$, $P_{t|t}$, $P_{t|T}$, and $\xi_{t-1|t}$ using y_t, $y_{t|t-1}$, $\xi_{t|t-1}$ and $P_{t|t-1}$ as well as the parameter matrices H, F, and R as input. The dimensions of the outputs are:

`Ksitt`: $r \times (T - t_m + 1)$ matrix $[\xi_{t_m|t_m} \cdots \xi_{T|T}]$.
`Ptt`: $r \times r \times (T - t_m + 1)$ 3 dimensional matrix where $\texttt{Ptt}(:, :, t - t_m + 1) = P_{t|t}$.
`KsitT`: $r \times (T - t_m + 1)$ matrix $[\xi_{t_m|T} \cdots \xi_{T|T}]$.
`PtT`: $r \times r \times (T - t_m + 1)$ 3 dimensional matrix where $\texttt{PtT}(:, :, t - t_m + 1) = P_{t|T}$.
`Ksit1t`: $r \times (T - t_m + 1)$ matrix with the 1-step smoothed predictions $\xi_{t-1|t}$.

The required inputs are given by `Y`, `Yhat`, `Ksitt1`, `Ptt1`, `H`, `F`, and `R`. For the `StateSmoother` version, the `H` matrix has dimension $r \times n$, while for `StateSmootherHt` is has dimension $r \times n \times T$.

3.9.4. `DoublingAlgorithmLyapunov`

The function `DoublingAlgorithmLyapunov` computes Σ, the unconditional covariance matrix of the state vector ξ, using S ($m \times m$), W ($m \times m$ and positive semi-definite) as inputs as well as the positive integer `MaxIter`, reflecting the maximum number of iterations to perform, and the positive real `ConvValue`, measuring the value when the convergence criterion is satisfied.

The convergence criterion is simply the largest singular value of $|\gamma_{k+1} - \gamma_k|$; see Matlab's `norm` function. The dimensions of the outputs are:

`M`: $m \times m$ positive semi-definite matrix.

`status`: Boolean variable which is 0 if the algorithm converged and 1 otherwise.

When called from `KalmanFilter`, the first two inputs are given by `F` and `Q`, while the maximum number of iterations and the tolerance value for the function can be determined by the user.[3]

4. Solving a DSGE Model with AiM

4.1. *The DSGE Model Specification and Solution*

To make use of the Kalman filter for evaluating the log-likelihood function we need to have a mapping from the structural parameters of the DSGE model to the "reduced form" parameters in the state-space representation. This objective can be achieved by attempting to solve the DSGE model for a given set of parameter values. Given that there exists a unique convergent solution, we can express the solution as a reduced form VAR(1) representation of the form given by the state equation in (3.2).

In this section I will present the general setup for solving linearized rational expectations models with the Anderson-Moore algorithm. This algorithm is implemented in YADA via AiM. Similar to Zagaglia (2005) the DSGE model is expressed as:

$$\sum_{i=1}^{\tau_L} H_{-i} z_{t-i} + H_0 z_t + \sum_{i=1}^{\tau_U} H_i E_t[z_{t+i}] = D\eta_t, \tag{4.1}$$

where $\tau_L > 0$ is the number of lags and $\tau_U > 0$ the number of leads. The z_t $(p \times 1)$ vector are here the endogenous variables, while η_t $(q \times 1)$ are pure innovations, with zero mean and unit variance conditional on the time $t-1$ information. The H_i matrices are of dimension $p \times p$ while D is $p \times q$. When $p > q$ the covariance matrix of $\epsilon_t = D\eta_t$, DD', has reduced rank since the number of shocks is less than the number of endogenous variables.[4]

Adolfson, Laséen, Lindé, and Svensson (2008) shows in some detail how the AiM algorithm can be used to solve the model in equation (4.1) when $\tau_U = \tau_L = 1$. As pointed out in that paper, all linear systems can be reduced to this case by replacing a variable with a long lead or a long lag with a new variable. Consider therefore the system of stochastic difference equations:

$$H_{-1} z_{t-1} + H_0 z_t + H_1 E_t[z_{t+1}] = D\eta_t. \tag{4.2}$$

The AiM algorithm takes the H_i matrices as input and returns B_1, called the convergent autoregressive matrix, and S_0, such that the solution to (4.2) can be expressed as an autoregressive process

$$z_t = B_1 z_{t-1} + B_0 \eta_t, \tag{4.3}$$

where

$$B_0 = S_0^{-1} D, \tag{4.4}$$

$$S_0 = H_0 + H_1 B_1. \tag{4.5}$$

Moreover, the matrix B_1 satisfies the identity

$$H_{-1} + H_0 B_1 + H_1 B_1^2 = 0. \tag{4.6}$$

This can be seen by leading the system in (4.2) one period and taking the expectation with respect to time t information. Evaluating the expectation through (4.3) yields the identity. From equations (4.5) and (4.6) it can be seen that B_1 and S_0 only depend on the H_i matrices, but not on D. This is consistent with the certainty equivalence of the system.

[3] The settings tab in YADA contains options for selecting the doubling algorithm rather than the vectorized solution technique, and for selecting the maximum number of iterations and the tolerance level for the algorithm. The default values are `100` and `1.0e-8`.

[4] The specification of i.i.d. shocks and the matrix D is only used here for expositional purposes. AiM does not make any distinction between endogenous variables and shocks. In fact, z_t would include η_t and, thus, H_0 would include D.

More generally, the conditions for the existence of a unique convergent solution (Anderson and Moore, 1983, 1985) can be summarized as follows:

- *Rank condition*:
$$\text{rank}\left(\sum_{i=-\tau_L}^{\tau_U} H_i\right) = \dim(z).$$
- *Boundedness condition*: For all $\{z_i\}_{i=-\tau_L}^{-1}$ there exists $\{z_t\}_{t=0}^{\infty}$ that solves (4.1) such that
$$\lim_{T\to\infty} E_{t+j}\left[z_{t+j+T}\right] = 0, \quad \forall j \geq 0.$$

The rank condition is equivalent to require that the model has a unique non-stochastic steady state, while the boundedness condition requires that the endogenous variables eventually converge to their steady state values. Given that a unique convergent solution exists, AiM provides an autoregressive solution path

$$z_t = \sum_{i=1}^{\tau_L} B_i z_{t-i} + B_0 \eta_t, \tag{4.7}$$

where $B_0 = S_0^{-1} D$. The VAR(1) companion form of (4.7) is given by

$$\begin{bmatrix} z_t \\ z_{t-1} \\ \vdots \\ z_{t-\tau_L+1} \end{bmatrix} = \begin{bmatrix} B_1 & B_2 & \cdots & B_{\tau_L} \\ I & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & I & 0 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ z_{t-2} \\ \vdots \\ z_{t-\tau_L} \end{bmatrix} + \begin{bmatrix} B_0\eta_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{4.8}$$

With $\xi_t = [z_t \cdots z_{t-\tau_L+1}]'$ the F matrix is immediately retrieved from (4.8), while the state shocks, ν_t, are given by the second term on the right hand side. The Q matrix is equal to the zero matrix, except for the upper left corner which is given by $B_0 B_0'$. If $\tau_L = 1$, then $Q = B_0 B_0'$, while $F = B_1$.

4.2. *YADA Code*

YADA uses only Matlab *functions* for running the AiM procedures, whereas most Matlab implementations include *script* files. The main reason for this change is that the construction of various outputs can more easily be traced to a particular Matlab file when functions are used, while script files tend to hide a lot of variables, most of which are only needed locally. Moreover, inputs are also easier to keep track of, since they can be given local names in a function.

The main functions in YADA for solving the DSGE model and setting up the output as required by the Kalman filter are: `AiMInitialize`, `AiMSolver`, and `AiMtoStateSpace`. A number of other functions are also included for utilizing AiM, but these are not discussed here.[5] It should be noted that the computationally slowest function, `AiMInitialize`, needs only be run once for a given model specification. The other two main functions need to be run for each set of parameter values to be analysed by the code.

4.2.1. `AiMInitialize`

The function `AiMInitialize` runs the AiM parser on `ModelFile`, a text file that sets up the DSGE model in a syntax that the AiM parser can interpret. YADA refers to this file as the AiM model file. If parsing is successful (the `ModelFile` is correctly written and the model is otherwise properly specified), the AiM parser writes two Matlab files to disk. The first is a Matlab function called `compute_aim_data.m`, and the second is a script file called `compute_aim_matrices.m`. The latter file is then internally parsed by `AiMInitialize`, rewriting it as a function that accepts a structure `ModelParameters` as input, where the fields of the structure are simply the parameter names as they have been baptized in the AiM model file, and provides the necessary output. For

[5] Most, if not all, of these Matlab functions originate from the AiM implementation at the Federal Reserve System; see, e.g., Zagaglia (2005).

example, if the model file has a parameter called `omega`, then the structure `ModelParameters` has a field with the same name, i.e., `ModelParameters.omega`.

The functions `compute_aim_data.m` and `compute_aim_matrices.m` are stored on disk in a sub-directory to the directory where the model file is located. By default, the name of this directory depends only on the name of the model specification (which can be different from the AiM model file, since the latter can be shared by many model specifications). `AiMInitialize` therefore also takes the input arguments `NameOfModel` and (optionally) `OutputDirectory`.

`AiMInitialize` also runs the function `compute_aim_data.m` and stores the relevant output from this function in a `mat`-file located in the same directory as the `compute_aim_data.m` file. Finally, `AiMInitialize` provides as output the status of the AiM parsing, and the output given by the `compute_aim_data.m` function. The status variable is 0 when everything went OK; it is 1 if the parsing did not provide the required output; and 2 if the number of data variables did not match the number of stochastic equations. All output variables from `AiMInitialize` are required, while the required input variables are given by `ModelFile`, being a string vector containing the full path plus name and extension of the model file, and `NameOfModel`, a string vector containing the name of the model specification and `OutputDirectory`, the directory where the AiM output is stored. The `NameOfModel` determines the name of the `mat`-file that is created when running the function `compute_aim_data.m`.

4.2.2. `AiMSolver`

The function `AiMSolver` attempts to solve the DSGE model. To this end it requires as inputs the `ModelParameters` structure (containing values for all model parameters), the number of AiM equations (`NumEq`, often being at least $p + q + 1$), the number of lags (`NumLag` being τ_L), the number of leads (`NumLead` being τ_U), and the numerical tolerance for AiM (`AIMTolerance`).

As output the function provides a scalar `mcode` with information about the solvability properties of the DSGE model for the parameter values found in the `ModelParameters` structure. If `mcode` returns 1, then a unique convergent solution exists, while other values reflect various problems with the selected parameters (see the `AiMSolver` file for details).

Given that a unique convergent solution exists, the solution matrices as well as the maximum absolute error (`MaxAbsError`) when computing the solution are calculated. The solution matrices are given by all the B_i's, provided in `BMatrix` ($[B_{\tau_L} \cdots B_1]$), and all the S_j's, returned as the matrix `SMatrix` ($[S_{\tau_L} \cdots S_1 \; S_0]$). These matrices have dimensions `NumEq` $\times \tau_L$`NumEq` and `NumEq` $\times (\tau_L + 1)$`NumEq`, respectively.

4.2.3. `AiMtoStateSpace`

The function `AiMtoStateSpace` creates the F matrix for the state equation (3.2) based on the input matrix `BMatrix`. When $\tau_L > 1$, the function reverses the ordering of the individual B_i matrices in `BMatrix` to reflect the ordering in (4.8) and concatenates the necessary identity and zero matrices. Furthermore, the function creates B_0 from `SMatrix`, extended with zero matrices below if $\tau_L > 1$ such that `BMatrix` and `SMatrix` have the same number of rows.

Since the output from `AiMSolver` treats all equations in a similar fashion, the vectors z_t and η_t are both often included as separate equations. Hence, `NumEq` $\geq p + q$. The additional input variables `StateVariablePositions` and `StateShockPositions` are therefore needed to locate which rows and columns of `BMatrix` and `SMatrix` contain the coefficients on the z and η variables. These input vectors are created with the YADA GUI. Finally, since the ordering of the lags in `BMatrix` is from the largest lag (τ_L) to the lowest (1), the function `AiMReverseCoeffMatrix` is applied to obtain an ordering from the lowest to the largest whenever $\tau_L > 1$.

5. Analysing the Properties of a DSGE Model

There are several ways that you can evaluate an estimated DSGE model in YADA through the behavior of its economic shocks. In this section I will consider a number of tools that are designed for this purpose. Namely, the estimated economic shocks (the η_t's), historical forecast error decompositions, impulse response functions, forecast error variance decompositions, conditional

variance decompositions, conditional correlations, historical observed variable decompositions, and parameter scenarios. Finally, we turn to the issue if a VAR model can uncover the economic shocks and measurement errors of the DSGE model.

5.1. *Estimation of the Economic Shocks*

In Section 3.6 the smooth estimates of the state shocks for the state equation of the Kalman filter were given in equation (3.17). For expositional reasons, let us assume here that $\tau_L = 1$ such that $Q = B_0 B_0'$ with B_0 having full column rank. This latter assumption simply means that we assume that there are no redundant shocks in the DSGE model.

For a given sequence of smoothly estimated state shocks, $\nu_{t|T}$, smooth estimates of the economic shocks are now given by:

$$\eta_{t|T} = \left(B_0' B_0\right)^{-1} B_0' \nu_{t|T}. \tag{5.1}$$

One indication that the DSGE model is well behaved at the underlying parameter estimates is that the economic shocks have close to zero mean and identity covariance matrix. If some estimated economic shocks at, say, the posterior mode of θ are highly correlated this is an indication that the estimated DSGE model is misspecified. What we mean by highly correlated is not crystal clear here, but by assumption the economic shocks should be uncorrelated.

5.2. *Historical Forecast Error Decomposition*

A common tool in the analysis of the structural VAR model is the decomposition of the forecast errors for the observed variables into the underlying structural shocks. For such models, the forecast errors are linear combinations of the structural shocks. In state-space models a forecast error decomposition becomes more delicate since the forecast errors not only depend on the structural, economic shocks, but also on measurement errors and prediction errors of the unobserved state variables. Since the prediction errors of these variables depend on the economic shocks and on the measurement errors, one cannot really speak about a unique decomposition since one may further decompose the variable prediction errors.

Let $\varepsilon_{t+h} = y_{t+h} - y_{t+h|t}$ denote the h-steps ahead forecast error of the observed variables when we condition on the parameters. Furthermore, notice that $y_{t+h} = y_{t+h|T}$ provided that $t+h \leq T$. From the measurement error estimation equation (3.18) and the multistep forecasting equation (3.20) we thus have that for any h such that $t + h \leq T$:

$$\varepsilon_{t+h} = H'\left(\xi_{t+h|T} - \xi_{t+h|t}\right) + w_{t+h|T}, \quad t = 1, \ldots, T-h. \tag{5.2}$$

By making use of equation (3.23) we can rewrite the difference between the smoothed and the h-steps forecasted state vector on the right hand side of (5.2) as

$$\xi_{t+h|T} - \xi_{t+h|t} = F^h\left(\xi_{t|T} - \xi_{t|t}\right) + \sum_{i=0}^{h-1} F^i \nu_{t+h-i|T}.$$

Substituting this expression into (5.2) and noticing that $\nu_{t+h-i|T} = B_0 \eta_{t+h-i|T}$, we obtain the following candidate of a historical forecast error decomposition

$$\varepsilon_{t+h} = H'F^h\left(\xi_{t|T} - \xi_{t|t}\right) + H' \sum_{i=0}^{h-1} F^i B_0 \eta_{t+h-i|T} + w_{t+h|T}, \quad t = 1, \ldots, T-h. \tag{5.3}$$

Unless the state vector can be uniquely recovered from the observed variables[6] the first term on the right hand side is non-zero. It measures the improvement in the prediction of the state vector when the full sample is observed relative to the partial sample. As the forecast horizon h increases, this term converges towards zero. It may be argued that the choice of time period t for the state vector is here somewhat arbitrary. We could, in principle, decompose this term further until we reach period 1. However, the choice of state variable period t for the forecast error is reasonable since this is the point in time when the forecasts are made. Moreover,

[6] This would be the case if the state vector could be expressed as a linear function of the observed variables (and the deterministic) by inverting the measurement equation.

shifting it back to, say, period 1 would mean that the historical forecast error decomposition would include estimates of the economic shocks that are based not only on the full sample, but also on the $\mathcal{Y}_t$ information set and, furthermore, the timing of those shocks would be for time periods $\tau = 2, \ldots, t$, and would therefore be shocks with time period prior to $t + 1$, the first time period of the forecast period.

Apart from the timing of the terms, the decomposition in (5.3) also has the advantage that the dynamics of the state variables enter the second moving average term that involves only the economic shocks, while the measurement errors in the third term do not display any serial correlation. We may regard this as a "model consistent" decomposition in the sense that only the state variables display dynamics and the measurement errors are independent of the state variables. Hence, serial correlation in the forecast errors should stem from the shocks that affect the dynamics, i.e., the economic shocks.

5.3. *Impulse Response Functions*

The responses of the observed variables y_t from shocks to η_t can easily be calculated through the state-space representation and the relationship between the state shocks and the economic shocks. Suppose that $\eta_t = e_j$ and zero thereafter, with e_j being the j:th column of I_q. Hence, we consider the case of a one standard deviation impulse for the j:th economic shock. From the state equation (3.2) the responses in ξ_{t+h} for $h \geq 0$ are:

$$\text{resp}\big(\xi_{t+h} | \eta_t = e_j\big) = F^h B_0 e_j, \quad h \geq 0. \tag{5.4}$$

If the model is stationary, then the responses in the state variables tend to zero as h increases.

From the measurement equation (3.1) we can immediately determine the responses in the observed variables from changes to the state variables. These changes are here given by equation (5.4) and, hence, the responses of the observed variables are:

$$\text{resp}\big(y_{t+h} | \eta_t = e_j\big) = H' F^h B_0 e_j, \quad h \geq 0. \tag{5.5}$$

Again, the assumption that the state variables are stationary implies that the responses of the observed variables tend to zero as the response horizon h increases.

5.4. *Forecast Error Variance Decompositions*

The conditional forecast error covariance matrix for the h-steps ahead forecast of the observed vector y is given in equation (3.22). It can be seen from this equation that this covariance matrix is time-varying. Although this is of interest when we wish to analyse the forecast errors at a particular point in time, the time-variation that the state-space model has introduced is somewhat artificial since it depends only on the choice of $t = 1$. For this reason we may wish to consider "unconditional" forecast error variances, i.e., the expected value of (3.22).[7]

Assume that a unique asymptote of the forecast error covariance matrix $P_{t+h|t}$ exists and let it be denoted by P_h for $h = 0, 1, \ldots$. By equation (3.21) it follows that

$$P_h = F P_{h-1} F' + Q, \quad h \geq 1. \tag{5.6}$$

Similarly, from the expression for $P_{t|t}$ below equation (3.15) we have that

$$P_0 = P_1 - P_1 H \big[H' P_1 H + R\big]^{-1} H' P_1. \tag{5.7}$$

Let $h = 1$ in equation (5.6) and substitute for P_0 from (5.7). We then find that the asymptote P_1 must satisfy

$$P_1 = F P_1 F' - F P_1 H \big[H' P_1 H + R\big]^{-1} H' P_1 F' + Q. \tag{5.8}$$

Given that we can solve for a unique asymptote P_1, all other P_h matrices can be calculated using (5.6) and (5.7).

The assumptions that (i) F has all eigenvalues inside the unit circle, and (ii) Q and R are positive semi-definite, are sufficient for the *existence* of an asymptote, P_1, that satisfies (5.8);

[7] The expected value we shall examine is strictly speaking not unconditional since it depends on the choice of parameter values.

see, e.g., Proposition 13.1 in Hamilton (1994). Let the asymptote for the Kalman gain matrix in (3.10) be denoted by K, where

$$K = FP_1H\left[H'P_1H + R\right]^{-1}.$$

The assumptions (i) and (ii) also imply that all the eigenvalues of $(F - KH')$ lie on or inside the unit circle.

In fact, if we replace (ii) with the stronger assumption that either Q or R is positive definite, then the asymptote P_1 is also *unique*; see Proposition 13.2 in Hamilton (1994). This stronger assumption is in the case of DSGE models often not satisfied since the number of economic shocks tends to be lower than the number of state variables (Q singular) and not all observed variables are measured with error (R singular). Nevertheless, from the proof of Proposition 13.2 in Hamilton it can be seen that the stronger assumption about Q, R can be replaced with the assumption that all the eigenvalues of $(F - KH')$ lie inside the unit circle. From a practical perspective this eigenvalue condition can easily be checked once an asymptote P_1 has been found.

The expression in (5.8) is a discrete algebraic *Riccati equation* and we can therefore try to solve for P_1 using well known tools from control theory. The matrix Q is typically singular for DSGE models since there are usually fewer economic shocks than state variables. Moreover, the matrix R is not required to be of full rank. For these reason, YADA cannot *directly* make use of the function `dare` from the *Control System Toolbox* in Matlab or the procedures discussed by Anderson, Hansen, McGrattan, and Sargent (1996). Instead, YADA uses a combination of iterations (with Σ as an initial value) and Schur decompositions, where a solution to (5.8) is attempted in each iteration using the `dare` function for a reduction of P_1. The details on this algorithm are presented below in Section 5.11.

Prior to making use of such potentially time consuming algorithms it makes sense to first consider a very simple test. From equation (5.6) is can be seen that if $P_0 = 0$ then $P_1 = Q$. This case seems most likely to be relevant when $R = 0$ since we know from (5.7) that P_0 is idempotent under this condition. Nevertheless, the test $P_0 = 0$ for $P_1 = Q$ can be performed very quickly and when successful save considerable computing time.

The forecast error covariance matrix for y_{t+h} can now be calculated by taking the expectation of both sides of equation (3.22) (for a given value of the parameters). This gives us:

$$E\left[\left(y_{t+h} - y_{t+h|t}\right)\left(y_{t+h} - y_{t+h|t}\right)'\right] = H'P_hH + R, \quad h \geq 1. \tag{5.9}$$

It is now relatively straightforward to calculate forecast error variance decompositions for y based on (5.9). The total h-steps ahead forecast error variances are simply the diagonal elements of this matrix. To compute the variance due to all the measurement errors, we let $Q = 0$ and, therefore, $P_h = 0$ for all $h \geq 0$. The forecast error covariance matrix of y_{t+h} is now R for all h when the economic shocks are "shut down". The share of the forecast error variance due to each one of the measurement errors is now the inverse of the diagonal of the right hand side of (5.9) times R. The share explained by all measurement errors is now the sum over the columns of this $n \times n$ matrix.

Similarly, to compute the variance due to economic shock j we let $R = 0$ and $Q = B_{0j}B_{0j}'$, where B_{0j} is the j:the column of B_0. Given these new error covariance matrices we first check if $P_0 = 0$ when we let $P_1 = Q$. If the condition is *not* satisfied we attempt to solve for P_1 using (5.8). Since the $n \times n$ matrix $H'P_1H$ is singular when $\text{rank}(Q) < n$, the inverse of this matrix is replaced by $S(S'[H'P_1H]S)^{-1}S'$, where the $n \times s$ matrix S is computed from a Schur decomposition of $H'P_1H$.[8] Furthermore, Σ is no longer the "best" initial value. Instead we may calculate a new Σ using the new Q, i.e., $B_{0j}B_{0j}'$.

If the Riccati equation has a unique asymptote, P_1, under the new Q, we can compute the new P_h matrices. Next, the shares of the forecast error variance of the observed variables at horizon

[8] Let Λ be a $s \times s$ diagonal matrix whose diagonal elements are the non-zero eigenvalues of $H'P_1H$. The Schur decomposition in YADA is then $S\Lambda S' = H'P_1H$, where $S'S = I_s$. The inverse of $H'P_1H$ is therefore replaced with $S\Lambda^{-1}S'$. Moreover, since $\text{rank}(Q) = 1$ here we expect that $s = 1$.

h for economic shock j can be determined by premultiplying the vector of diagonal elements of the newly computed $H'P_hH$ matrix with the inverse of the diagonal elements of the original forecast error variance matrix of the observed variables at horizon h.

When the model has measurement errors, then the sum of the share due to the measurement errors and the share due to the economic shocks is less than one for finite h. Let P_h^R denote the forecast error covariance matrix of the state variables for any positive semi-definite R, while P_h^0 denotes this covariance matrix when $R = 0$. The right hand side of (5.9) can be written as

$$H'P_h^RH + R = H'P_h^0H + R + H'(P_h^R - P_h^0)H. \tag{5.10}$$

The first term on the right hand side of (5.10) is the share explained by the economic shocks, while the second term is the share explained by the measurement errors. The share of the forecast error variances of the observed variables that cannot *directly* be attributed to these two sources of uncertainty is given by the third term. This additional covariance term is due to the inherent signal extraction problem when trying to separate the influence on the observed variables from measurement errors and from the state variables. Alternatively, the presence of measurement errors makes the information signal in the observed variables for predicting the state variables noisier; cf. the Kalman gain matrix in equation (3.10). Hence, the forecast error variance term $H'(P_h^R - P_h^0)H$ can be interpreted as a signal extraction error variance.

The long-run forecast error variance decomposition can also be calculated using the above relations. First, we note that if the unique asymptote P_1 exists, then all P_h exist and are unique. Moreover, $\lim_{h\to\infty} P_h = \Sigma$ so that the long-run forecast error covariance of the observed variables is simply the unconditional covariance matrix $H'\Sigma H + R$. The long-run forecast error variance decomposition can now be calculated in analogy with the above procedure. The long-run forecast error covariance matrix due to the measurement errors is therefore simply R, while the covariance matrix due to economic shock j is given by $H'\Sigma_jH$, where Σ_j is the solution to the Lyapunov equation:

$$\Sigma_j = F\Sigma_jF' + B_{0j}B_{0j}'.$$

Since $\lim_{h\to\infty} P_h^R = \Sigma$ for any finite R, the signal extraction error variance discussed in the previous paragraph converges to 0 and the long-run forecast error variances can therefore be fully explained by the measurement and the economic shocks.

The h-steps ahead forecast error for the observed variables when the forecasts are performed based on the observations in period T can be expressed as:

$$\varepsilon_{T+h} = H'\left(\xi_{T+h} - \xi_{T+h|T}\right) + w_{T+h}, \quad h = 1, 2, \ldots. \tag{5.11}$$

The covariance matrix of the forecast error is therefore as shown in equation (5.9), with

$$\begin{aligned} \Sigma_{\varepsilon_h} &= E\left[\varepsilon_{T+h}\varepsilon_{T+h}'\right] \\ &= H'P_hH + R. \end{aligned} \tag{5.12}$$

If we assume that all observed variables are expressed as first differences, the levels are given by the accumulation (and an initial condition). This means that the forecast error of the levels is the accumulation of the forecast errors for the observed variables. Letting $\bar{\varepsilon}_{T+h}$ denote the h-steps ahead forecast error of the levels variables, it is given by

$$\begin{aligned} \bar{\varepsilon}_{T+h} &= \sum_{j=1}^{h} \varepsilon_{T+j}, \\ &= \sum_{j=1}^{h} H'\left(\xi_{T+j} - \xi_{t+j|T}\right) + \sum_{j=1}^{h} w_{T+j}. \end{aligned} \tag{5.13}$$

The covariance matrix of the levels h-steps ahead forecast error is denoted by $\Sigma_{\bar{\varepsilon}_h}$. For $h = 1$ we find that $\Sigma_{\bar{\varepsilon}_1} = \Sigma_{\varepsilon_1}$, while a bit of algebra reveals that

$$\Sigma_{\bar{\varepsilon}_h} = \Sigma_{\varepsilon_h} + \Sigma_{\bar{\varepsilon}_{h-1}} + \sum_{i=1}^{h-1} H'\left(F^iP_{h-i} + P_{h-i}\left(F'\right)^i\right)H, \tag{5.14}$$

where the third term on the right hand side is due to the covariance between the state variable errors in period $T+h$ and periods $T+1$ until $T+h-1$, i.e., the covariance between ε_{T+h} and $\bar{\varepsilon}_{T+h-1}$.

5.5. *Conditional Variance Decompositions*

An alternative variance decomposition approach to the forecast error variance decomposition in Section 5.4 is suggested through the historical forecast error decomposition in equation (5.3). Rather than relying on the smooth estimates, the h-steps ahead forecast error using data until period T can be expressed as:

$$\varepsilon_{T+h} = H'F^h\big(\xi_T - \xi_{T|T}\big) + H'\sum_{i=0}^{h-1} F^i B_0 \eta_{T+h-i} + w_{T+h}, \quad h = 1, 2, \ldots, H. \tag{5.15}$$

If we condition the forecast error ε_{T+h} on the state projection error (first term) and the measurement error (third term), the conditional h-steps ahead forecast error variance is given by the variance of the second term on the right hand side. That is,

$$V_h = \sum_{i=0}^{h-1} H'F^i B_0 B_0' (F')^i H. \tag{5.16}$$

This forecast error variance is identical to the forecast error variance that we obtain when a VAR model is written on state-space form. It is therefore analogous to a "variance decomposition" that is calculated from the impulse response functions in (5.5).

Letting $R_i = H'F^iB_0$, the conditional forecast error variance decomposition can be expressed as the $n \times q$ matrix

$$\nu_h = \left[\sum_{i=0}^{h-1}\big(R_i R_i' \odot I_n\big)\right]^{-1}\left[\sum_{i=0}^{h-1}\big(R_i \odot R_i\big)\right], \tag{5.17}$$

where $\odot$ is the Hadamard (element-by-element) product. With $e_i^{(n)}$ being the i:th column of I_n, the share of the h-steps ahead conditional forecast error variance of the i:th observed variable that is explained by the j:th economic shock is given by $e_i^{(n)\prime}\nu_h e_j^{(q)}$.

YADA can also handle conditional variance decompositions for levels variables. To illustrate how this is achieved assume for simplicity that all observed variables are expressed as first differences so that the levels are obtained by accumulating the variables. This means that the h-steps ahead forecast error for the levels is the accumulation of the error in (5.15), i.e.,

$$\bar{\varepsilon}_{T+h} = H'\sum_{j=1}^{h} F^j\big(\xi_T - \xi_{T|T}\big) + H'\sum_{j=1}^{h}\sum_{i=0}^{j-1} F^i B_0 \eta_{T+j-i} + \sum_{j=1}^{h} w_{T+j}, \quad h = 1, 2, \ldots, H. \tag{5.18}$$

The conditional h-steps ahead forecast error for the levels variables is the second term on the right hand side of (5.18). This can be expressed as

$$\begin{aligned}\bar{\varepsilon}_{T+h}^{(c)} &= \sum_{j=1}^{h}\sum_{i=0}^{j-1} R_i \eta_{T+j-i} \\ &= \sum_{j=0}^{h-1} R_j^* \eta_{T+h-j},\end{aligned} \tag{5.19}$$

where $R_j^* = \sum_{i=0}^{j} R_i$. It therefore follows that the conditional forecast error variance for the levels of the observed variables is

$$V_h^* = \sum_{j=0}^{h-1} R_j^* R_j^{*\prime}. \tag{5.20}$$

We can then define the levels variance decomposition as in equation (5.17) with R_j^* instead of R_i (and summing over $j = 0, 1, \ldots, h-1$).

By collecting the products $R_i R_i'$ into one group and all other product into a second group and dividing both sides of equation (5.20) by h, it can be rewritten as:

$$\begin{aligned}\bar{V}_h &= V_h^* / h \\ &= \sum_{i=0}^{h-1} \left(\frac{h-i}{h} \right) R_i R_i' + \sum_{m=1}^{h-1} \sum_{i=0}^{m-1} \left(\frac{h-m}{h} \right) \left(R_i R_m' + R_m R_i' \right).\end{aligned} \tag{5.21}$$

Taking the limit of $\bar{V}_h$ as the forecast horizon approaches infinity we obtain an finite expression of the long-run forecast error covariance. We here find that

$$\begin{aligned}\lim_{h\to\infty} \bar{V}_h &= \sum_{i=0}^{\infty} R_i R_i' + \sum_{m=1}^{\infty} \sum_{i=0}^{m-1} \left(R_i R_m' + R_m R_i' \right) \\ &= \left(\sum_{i=0}^{\infty} R_i \right) \left(\sum_{i=0}^{\infty} R_i \right)'.\end{aligned} \tag{5.22}$$

Hence, if we divide the h-steps ahead forecast error covariance matrix by h and take the limit of this expression, we find that the resulting long-run covariance matrix is equal to the cross product of the accumulated impulse responses.

These results allow us to evaluate how close the forecast error covariance matrix at the h-steps horizon is to the long-run forecast error covariance matrix. The ratio between the l:th diagonal element in (5.21) and in (5.22) is an indicator of such convergence. A value close to unity can be viewed as long-run convergence at forecast horizon h, while a very large or very small value indicates a lack of convergence.

We can also use the result in (5.22) to estimate the long-run conditional forecast error variance decomposition. Letting $R_{lr} = \sum_{i=0}^{\infty} R_i$, we find that

$$\nu_{lr} = \left[R_{lr} R_{lr}' \odot I_n \right]^{-1} \left[R_{lr} \odot R_{lr} \right], \tag{5.23}$$

provides such a decomposition. To approximate the infinite summation in R_{lr} we may simply use a large finite number. YADA here uses the maximum of $\{h, 200\}$.

5.6. *Conditional Correlations*

The basic idea behind conditional correlations is to examine the correlation pattern between a set of variable conditional on one source of fluctuation at a time, e.g., technology shocks. Following the work by Kydland and Prescott (1982) the literature on real business cycle models tended to focus on matching unconditional second moments. This was critized by several economists since a model's ability to match unconditional second moments well did not imply that it could also match conditional moments satisfactorily; see, e.g., Galí (1999).

We can compute conditional correlations directly from the state-space representation. Let column j of B_0 be denoted by $B_{0,j}$, while the j:th economic shock is $\eta_{j,t}$. The covariance matrix for the state variables conditional on only shock j is therefore given by

$$\Sigma_\xi^{(j)} = F \Sigma_\xi^{(j)} F' + B_{0,j} B_{0,j}', \tag{5.24}$$

where $\Sigma_\xi^{(j)} = E[\xi_t \xi_t' | \eta_{j,t}]$. We can estimate $\Sigma_\xi^{(j)}$ at θ by either solving (5.24) analytically through the vec operator, or numerically using the doubling algorithm discussed in Section 3.4. The conditional correlation for the observed variables can thereafter be calculated from the conditional covariance matrix

$$\Sigma_y^{(j)} = H' \Sigma_\xi^{(j)} H. \tag{5.25}$$

As an alternative to conditional population moments we can also consider simulation methods to obtain estimates of conditional sample moments. In that case we can simulate a path for the state variables conditional on only shock j being nonzero, by drawing T values for $\eta_{j,t}$ and letting

$$\xi_t^{(s)} = F \xi_{t-1}^{(s)} + B_{0,j} \eta_{j,t}, \quad t = 1, \ldots, T. \tag{5.26}$$

where $\xi_0^{(s)}$ is drawn from $N(0, \Sigma_\xi^{(j)})$. The conditional sample correlations for simulation s can now be computed as:

$$\hat{\Sigma}_y^{(j,s)} = \frac{1}{T}\sum_{t=1}^{T} H'\xi_t^{(s)}\xi_t^{(s)\prime}H, \quad s = 1, \ldots, S. \tag{5.27}$$

By repeating the simulations S times we can estimate the distribution of the conditional sample correlations for a given θ.

5.7. *Historical Observed Variable Decomposition*

Given the smooth estimates of the economic shocks (5.1) and the measurement errors (3.18) we can also calculate a historical decomposition of the observed variables. Specifically, we know that

$$y_t = A'x_t + H'\xi_{t|T} + w_{t|T}.$$

The estimated share of y_t due to measurement errors is therefore simply $w_{t|T}$, while the shares due to the various economic shocks need to be computed from the smoothed state variables and the state equation. Specifically, we have that

$$\xi_{t|T} = F^t\xi_{0|T} + \sum_{i=0}^{t-1} F^i B_0 \eta_{t-i|T}, \tag{5.28}$$

where $\xi_{0|T} = 0$ when the observed data begins in period $t = 1$, while $\eta_{1|T} = (B_0'B_0)^{-1}B_0'(\xi_{1|T} - F\xi_{0|T})$. From (5.28) we can decompose the smooth estimates of the state variables into terms due to the q economic shocks. Substituting this expression into the measurement equation based on smoothed estimates, we obtain

$$y_t = A'x_t + H'F^t\xi_{0|T} + \sum_{i=0}^{t-1} H'F^iB_0\eta_{t-i|T} + w_{t|T}, \quad t = 1, \ldots, T. \tag{5.29}$$

Accordingly, it is straightforward to decompose the observed variables into terms determined by (i) the deterministic variables, (ii) the initial state estimate ($\xi_{0|T}$), (iii) the q economic shocks, and (iv) the measurement errors.

The decomposition in (5.29) can also be generalized into decompositions for all possible subsamples $\{t_0 + 1, \ldots, T\}$, where $t_0 = 0, 1, \ldots T - 1$. In the decomposition above the choice is $t_0 = 0$. The generalization into an arbitrary t_0 gives us:

$$y_t = A'x_t + H'F^{t-t_0}\xi_{t_0|T} + \sum_{i=0}^{t-t_0-1} H'F^iB_0\eta_{t-i|T} + w_{t|T}, \qquad t = t_0 + 1, \ldots, T. \tag{5.30}$$

This provides a decomposition of the observed variables y_t into (i) deterministic variables, (ii) the estimated history of the state until t_0 ($\xi_{t_0|T}$), (iii) the q economic shocks over the period after t_0 and up until t, and (iv) the measurement error.

5.8. *Parameter Scenarios*

Parameter scenarios are used to examine the impact that changing some parameters has on the behavior of the variables or on the economic shocks. Let θ_b be the baseline value of the parameter vector and θ_a the alternative value. The baseline value can, for example, be the posterior mode estimate of θ.

Assuming that the model has a unique and convergent solution at both θ_b and at θ_a, YADA provides two approaches for parameter scenario analysis. The first is to calculate smooth estimates of the economic shocks under the two parameter vectors. The path for the observed variables are, in this situation, the same for both parameter vectors.[9]

The second approach takes the economic shocks and measurement errors based on θ_b as given and calculates the implied observed variables from the state-space representation with

[9] Alternatively, one may wish to compare smooth estimates of the state variables under the two parameter vectors.

the parameter matrices A, H, F and B_0 determined by the alternative θ_a vector.[10] This path can then be compared with the actual data for the observed variables.

5.9. *Linking the State-Space Representation to a VAR*

To address the issue if the economic shocks and measurement errors of the state-space representation can be uncovered from a VAR representation of the observed variables, Fernández-Villaverde, Rubio-Ramírez, Sargent, and Watson (2007) provide a simple condition for checking this.

To cast equations (3.1) and (3.2) into their framework we first rewrite the measurement error as:

$$w_t = \Phi\omega_t,$$

where $R = \Phi\Phi'$ while $\omega_t \sim N(0, I)$. The matrix Φ is of dimension $n \times m$, with $m = \text{rank}(R)$. The random vector ω_t thus gives the *unique* measurement errors.

Substituting for ξ_t from the state equation into the measurement equation we get:

$$\begin{aligned} y_t &= A'x_t + H'F\xi_{t-1} + H'B_0\eta_t + \Phi\omega_t \\ &= A'x_t + H'F\xi_{t-1} + D\varphi_t, \end{aligned} \tag{5.31}$$

where the residual vector $\varphi_t = [\eta_t' \;\; \omega_t']'$ while $D = [H'B_0 \;\; \Phi]$ is of dimension $n \times (q + m)$.

The state equation can likewise be expressed as:

$$\xi_t = F\xi_{t-1} + B\varphi_t, \tag{5.32}$$

where $B = [B_0 \;\; 0]$ is an $r \times (q + m)$ matrix.

The state-space representation has a VAR representation when φ_t can be retrieved from the history of the observed variables. The first condition for this is that D has rank $q + m$ such that a Moore-Penrose inverse $D^+ = (D'D)^{-1}D'$ exists. A necessary condition for the existence of this inverse is clearly that $n \geq q + m$, i.e., that we have at least as many observed variables as economic shocks and unique measurement errors.

Assuming D^+ exists we can write φ_t in (5.31) as a function of y_t, x_t, and ξ_{t-1}. Substituting the corresponding expression into the state equation and rearranging terms yields

$$\xi_t = \left(F - BD^+H'F\right)\xi_{t-1} + \left(y_t - A'x_t\right). \tag{5.33}$$

If the matrix $F - BD^+H'F$ has all eigenvalues inside the unit circle, then the state variables are uniquely determined by the history of the observed (and the exogenous) variables. The state vector ξ_t can therefore be regarded as known, and, moreover, this allows us to express the measurement equation as an infinite order VAR model. Accordingly, the economic shocks and the measurement errors are uniquely determined by the history of the observed data (and the parameters of the DSGE model). The eigenvalue condition is called a "poor man's invertibility condition" by Fernández-Villaverde et al. (2007).

5.10. *Fisher's Information Matrix*

If the vector of parameters θ is estimated with maximum likelihood, then the inverse of Fisher's information matrix is the asymptotic covariance matrix for the parameters. If this matrix has full rank when evaluated at the true parameter values, the parameters are said to be locally identified; cf. Rothenberg (1971).

Since DSGE models are typically estimated with Bayesian methods, identification problems can likewise be viewed through the behavior of the Hessian of the log-posterior distribution. However, such problems can be dealt with by changing the prior such that the log-posterior has more curvature. Still, use of prior information to deal with identification problems is unsatisfactory. One way to examine how much information there is in the data about a certain parameter

[10] One alternative to the second approach is to simulate the model under θ_a by drawing the economic shocks and the measurement errors from their assumed distribution a large number of times, compute the implied path for the observed variables, and then compare, say, the average of these paths to the actual data for the observed variables.

is to compare the plots of the prior and the posterior distributions. If these distributions are very similar, then it is unlikely that the data is very informative about this particular parameter.

The comparison between prior and posterior distributions require that we have access to draws from the posterior. Since drawing from the posterior may be very time consuming, it may be useful to consider an alternative approach. In this respect, Fisher's information matrix may also be useful when considering identification issues from a Bayesian perspective. This approach has been investigated in a series of articles by Nikolay Iskrev; see Iskrev (2006, 2007a, 2007b).[11]

Let $\tilde{y}_t = y_t - y_{t|t-1}$ and $B_t = H'P_{t|t-1}H + R$. Using standard results from matrix differential algebra (see Magnus and Neudecker, 1988) it has been shown by Klein and Neudecker (2000) that the second differential of the log-likelihood function in (3.13) can be written as:

$$\begin{aligned} d^2 \ln L(\mathbf{y}_T;\theta) = \frac{1}{2}\sum_{t=1}^{T} \mathrm{tr}\Big\{ B_t^{-1}\big(dB_t\big)B_t^{-1}\big(dB_t\big)\Big\} - \sum_{t=1}^{T}\big(d\tilde{y}_t\big)'B_t^{-1}\big(d\tilde{y}_t\big) + \\ - \sum_{t=1}^{T} \mathrm{tr}\Big\{ B_t^{-1}\big(dB_t\big)B_t^{-1}\big(dB_t\big)B_t^{-1}\tilde{y}_t\tilde{y}_t'\Big\} + \\ + 2\sum_{t=1}^{T} \mathrm{tr}\Big\{ B_t^{-1}\big(dB_t\big)B_t^{-1}\big(d\tilde{y}_t\big)\tilde{y}_t'\Big\} - \sum_{t=1}^{T} \mathrm{tr}\Big\{ B_t^{-1}\big(d^2\tilde{y}_t\big)\tilde{y}_t'\Big\}. \end{aligned}$$

Taking the expectation of both sides with respect to θ, the Lemma in Klein and Neudecker implies that the last two terms on the right hand side are zero. Moreover, with $E[\tilde{y}_t\tilde{y}_t';\theta] = E[B_t;\theta]$, the above simplifies to

$$\begin{aligned} E_\theta\Big[d^2 \ln L(\mathbf{y}_T;\theta)\Big] = -\frac{1}{2}\sum_{t=1}^{T} E_\theta\Big[\Big(d\mathrm{vec}\big(B_t\big)\Big)'\Big[B_t^{-1}\otimes B_t^{-1}\Big]d\mathrm{vec}\big(B_t\big)\Big] + \\ - \sum_{t=1}^{T} E_\theta\Big[\big(d\tilde{y}_t\big)'B_t^{-1}d\tilde{y}_t\Big] \end{aligned}$$

The matrix B_t is a differentiable function of the parameters θ such that

$$d\mathrm{vec}\big(B_t\big) = \frac{\partial \mathrm{vec}\big(B_t\big)}{\partial\theta'}d\theta.$$

Similarly, we let

$$d\tilde{y}_t = \frac{\partial \tilde{y}_t}{\partial\theta'}d\theta.$$

Collecting these results, the Fisher's information matrix may be expressed as:

$$\begin{aligned} -E_\theta\left[\frac{\partial^2 \ln L(\mathbf{y}_T;\theta)}{\partial\theta\partial\theta'}\right] = \sum_{t=1}^{T} E_\theta\left[\Big(\frac{\partial\tilde{y}_t}{\partial\theta'}\Big)'B_t^{-1}\frac{\partial\tilde{y}_t}{\partial\theta'}\right] + \\ + \frac{1}{2}\sum_{t=1}^{T} E_\theta\left[\Big(\frac{\partial \mathrm{vec}\big(B_t\big)}{\partial\theta'}\Big)'\Big[B_t^{-1}\otimes B_t^{-1}\Big]\frac{\partial \mathrm{vec}\big(B_t\big)}{\partial\theta'}\right]. \end{aligned} \tag{5.34}$$

The partial derivatives of $\tilde{y}_t$ and B_t with respect to the *reduced form* parameters (A, H, R, F, Q) can be determined analytically. The form depends on how the initial conditions for the state variables relate to the parameters; see Zadrozny (1989, 1992) for details. The step from reduced form parameters to θ is explained by Iskrev (2007a). Instead of making use of these analytic results, YADA currently computes numerical derivatives of $\tilde{y}_t$ and B_t with respect to θ.

5.11. *The Riccati Equation Solver Algorithm in YADA*

The algorithm used by YADA to (try to) solve the Riccati equation (5.8) for the forecast error variances uses a combination of iterative and non-iterative techniques. Let the Riccati equation

[11] See also Beyer and Farmer (2004) and Canova and Sala (2006) for discussions of identifiability issues in DSGE models.

be given by

$$P = FPF' - FPH\left[H'PH + R\right]^{-1}H'PF' + Q, \tag{5.35}$$

where Q and R are positive semi-definite and P is used instead of P_1. Below I will discuss the main ingredients of the algorithm, where each iteration follows the same steps.

First, a positive semi-definite value of P is required. This value is used to evaluate the right hand side of (5.35), yielding a new value for P that we shall explore.[12] Since the new value of P may have reduced rank, we first use a Schur decomposition such that for the $r \times r$ positive semi-definite matrix P

$$P = N\Lambda N',$$

where N is $r \times s$ such that $N'N = I_s$, while Λ is an $s \times s$ diagonal matrix with the non-zero eigenvalues of P. Substituting this expression for P into (5.35), premultiplying both sides by N' and postmultiplying by N, we obtain the new Riccati equation

$$\Lambda = A\Lambda A' - A\Lambda B\left[B'\Lambda B + R\right]^{-1}B'\Lambda A' + C, \tag{5.36}$$

where $A = N'FN$, $B = N'H$, and $C = N'QN$.

Next, if the matrix $B'\Lambda B + R$ in (5.36) has reduced rank, YADA performs a second Schur decomposition. Specifically,

$$B'\Lambda B + R = D\Gamma D',$$

where D is $n \times d$ such that $D'D = I_d$, while Γ is a $d \times d$ diagonal matrix with the non-zero eigenvalues of $B'\Lambda B + R$. Replacing the inverse of this matrix with $D(B^{*\prime}\Lambda B^* + R^*)^{-1}D$, with $B^* = BD$ and $R^* = D'RD$, the Riccati equation (5.36) can be rewritten as

$$\Lambda = A\Lambda A' - A\Lambda B^*\left[B^{*\prime}\Lambda B^* + R^*\right]^{-1}B^{*\prime}\Lambda A' + C. \tag{5.37}$$

When the matrix $B'\Lambda B + R$ in (5.36) has full rank, YADA sets $D = I_n$.

YADA now tries to solve for Λ in (5.37) using `dare` from the *Control System Toolbox*; see Arnold and Laub (1984) for details on the algorithm used by `dare`. If `dare` flags that a unique solution to this Riccati equation exists, Λ^*, then YADA lets $P = N\Lambda^* N'$. When the call to `dare` does not yield a unique solution, YADA instead compares the current P to the previous P. If the difference is sufficiently small it lets the current P be the solution. Otherwise, YADA uses the current P as input for the next iteration.

5.12. *YADA Code*

The impulse responses are handled by the function `DSGEImpulseResponseFcn`, historical forecast error decompositions by `DSGEHistDecompFcn`, while the variance decompositions are calculated by the function `DSGEVarianceDecompFcn` for the original data and for the levels by `DSGEVarianceDecompLevelsFcn`. Since the latter depends on solving Riccati equations, the code also includes the function `RiccatiSolver`. The conditional correlations are performed by the function `DSGEConditionalCorrsTheta` that can deal with both population-based and sample-based correlations.

The conditional variance decompositions are handled by `DSGECondVarianceDecompFcn`, while output on estimates of unobserved variables and observed variable decompositions is provided by the function `CalculateDSGEStateVariables`. The levels of the conditional variance decompositions are handled by `DSGECondLevVarianceDecompFcn`, while the levels of the impulse response functions are taken care of by `DSGELevImpulseResponseFcn`. The function `DSGEtoVARModel` checks the "poor man's invertibility condition" of Fernández-Villaverde et al. (2007), i.e., if all the eigenvalues of the matrix on lagged states in equation (5.33) lie inside the unit circle. Annualizations of the conditional variance decompositions are computed by `DSGECondAnnVarianceDecompFcn`, while impulse responses for annualized data is calculated directly from the output of `DSGEImpulseResponseFcn`. With s being the data frequency, this typically involves summing s consecutive impulse responses provided that the variable is annualized

[12] In the event that $H'PH + R$ has reduced rank, its "inverse" is replaced by $S(S'[H'PH + R]S)^{-1}S'$ where S is obtained from the Schur decomposition $H'PH + R = S\Gamma S'$.

by summing s consecutive observations. For response horizons prior to period $s-1$ all responses from period 0 until the response horizon are summed. Finally, we shall return to Fisher's information matrix from Section 5.10. The function used for this purpose is `DSGEInformationMatrix` which can estimate the information matrix for any selected value of θ.

5.12.1. `DSGEImpulseResponseFcn`

The function `DSGEImpulseResponseFcn` is used to calculate the responses in the state variables and the observed variables from the economic shocks. It provides as output the structure `IRStructure` using the inputs `H`, `F`, `B0`, and `h`. The $r \times n$ matrix `H` is given by the measurement equations, while `F` and `B0` are obtained from the DSGE model solution as determined by the function `AiMtoStateSpace`. The last input `h` is a positive integer denoting the maximum horizon for the impulse responses.

The output structure `IRStructure` has two fields `Ksi` and `Y`. The fields contain the responses of the state variables and of the observed variables, respectively. These are provided as 3D matrices. The first dimension is the number of states (r) for the field `Ksi` and observed variables (n) for `Y`, the second dimension is the number of shocks (q), and the third is the number of responses plus one ($h + 1$). First instance, `IRStructure.Y(:,:,i+1)` hold the results for response horizon i.

5.12.2. `DSGELevImpulseResponseFcn`

The function `DSGELevImpulseResponseFcn` is used to calculate the accumulated responses in the state variables and the levels responses of the observed variables from the economic shocks. It provides as output the structure `IRStructure` using the inputs `H`, `F`, `B0`, `AccMat`, and `h`. The $r \times n$ matrix `H` is given by the measurement equations, while `F` and `B0` are obtained from the DSGE model solution as determined by the function `AiMtoStateSpace`. The matrix `AccMat` is an $n \times n$ diagonal 0-1 matrix. It is used to accumulate the responses in the observed variables provided that they are viewed as being in first differences. The last input `h` is a positive integer denoting the maximum horizon for the impulse responses.

The output structure `IRStructure` has the same dimensions as for the original data function `DSGEImpulseResponseFcn`. The fields contain the responses of the state variables (`Ksi`) and of the observed variables (`Y`), respectively. While the responses in the state variables are pure accumulations of the response function in (5.4), the levels response for the observed variables are only accumulated for those variables which are viewed as being in first differences. Specifically, with A being the 0-1 diagonal matrix `AccMat`, the levels responses for the observed variables are given by

$$\text{resp}\big(y^L_{t+h}|\eta_t = e_j\big) = A \cdot \text{resp}\big(y^L_{t+h-1}|\eta_t = e_j\big) + H'F^hB_0e_j, \quad h \geq 1,$$

where $\text{resp}(y^L_t|\eta_t = e_j) = H'B_0e_j$. Observed variables are viewed by YADA as being in first differences based on the user defined input in the *Data Construction File*; cf. Section 6.1.

5.12.3. `CalculateDSGEStateVariables`

The function `CalculateDSGEStateVariables` provides output on estimates of various unobserved variables. To achieve this it needs 5 inputs: `theta`, `thetaPositions`, `ModelParameters`, `DSGEModel`, and `ObsVarDec`. The first three inputs are discussed in some detail in connection with the prior file handling function `VerifyPriorData`, while the structure `DSGEModel` is discussed in connection with the posterior mode estimation function `PosteriorModeEstimation`; cf. Section 8.2. The last input, `ObsVarDec`, is a boolean variable that determines if the function should compute the observed variable decompositions or not. This input is optional and defaults to 0 if not supplied.

The only required output from the function is the structure `StateVarStructure`. In addition, the function can provide output on `status`, the `mcode` output from the `AiMSolver` function, and `kalmanstatus`, the `status` output from the `KalmanFilter` function. The latter two outputs are only taken into account by YADA when initial parameter estimates are given to `CalculateDSGEStateVariables`.

The structure StateVarStructure has at most 26 fields. First of all, the fields Y and X hold the data matrices for the observed variables and the exogenous variables for the actual sample used by the Kalman filter. Furthermore, the field TrainSample hold a boolean variable which reports if a training sample was used or not when computing the log-likelihood through the Kalman filter. Furthermore, the output on the state variable estimates are given through the fields Ksitt1 (forecast), Ksitt (update), and KsitT (smooth), while the forecasted observed variables are held in the field Yhat. Next, the field lnLt stores the vector with sequential log-likelihood values, i.e., the left hand side of equation (3.14).

The smooth estimates of the economic shocks are located in the field etatT. The matrix stored in this field has the same number of rows as there are shocks with a non-zero impact on at least one variable. This latter issue is determined by examining the estimated B_0 matrix by removing the columns that are zero. The columns that are non-zero are stored as a vector in the field KeepVar. Update estimates of the economic shocks are located in etatt. If the model contains measurement errors, then the smoothed estimates of the non-zero measurement errors are located in the field wtT, while names of equations for these non-zero measurement errors are stored as a string matrix in the field wtNames. At the same time, all estimated measurement errors are kept in a matrix in the field wthT. Similarly, update estimates of the measurement errors are found in wtt and wtht.

Given that the boolean input ObsVarDec is unity, the historical observed variable decompositions are calculated. The field XiInit contains a matrix with typical column element given by $H'F^t\xi_{0|T}$. Similarly, the field etaDecomp holds a 3D matrix with the contributions to the observed variables of the non-zero economic shocks. For instance, the contribution of shock i for observed variable j can be obtained for the full sample as etaDecomp(j,:,i), a $1 \times T$ vector. The historical decompositions for the state variables are similarly handled by the fields XietaInit and XietaDecomp.

The structure StateVarStructure also has 5 fields with parameter matrices A, H, and R from the measurement equation, and F and B0 from the state equation. The last field is given by MaxEigenvalue, which, as the name suggests, holds the largest eigenvalue (modulus) of the state transition matrix F.

5.12.4. DSGEHistDecompFcn

The function DSGEHistDecompFcn calculates the historical forecast error decomposition of the h-steps ahead forecast errors as described in equation (5.3). The inputs for the function are given by Y, X, A, H, F, B0, Ksitt, KsitT, etatT, wtT, and h. As before, Y and X are the data on the observed variables and the exogenous variables, respectively, A and H are given by the measurement equation, while F and B0 are obtained from the AiMtoStateSpace function regarding the state equation. Furthermore, Ksitt and KsitT are the updated and smoothed state variables that are prepared by the StateSmoother function. The input etatT is the smoothed estimate of the economic shocks in equation (5.1), wtT is the smoothed estimate of the measurement error in (3.18), while h is the forecast horizon h.

The function supplies the structure HistDecompStructure as output. This structure has 5 fields: epstth, an $n \times (T-h)$ matrix with the forecast errors for the observed variables; KsiErr, an $n \times (T-h)$ matrix with the state projection error in the first term of the right hand side of equation (5.3); etathT, $n \times (T-h) \times q$ matrix with the shares of the q economic shocks in the second term of the equation;[13] wthT, an $n \times (T-h)$ matrix with the smoothed measurement errors; and KeepVar, a vector with index values of the columns of B_0 that are non-zero.

5.12.5. DSGEConditionalCorrsTheta

The function DSGEConditionalCorrsTheta can calulate either population-based or sample-based conditional correlations as described in Section 5.6. It takes 10 input variables: theta, thetaPositions, ModelParameters, NumPaths, EstType, DSGEModel, CurrINI, SimulateData,

[13] This means that the part of the h-steps ahead forecast error that is due to economic shock j is obtained from etathT(:,:,j), an $n \times (T-h)$ matrix.

FirstPeriod, LastPeriod. The structures thetaPositions, ModelParameters, DSGEModel, and CurrINI have all been discussed above. The vector theta holds the values for the parameters as usual. The integer NumPaths is equal to the number of simulations and is used only if sample-based conditional correlation should be calculated. The text string EstType indicates if posterior mode or initial parameter values are used. The boolean variable SimulateData is 1 (0) if sample-based (population-based) conditional correlations should be computed. Finally, the integers FirstPeriod and LastPeriod marks the sample start and end point when teh sample-based approach should be used.

The function provides one required and one optional output. The required output variable is CondCorr, a structure with fields Mean, Quantiles, and ShockNames. When the field Quantiles is not empty, then it has length equal to the number of quantiles, and each sub-entry has fields percent and Mean. The former stores the percentile value of the distribution, while th latter stores the conditional correlations at that percentile. The optional output variable is status that indicates if the solution to the DSGE model is unique or not. The value is equal to the variable mcode given by the function AiMSolver.

5.12.6. DSGEParameterScenariosTheta

The function DSGEParameterScenariosTheta calculates the parameter scenario for two values of the parameter vector, the baseline value and the alternative value. It takes 10 input variables: DSGEModel, theta, thetaScenario, thetaPositions, ModelParameters, FirstPeriod, LastPeriod, BreakPeriod, CopyFilesToTmpDir, and finally CurrINI. The structures DSGEModel, ModelParameters, thetaPositions and CurrINI have all been discussed above. The vector theta holds the baseline values of the parameters, while thetaScenario holds the alternative (scenario) values of the parameters. The integers FirstPeriod and LastPeriod simply indicate the first and the last observation in the estimation sample (not taking a possible training sample for the state variables into account). The integer BreakPeriod indicates the position in the sample (taking the training sample into account) where the parameters change, while the boolean CopyFilesToTmpDir indicates if certain files should be copied to the tmp directory of YADA or not.

The function provides 8 required output variables. These are: Status, a boolean that indicates if all calculations were completed successfully or not. Next, the function gives the actual path for the observed variables in the matrix Y, as well as the matrix YScenario, holding the alternative paths. As mentioned in Section 5.8, these paths are based on feeding the smooth estimates of the economic shocks (and measurement errors) based on the baseline parameters into the state-space model for the alternative parameters. Next, the function gives two matrices with smooth estimates of the economic shocks: OriginalShocks and ScenarioShocks. The former holds the values of the economic shocks under the baseline parameter values, while the latter gives the values of the economic shocks under the alternative parameter values. Similarly, two matrices with state variable estimates are provided: OriginalStates and ScenarioStates, where the former holds the smooth estimates of the state variables for the baseline parameter values, while the latter matrix holds the implied state variables for the alternative parameter values. That is, when the state equation is applied to the combination of the smoothly estimated economic shocks under the baseline parameter values along and the F and B_0 matrices for the alternative parameter values. Finally, the function provides a vector with positive integers, KeepShocks, signalling which of the economic shocks have a non-zero influence on the variables of the DSGE model.

5.12.7. DSGEtoVARModel

The function DSGEtoVARModel is used to check if the state-space representation of the DSGE model satisfies the "poor man's invertibility condition" of Fernández-Villaverde et al. (2007). The function takes 4 inputs: H, R, F, and B0. These are, as before, the matrices H and R from the measurement equation, and the matrices F and B_0 of the state equations; see, e.g., the details on DSGEImpulseResponseFcn.

As output the function provides `status` and `EigenValues`. The integer `status` is unity if the state-space model can be rewritten as a VAR model, and 0 if some eigenvalues is on or outside the unit circle. In the event that the number of economic shocks and unique measurement errors exceeds the number of observed variables, `status` is equal to -1. The vector `EigenValues` provides the modulus of the eigenvalues from the invertibility condition when `status` is non-negative.

5.12.8. `DSGEInformationMatrix`

The function `DSGEInformationMatrix` is used to estimate Fisher's information matrix as it is given in equation (5.34). To achieve this 6 input variables are required: `theta`, `thetaPositions`, `ModelParameters`, `ParameterNames`, `DSGEModel`, and `CurrINI`. The first 3 variables are identical to the input variables with the same names in the `CalculateDSGEStateVariables` function. The 4th input is a string matrix with the names of the estimated parameters. The last two input variables are structures that have been mentioned above; see e.g. Section 8.2.

The function provides the output variable `InformationMatrix` which is an estimate of the right hand side in (5.34) with the partial derivatives $\partial\tilde{y}_t/\partial\theta'$ and $\partial\text{vec}(B_t)/\partial\theta'$ replaced by numerical partials. By default, each parameter change is equal to 0.1 percent of its given value. If the model cannot be solved at the new value of θ, YADA tries a parameter change of 0.01 percent. Should YADA also be unsuccessful at the second new value of θ, estimation of the information matrix is aborted.

5.12.9. `DSGECondVarianceDecompFcn`

The function `DSGECondVarianceDecompFcn` computes the conditional forecast error variance decomposition in (5.17). The function needs 4 inputs: `H`, `F`, `B0`, and `h`. These are exactly the same as those needed by `DSGEImpulseResponseFcn`.

As output the function provides the 3D matrix `FEVDs`. This matrix has dimension $n \times q \times h$, with n being the number of observed variables, q the number of economic shocks, and h the forecast horizon.

5.12.10. `DSGECondLevVarianceDecompFcn`

The function `DSGECondLevVarianceDecompFcn` computes the conditional forecast error variance decomposition in (5.17), but where R_i is partly an accumulation of R_{i-1}. Specifically, let A denote a diagonal 0-1 matrix. The R_i matrices are here calculated according to

$$R_i = AR_{i-1} + H'F^iB_0, \quad i = 1, 2, \ldots,$$

while $R_0 = H'B_0$. This allows YADA to compute levels effects of observed variables that only appear in first differences in the y_t vector, e.g., GDP growth. At the same time, variables that already appear in levels, e.g., the nominal interest rate, are not accumulated. The function needs 5 inputs: `H`, `F`, `B0`, `AccMat`, and `h`. These are identical to the inputs accepted by `DSGELevImpulseResponseFcn`.

As output the function provides the 3D matrix `FEVDs`. This matrix has dimension $n \times q \times h$, with n being the number of observed variables, q the number of economic shocks, and h the forecast horizon.

5.12.11. `DSGECondAnnVarianceDecompFcn`

The function `DSGECondAnnVarianceDecompFcn` calculates the conditional variance decomposition in (5.17), but where is partly and accumulation of lagged R_i's. In particular, let A denote a diagonal 0-1 matrix, while s is the frequency of the data, e.g., $s = 4$ ($s = 12$) for quarterly (monthly) data. The R_i matrices are now computed from as:

$$R_i = A\left(\sum_{j=1}^{\min\{i,s\}-1} H'F^{i-j}B_0\right) + H'F^iB_0, \quad i = 0, 1, 2, \ldots.$$

A diagonal element of A is unity if the corresponding observed variable should be annualized by adding the current and previous $s-1$ observations, and zero otherwise.

The function needs 6 inputs: `H`, `F`, `B0`, `AccMat`, `h`, and `AccHorizon`. The first five are identical to the inputs accepted by `DSGECondLevVarianceDecompFcn`, with the exception of `AccMat` which is now given by the matrix A from the equation above. The final input is `AccHorizon` which corresponds to the integer s above.

As output the function provides the 3D matrix `FEVDs`. This matrix has dimension $n \times q \times h$, with n being the number of observed variables, q the number of economic shocks, and h the forecast horizon.

5.12.12. `DSGEVarianceDecompFcn & DSGEVarianceDecompLevelsFcn`

The function `DSGEVarianceDecompFcn` computes all the forecast error variance decompositions for the original variables, while `DSGEVarianceDecompLevelsFcn` takes care of the levels. Both functions accept the same input variables and return the same output variables.

They require the 9 input variables, `H`, `F`, `B0`, `R`, `h`, `DAMaxIter`, `DAConvValue`, `RicMaxIter`, and `RicConvValue`. The first three and the fifth inputs are the same as those required by `DSGEImpulseResponseFcn`, while the input `R` is simply the covariance matrix of the measurement errors. The last four inputs concern the maximum number of iterations (`DAMaxIter`, `RicMaxIter`) and the tolerance level (`DAConvValue`, `RicConvValue`) when calling the functions `DoublingAlgorithmLyapunov` and `RiccatiSolver`, respectively. The values for these inputs can be determined by the user on the *Settings* tab for the doubling algorithm and on the *Miscellaneous* tab for the Riccati solver.

As output the functions give `FEVDs`, `LRVD`, `status`, `RiccatiResults`, and `UniquenessCheck`. Provided that all potential calls to `RiccatiSolver` gave valid results, the matrix `FEVDs` is $n \times (n+q+1) \times h$, while `LRVD` is $n \times (n+q)$. The first 3D matrix has the $n \times (n+q+1)$ variance decomposition matrix for horizon i in `FEVDs(:,:,i)`. The first n columns contain the shares due to the n potential measurement errors, while the following q columns have the shares due to the economic shocks. The last column contains the shares due to the signal extraction error. The matrix `LRVD` is structured in the same way, except that there are no signal extraction errors. Note that if one single call to `RiccatiSolver` gives an invalid result, then both outputs from `DSGEVarianceDecompFcn` are empty.

The `status` variable is obtained from the same named output variable of the `RiccatiSolver` function, while `RiccatiResults` is a 1×2 vector with information about from the Riccati solver for the overall forecast error variance calculation. The first value gives the number of iterations used by the algorithm, while the second gives the value of the convergence measure. Finally, the `UniquenessCheck` scalar records the largest eigenvalue of $(F - KH')$, where K is the asymptote of the Kalman gain matrix for the overall forecast error variance calculation.

5.12.13. `RiccatiSolver`

The function `RiccatiSolver` tries to solve the Riccati equation (5.8) iteratively and through the use of the Matlab function `dare` from the *Control System Toolbox*. It requires 7 inputs. They are `F`, `H`, `Q`, `R`, `P1`, `MaxIter`, and `ConvValue`. As before `F` is the state transition matrix, `H` the mapping from the state variables to the observed, `Q` is the covariance matrix of the state shocks, and `R` the covariance matrix of the measurement errors. The matrix `P1` is the initial value for the covariance matrix P_1. YADA always sets this value equal to the covariance matrix of the state variables, given the assumptions made about the shocks. Finally, `MaxIter` is the maximum number of iterations that can be used when attempting to solve the Riccati equation, while `ConvValue` is the tolerance level. As in the case of the function `DoublingAlgorithmLyapunov`, the convergence criterion is given by the Matlab `norm` function applied to the change in P_1, unless the call to `dare` indicates a solution of the Riccati equation. The details of the algorithm are given below.

The Riccati solver function gives three required output variables `P1`, `status`, and `NumIter`, as well as one optional variable, `TestValue`. The first required variable is the solution candidate for P_1. The solution is considered as valid by YADA if the iterations have converged within the

maximum number of iterations. If so, the variable `status` is assigned the value 0, otherwise it is 1 unless infinite values or NaN's were located. This last case results in a value of 2 for `status`. The third output `NumIter` is simply the number of iterations used, while the fourth (optional) variable gives the value of the convergence criterion for the last iteration.

6. Required Input for YADA

In order to estimate a DSGE model YADA needs input from the user on the observed data, the measurement equation, the prior distribution, parameters that are defined from other parameters (such as the steady state), and the DSGE model. This section discusses all these topics through an example from the literature. Specifically, a log-linearized DSGE model from An and Schorfheide (2007) is used to show how the inputs should be specified.

Let $\hat{x}_t = \ln(x_t/x)$ denote the natural logarithm of some variable x_t relative to its steady state value x. The log-linearized version of the An and Schorfheide (2007) model we shall consider has 6 equations describing the behavior of (detrended) output (y_t), (detrended) consumption (c_t), (detrended) government spending (g_t), (detrended) technology (z_t), inflation π_t, and a short term nominal interest rate R_t. The equations are given by

$$\begin{aligned}
\hat{y}_t =& E_t\hat{y}_{t+1} + \hat{g}_t - E_t\hat{g}_{t+1} - \frac{1}{\tau}\left(\hat{R}_t - E_t\hat{\pi}_{t+1} - E_t\hat{z}_{t+1}\right),\\
\hat{\pi}_t =& \beta E_t\hat{\pi}_{t+1} + \kappa\left(\hat{y}_t - \hat{g}_t\right),\\
\hat{c}_t =& \hat{y}_t - \hat{g}_t,\\
\hat{R}_t =& \rho_R\hat{R}_{t-1} + (1-\rho_R)\psi_1\hat{\pi}_t + (1-\rho_R)\psi_2\left(\hat{y}_t - \hat{g}_t\right) + \sigma_R\eta_{R,t},\\
\hat{g}_t =& \rho_G\hat{g}_{t-1} + \sigma_G\eta_{G,t},\\
\hat{z}_t =& \rho_Z\hat{z}_{t-1} + \sigma_Z\eta_{Z,t}.
\end{aligned} \tag{6.1}$$

The shocks $\eta_{i,t} \sim \text{iid}N(0,1)$ for $i = R, G, Z$. The steady state for this model is given by $r = \gamma/\beta$, $R = r\pi^*$, $\pi = \pi^*$, $y = g(1-\nu)^{1/\tau}$, and $c = (1-\nu)^{1/\tau}$.

The measurement equation linking the data on quarter-to-quarter per capita GDP growth (ΔY_t), annualized quarter-to-quarter inflation rates (Π_t), and annualized nominal interest rates (I_t) to the model variables are given by:

$$\begin{aligned}
\Delta Y_t =& \gamma^{(Q)} + 100\left(\hat{y}_t - \hat{y}_{t-1} + \hat{z}_t\right),\\
\Pi_t =& \pi^{(A)} + 400\hat{\pi}_t,\\
I_t =& \pi^{(A)} + r^{(A)} + 4\gamma^{(Q)} + 400\hat{R}_t.
\end{aligned} \tag{6.2}$$

Additional parameter definitions are:

$$\beta = \frac{1}{1 + \dfrac{r^{(A)}}{400}}, \quad \gamma = 1 + \frac{\gamma^{(Q)}}{100}, \quad \pi = 1 + \frac{\pi^{(A)}}{400}, \tag{6.3}$$

where only the β parameter is of real interest since it appears in (6.1), while γ and π are related to the steady state only. The parameters to estimate for this model are therefore given by

$$\theta = [\tau\ \kappa\ \psi_1\ \psi_2\ \rho_R\ \rho_G\ \rho_Z\ r^{(A)}\ \pi^{(A)}\ \gamma^{(Q)}\ \sigma_R\ \sigma_G\ \sigma_Z]'. \tag{6.4}$$

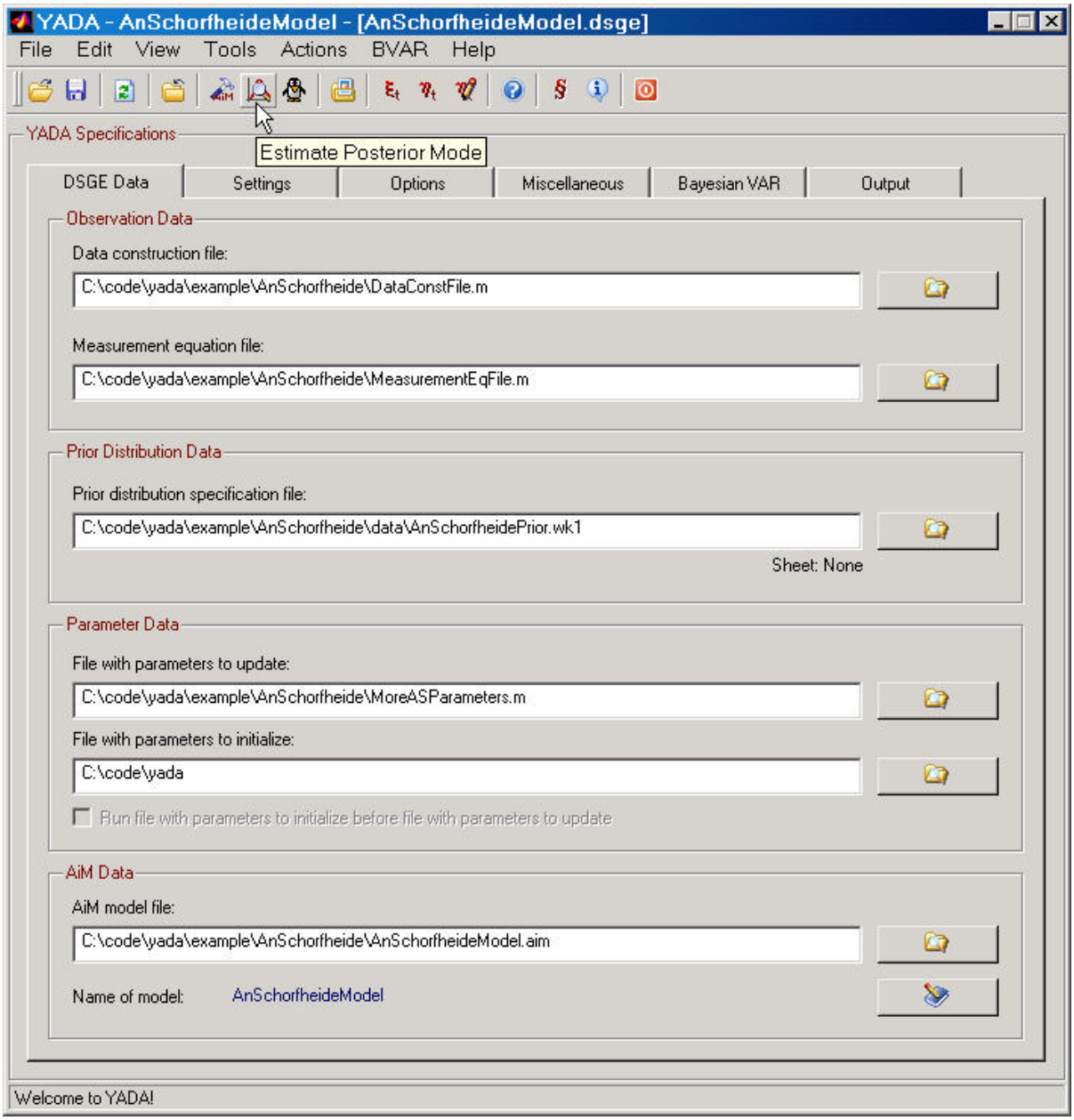
The *DSGE Data* tab in YADA is shown in Figure 1 indicating the various input files that are used to estimate the parameters of the An and Schorfheide model.

6.1. *Reading Observed Data into YADA*

To estimate θ with YADA the data on $y_t = [\Delta Y_t\ \Pi_t\ I_t]'$ and $x_t = 1$ needs to be read from a file. Since the user may wish to transform the raw data prior to defining y_t by, e.g., taking logs, YADA requires that the construction of data is handled in a Matlab m-file.

As an example, consider the data construction file `DataConstFile.m` that is located in the sub-directory `example\AnSchorfheide`. It assumes that there are no inputs for the function. The requirements on its setup concerns the structuring of the output. Specifically, the data

FIGURE 1. The DSGE Data tab on the YADA window.



construction file should return a structure, named e.g., `StructureForData`. The actual name of the structure is not important as it is a local variable to the function. The fields of this structure, however, have required names and setup. The matrix with observed data should appear as `StructureForData.Y.data`. It should preferably be of dimension $n \times T$ with $n < T$; if not, YADA will take its transpose.

If you need to transform your data prior to using them for estimation, you can always do so in your data construction file. For instance, you may want to take natural logarithms of some of the variables in your data input file, you may wish to rescale some variables, take first differences, remove a linear trend, etc. All matlab functions located on the `matlabpath` can be used for this purpose. It is important to note, however, that any files you instruct YADA to read data from should be specified in the data construction file with their full path. The reason is that YADA copies all the matlab m-files that you specify on the DSGE Data tab (see Figure 1) to the directory `tmp` and executes them from there. That way, YADA avoids having to deal with

temporary changes to the path. At the same time, a data file located in, e.g., the same directory as your data construction file will not be copied to the `tmp` directory. Hence, a command like

```
wk1read(['data\AnSchorfheideData.wk1']),
```

will not work unless you manually create a directory `data` below YADA's working directory and copy the file `AnSchorfheideData.wk1` to this directory.

The working directory for YADA is always given by `pwd`, i.e., the directory where `YADA.m` is located. Hence, if you store your data file in a sub-directory to YADA's working directory, e.g., `example\AnSchorfheide\data` you can use the command `pwd` to set the root for the path where your data file is located. In this case,

```
wk1read([pwd '\example\AnSchorfheide\data\AnSchorfheideData.wk1']).
```

When you exit YADA, all files found in the `tmp` directory are automatically deleted.

The names of the observed variables should appear in the field `StructureForData.Y.names`. It should be given as a cell array with n string elements. In `DataConstFile.m`, $n = 3$ while

```
StructureForData.Y.names = {'YGR' 'INFL' 'INT'}.
```

The data for any exogenous variables should be given by `StructureForData.X.data`, a matrix of dimension $k \times T$, e.g., a vector with ones for a constant term. Similarly, the cell array `StructureForData.X.names` provides the names of these variables, e.g., being given by `{'const'}`. If the model has no exogenous variables, then these two fields should be empty.

Given that the model has exogenous variables, it is possible to add extra data on these variables to the entry `StructureForData.X.extradata`. This is an optional entry that if used should either be an empty matrix or a $k \times T_h$ matrix. YADA views this data on the exogenous variables as having been observed after the data in `StructureForData.X.data`. Hence, the extra data can be used in, for instance, out-of-sample forecasting exercises where it will be regarded as observations $T + 1$ until $T + T_h$.

Next, the field `StructureForData.sample` should contain a 4 dimensional vector with entries giving the start year, start period, end year and end period. For instance,

```
StructureForData.sample = [1980 1 2004 4].
```

This sample data refers to the data in the matrix `StructureForData.Y.data` for the observed variables and the matrix `StructureForData.X.data` for the exogenous variables. The sample used for estimation can be changed on the Settings tab in YADA.

YADA requires that the data frequency is specified as a string. Valid string entries are `quarterly`, `monthly`, and `annual`. The first letter of these strings are also permitted. The name of the field for the data frequency is simply `StructureForData.frequency`.

6.1.1. *Transformations of the Data*

It is possible to transform the data on the observed variables (`StructureForData.Y.data`) through YADA. Such transformations, however, will not be applied to the data for estimation purposes. Rather, in certain situations, such as forecasting, it may be interesting to transform the data based on user defined functions. The information YADA needs for such data transformations is assumed to be provided via the data construction file, through the sub-field `StructureForData.Y.transformation`. This field is not mandatory, but if it is missing then YADA will not be able to transform the observed variables.

The transformations are performed on a variable-by-variable basis. For this reason YADA assumes that the name of an observed variable provides a sub-field in the structure. For instance, in the `DataConstFile.m` there is a field `StructureForData.Y.transformation.YGR` for GDP growth. For each variable specific sub-field there are 6 sub-sub-fields that are needed. These are: `fcn`, `partial`, `annualizefcn`, `annualizepartial`, `initial`, and `x`. In addition, there are 3 sub-sub-fields for inverting the transformation. These are: `invertfcn`, `invertinitial`, and `invertx`. Finally, YADA also has 5 sub-sub-fields for dealing with exporting of data: `exportfcn`, `exportinitial`, `exportx`, `exporttitle`, and `exportname`.

The fields `fcn` and `annualizefcn` hold string vectors for transforming the data. The `inline` function is used by YADA and for the string vector to be operational it is therefore required

that it holds valid Matlab syntax. The field `fcn` holds the general transformation function, while `annualizefcn`, as the name suggests, holds a particular function that is used when the transformation is assumed to provide an annualization of the data.

The fields `partial` and `annualizepartial` hold the first order partial derivatives of the functions in the `fcn` and `annualizefcn` string vectors with respect to the variable that should be transformed *times* a part of the variable in question. That is, this string vector should include the full first order Taylor expansion term for the variable to be transformed ($(\partial f(x)/\partial x)x_j$, where $x = x_1 + \ldots + x_n$ and $j = 1, \ldots, n$). Since the partial and the variable are both vectors, the element-by-element product operator (`.*`) should be used. If the partial is a constant, then the partial should take into account that x is the variable to be transformed and x_j is a part of this variable. The `partial` string vector must have exactly as many variables as the `fcn` string vector *plus* one additional variable, which must be the last variable in the string vector. Apart from the last term all other variables in `fcn` and `partial` must appear in the same order. This means that if the `fcn` string is `'4*(S-MeanS)'`, then `partial` should be, e.g., `'(4-(0*(S-MeanS))).*PartS'`. The same rule holds for the `annualizepartial` string vector.

The field `initial` holds a vector of initial values for the variable that is to be transformed. These values will be prepended to the vector of data on the variable prior to it being transformed. The dating of the initial values is assumed to be the periods just before to the start of the full sample for the observed variables.

The field `x` holds a matrix with data on any additional variables required by the transformation. The dimension of this matrix should be $d_x \times T_x$, where d_x is the number of additional variables. Since it is possible that certain transformation functions require different dimensions of the various additional variables, YADA will look for `NaN` entries at the beginning of each row of `x` and remove such entries on a variable-by-variable basis before executing the transformation.

To illustrate these features, suppose that the transformation function for a variable `YGR` is:

```
cumsum(YGR)+0.2*TREND
```

and that one initial value is given for `YGR`. Data for the variable `TREND` is stored in `x`. Since one observation is prepended to the vector of data for `YGR`, the dimension of the data for `TREND` must match the dimension for `YGR`, i.e., $T_x = T + 1$. This means that `x` is a $1 \times (T + 1)$ vector with data on `TREND`. For instance, this may be the vector $[0\ 1\ \cdots\ T]$.

Suppose we instead consider the following transformation function:

```
YGR-diff(N)+0.2*TREND
```

and that the variable `N` is located in the first row of `x` and `TREND` in the second row. In this case, `N` needs to have one more element than `YGR` once initial values for latter have been taken into account. At the same time `TREND` should have the same number of elements as `YGR`. By letting the first element in the second row of `x` be `NaN` (while the first element of the first row is a real number), this transformation can be achieved.

Since the `inline` function in Matlab, when executed with one input argument, creates a function with an input ordering of the variables that follows the order in which the variables appear in the string vector provided to `inline`, YADA requires that the variable to be transformed appears first in the function string, and all the additional variables thereafter. The ordering of the additional variables is assumed to match the ordering of these variables in the matrix `x`.

Assuming that the transformation has been performed successfully, YADA checks if the dimension of the transformed variable is greater than that of the original variable. Should this be the case, data at the beginning of the variable created by the transformation are removed such that the dimensions match. In the event that the dimension of the variable created is smaller than the original, then YADA assumes that the transformation used up data at the beginning from a time perspective. This would, for example, be the case if the transformation uses the `diff` function and no initial values are made available.

The field `invertfcn` holds a string that describes how the function in the field `fcn` should be inverted. In analogy with the `fcn` field, the fields `invertinitial` and `invertx` hold initial values and data on all additional variables that are needed by the inversion function. Similarly,

the function for exporting data is stored in the field `exportfcn`. Likewise, the information about initial values and additional variables required for the export transformation are located in the fields `exportinitial` and `exportx`. Finally, the fields `exporttitle` and `exportname` hold string vectors that makes it possible to use a different name of the variable in the file with exported data. The `exporttitle` string is written to the line above the `exportname` string.

The following example for GDP growth is found in the file `DataConstFile.m`:

```
StructureForData.Y.transformation.YGR.fcn = '100*(exp(YGR/100)-1)';
StructureForData.Y.transformation.YGR.partial = 'exp(YGR/100).*PartYGR';
StructureForData.Y.transformation.YGR.annualizefcn = ...
   '100*(exp((1/100)*(YGR(4:length(YGR))+YGR(3:length(YGR)-1)+...
   YGR(2:length(YGR)-2)+YGR(1:length(YGR)-3)))-1)';
StructureForData.Y.transformation.YGR.annualizepartial = ...
   'exp((1/100)*(YGR(4:length(YGR))+YGR(3:length(YGR)-1)+...
   YGR(2:length(YGR)-2)+YGR(1:length(YGR)-3))).*...
   (PartYGR(4:length(PartYGR))+PartYGR(3:length(PartYGR)-1)+...
   PartYGR(2:length(PartYGR)-2)+PartYGR(1:length(PartYGR)-3))';
StructureForData.Y.transformation.YGR.initial = [];
StructureForData.Y.transformation.YGR.x = [];
StructureForData.Y.transformation.YGR.invertfcn = ...
   '100*log(1+(YGR/100))';
StructureForData.Y.transformation.YGR.invertinitial = [];
StructureForData.Y.transformation.YGR.invertx = [];
StructureForData.Y.transformation.YGR.exportfcn = ...
   '100*(exp(cumsum(YGR)/100))';
StructureForData.Y.transformation.YGR.exportinitial = 0;
StructureForData.Y.transformation.YGR.exportx = [];
StructureForData.Y.transformation.YGR.exporttitle = 'Real GDP';
StructureForData.Y.transformation.YGR.exportname = 'DY';
```

Since `YGR` is the log first difference of GDP, the general function in the `fcn` field calculates the (quarterly) growth rate of GDP. The annualization function similarly provides the annual growth rate of GDP, while no initial data are supplied and the computations do not need any additional variables. The field `partial` (`annualizepartial`) is the first order partial derivative of the `fcn` (`annualizefcn`) function times a part of the variable that is transformed. The function is used when the transformation of the variable is applied to a linear decomposition of the variable, such as the observed variable decomposition in Section 5.7.

The field `invertfcn` inverts the calculation in `fcn`, while the field `exportfcn` gives the expression for calculating the levels data for `YGR` based on a constant initial value for the variable. The variables `INFL` and `INT` have their own transformation functions; see `DataConstFile.m` for details.

6.1.2. *Levels or First Differences*

To inform YADA about which variables appear in levels (like the interest rate) and which appear in first differences (output and inflation), the field `levels` should contain a vector whose elements are either 0 (first difference) or 1 (level). In our example this means that

```
StructureForData.Y.levels = [0 0 1].
```

This information makes it possible for YADA to compute, e.g., the levels responses in all observed variables of a certain economic shock. If the levels field is missing from the data construction file, then YADA displays a message box to remind you. The file is, however, regarded as valid, with all observed variables in levels. Once you add the field to the data construction file, YADA will stop nagging you about the missing information.

6.1.3. *Simple Annualization*

Furthermore, to tell YADA how to "annualize" observed variables, the field `annual` should contain a vector whose elements are either 0 (do not annualize/is already annualized) or 1 (annualize). YADA annualizes a variable by adding to the current value the previous 3 (11) observations for quarterly (monthly) data. For instance, the `YGR` variable measures quarterly logged GDP (per capita) growth. Summing this variable over 4 consecutive quarters thus gives the annual logged GDP (per capita) growth. The inflation and interest rate variable are already assumed to be measured in annual terms. This means that we set:

```
StructureForData.Y.annual = [1 0 0].
```

How the data frequency is specified is provided below. It may be noted that the `annual` field is optional, with the default being a zero vector, and that YADA does not display a nag screen when this field is missing.

In some situations you may wish that the annualized data should be multiplied by a constant. For example, the inflation equation in (6.2) "annualizes" quarter-to-quarter inflation by multiplying the quarterly price changes with 4. The annualized inflation series can therefore be calculated by adding four consecutive quarters of Π_t and dividing the sum with 4. For YADA to compute such an annualized inflation series field `annualscale` should be specified. Here, we may set

```
StructureForData.Y.annualscale = [1 0.25 1].
```

Notice that YADA will only use elements of this vector on the variables that you allow YADA to annualize. This means that the second element of the vector `StructureForData.Y.annual` must be unity before YADA will apply the scaling constant 0.25 to inflation. Moreover, the scaling vector is optional and is by default equal to a unit vector.

6.1.4. *Bayesian VAR*

If you wish to estimate a Bayesian VAR model with YADA you may include one additional field in the `StructureForData.Y` structure. This field, denoted by `BVAR`, should provide the variable positions in the matrix `StructureForData.Y.data`. These variable numbers may be ordered in any way you wish and all observed variables need not appear. In the An and Schorfheide example we let:

```
StructureForData.Y.BVAR = [1 2 3].
```

If the field `BVAR` is missing, then YADA automatically assumes that all observed variables are to be included in any Bayesian VAR analysis you wish to perform. Hence, the field `BVAR` is also optional and YADA does not display any nag screen to inform you about this. Please note that Section 12 discussed in some detail the Bayesian VAR models that you can estimate with YADA.

For Bayesian VAR analysis you may also wish to specify which exogenous variables should be included. You do this in an analogous way to how observed variables are made available as endogenous variables in the VAR model. That is, you let `StructureForData.X.BVAR` be equal to a vector with integer values corresponding to the variable position in the matrix with exogenous variables, i.e., `StructureForData.X.data`. Also, you need not include all exogenous variables that the estimated DSGE model has. This feature is optional and by default YADA will include all exogenous variables in the Bayesian VAR model when you decide to estimate such a model.

6.1.5. *Conditional Forecasting Data*

If you wish to perform conditional forecasting you need to set up the entries needed for conditioning. All fields discussed below for conditional forecasting are optional, but all are required if you wish to perform such projections. The mathematics behind conditional forecasting is explained in Section 13.2. Here I shall focus on definitions that can be made in the data construction file.

First, the variables that you wish to condition on are located in the field `Z` in the structure `StructureForData`. As in the case of the observed variables and the exogenous variables, the subfields `data` and `names` holds the data and the names of the conditioning information. It is

important to note that the data subfield should contain an $m \times T_z$ matrix with T_z being the number of available observations of the conditioning information. It is assumed that the first row in this matrix is measured at the same point in time as the first row in the matrix with the observed variables. Since the latter has T observation, the value of T_z should be greater than T if you plan to use all the observed data for estimation and then perform conditional forecasting for periods $T + 1$ and onwards until T_z. The subfield names should hold a cell array with length equal to m, the number of variables to condition on. Note that you may select a subset of these variables through the YADA GUI.

Second, the field Z has two subfields that contain matrices K1 and K2 that are needed to map the observed variables into the conditioning information; cf. equation (13.7). The first matrix is required to be of full rank m, while the second can be a zero matrix. Their common dimension should be $n \times m$, where n as before is the number of observed variables and m is the number of conditioning variables in StructureForData.Z.data.

Since the mapping from the observed variables to the conditioning variables may require an initial value, YADA also needs data in the field U in the structure StructureForData. This field should contain the subfield data that provides an $m \times T_u$ matrix with initial conditions in the mapping; cf. (13.7) for details. Again, YADA assumes that the first observation in this matrix comes from the same time period as the first observation in the matrix with observed variables. This initial conditions data can be used provided that T_u is at least equal to the last period used for estimation of the parameters in the DSGE model. Finally, the conditioning data can also be linked with transformation function and the logic is the same as in the case of the transformation functions for the observed variable scenario data, i.e., through the field StructureForData.Z.transformation.

6.1.6. *Percentiles for Distributions*

It is possible to control the percentiles that are used for plotting confidence bands for certain distributions. This is currently handled by StructureForData.percentiles. This optional entry should be a vector with integer values greater than 0 and less than 100. If there are at least 2 valid entries YADA will make use of them. For instance, we may write

```
StructureForData.percentiles = [85 50 15 5 95].
```

Unless the values are already expressed in ascending order YADA will sort them. Moreover, if the vector has an odd number of entries then the middle entry is ignored. In the above example we thus have that YADA would treat the 5 dimensional vector as equal to

```
StructureForData.percentiles = [5 15 85 95].
```

YADA then assumes that the first and the last element may be used to construct the outer confidence band, while the second and the third are used for an inner confidence band.

6.2. *Setting up the Measurement Equation*

The measurement equation is specified as a function in a Matlab m-file. This file takes 7 input arguments. The first is the structure ModelParameters, whose fields have names given by the parameter names. Hence, if the model has a parameter called rA, then the ModelParameters structure has a field ModelParameters.rA. If rA is specified in the prior distribution file, then this field is automatically generated by YADA. If you need to refer to this parameter in the measurement equation file, then the syntax should follow this example. Note, that ModelParameters can take any name you wish in the measurement equation file since the name of the structure is local to the function.

A second input argument for this function is the string matrix called, e.g., StateVarNames. This matrix contains a name in each row that is equal to the name obtained from parsing the AiM model file. That is, it is equal to the endog_ output from the compute_aim_data function that the AiM parser creates.

The third and fourth input variables for the measurement equation function are the string matrices VariableNames and XVariableNames. The rows of these matrices provide the names of the observed and the exogenous variables, respectively.

The last three input arguments for the measurement equation function are `n`, `r`, and `k`. These are equal to the dimension variables n, r, and k in (3.1).

The output from this function should be the three matrices A, H, and R; see, equation (3.1). The dimensions of these matrices should be exactly as they are specified in Section 3. That is, A is $k \times n$ if $k > 0$ and an empty matrix otherwise, H is typically an $r \times n$ matrix, while R is an $n \times n$ matrix with measurement error covariances. If there are no measurement errors in your model then $R = 0$.

The measurement equation from the An and Schorfheide model (cf. equation (6.2)), is specified in the file `MeasurementEqFile` in the sub-directory `example\AnSchorfheide`. This file shows how these matrices can be determined from the 7 input arguments to the function as well as using the `loc` function for locating the position of variables within the `StateVarNames`, `VariableNames`, and `XVariableNames` string matrices. It may be noted that of the 7 input arguments only the first is always likely to be useful since it includes all the parameter values. The other 6 inputs are provided primarily for convenience. When you write your own measurement equation m-file, you have to make sure that all the 7 inputs are accepted. Whether you then make use of them or not in the function is, of course, up to you.

It is possible to let the measurement matrix H be time-varying. YADA handles this case by letting the H matrix exported from the measurement equation file be 3-dimensional, i.e., an $r \times n \times T_H$ matrix, where T_H is large enough for the estimation sample to be covered. YADA assumes that `H(:,:,t)` measures the same time period t as column t in the $n \times T$ matrix `StructureForData.Y.data`; cf. Section 6.1.

It is important to note that if there are many calls to, say, the `loc` function for setting values to the correct entries of the A, H, and R matrices, this may slow down both the posterior mode estimation phase and when draws are taken from the posterior distribution via the random walk Metropolis algorithm. For smaller models, the additional time occupied by these calls may be negligible, but in larger models this may affect the computation time considerably. Hence, I would suggest that the `loc` function is used only for testing purposes in such cases, and that the entry numbers are hand-coded into the measurement equation file later on.

The measurement equation file is always checked for internal consistency before commencing with, e.g., posterior mode estimation.

6.3. *Specification of the Prior Distribution*

The file with the prior distribution data must be given by either a Lotus 1-2-3 spreadsheet (file extension `.wk1`) or an Excel spreadsheet (extension `.xls`). The An and Schorfheide example comes with a number of such prior distribution files, e.g., `AnSchorfheidePrior.wk1`.

The prior distribution file should list all the parameters that are going to be estimated. It may also list parameters that are calibrated.[14] The 7 required column headers in this file are given by `model parameter`, `status`, `initial value`, `prior type`, `prior parameter 1`, `prior parameter 2`, and `lower bound`. The entries under the `lower bound` header are in fact ignored unless the prior distribution is gamma, inverse gamma, or left truncated normal. Furthermore, all the *headers* are case insensitive in YADA.

YADA also supports an optional header `Upper bound`. This header is used for the beta distribution only. When YADA locates this header it will also take the lower bound for beta distributed parameters into account. If the header is missing YADA assumes that any beta distributed parameters have lower bound 0 and upper bound 1.

6.3.1. *The Model Parameter Header*

The names of all the parameters that need to be estimated should be specified under this header. It is important that the names are exactly the same as in other input files, e.g., the AiM model file (see, Section 6.5). Since Matlab is case sensitive regarding variable and field names, the parameter names are case sensitive.

[14] In Section 6.4, we discuss alternative and more flexible ways of specifying additional parameters that YADA needs to know about in order to solve the model using AiM.

TABLE 1. An example of the required and optional data for the prior distribution file.

Model parameter	Status	Initial value	Prior type	Prior parameter 1	Prior parameter 2	Lower bound	Upper bound
tau	estimated	1.87500	gamma	2.000	0.50	1.000	
kappa	estimated	0.15000	gamma	0.200	0.10	0.000	
psi1	estimated	1.45830	gamma	1.500	0.25	0.000	
psi2	estimated	0.37500	gamma	0.500	0.25	0.000	
rhoR	estimated	0.50000	beta	0.500	0.20	0	1
rhoG	estimated	0.84620	beta	0.800	0.10	0	1
rhoZ	estimated	0.70590	beta	0.660	0.15	0	1
rA	estimated	0.50000	gamma	0.500	0.50	0.000	
piA	estimated	6.42860	gamma	7.000	2.00	0.000	
gammaQ	estimated	0.40000	normal	0.400	0.20		
sigmaR	estimated	0.00358	invgamma	0.004	4.00	0.000	
sigmaG	estimated	0.00859	invgamma	0.010	4.00	0.000	
sigmaZ	estimated	0.00447	invgamma	0.005	4.00	0.000	

6.3.2. *The Status Header*

The status header reports if a parameter should be estimated or is to be viewed as calibrated. The valid entries are thus `es(timated)` and `calibrated`. In fact, as long as the status string is not empty, the parameter will be regarded as calibrated unless the first two letters of the status string are `es`. This entry is case insensitive.

6.3.3. *The Initial Value Header*

The initial value must be a scalar and it should be in the support of the prior distribution assigned to the parameter. Natural candidates as an initial value is either the mean or the mode of the prior distribution (if they exist).

6.3.4. *The Prior Type Header*

The prior type header determines the prior distribution of the parameter. The entry should be one of the following: gamma, beta, invgamma, normal, truncnormal, or uniform. This entry is case insensitive.

6.3.5. *The Prior Parameter 1 Header*

The prior parameter 1 header reports the first parameter for the prior distribution. This parameter is assumed to be the mean of the gamma, beta and normal distributions, the s parameter for the inverse gamma distribution (see equation (2.9)), the μ parameter for the left truncated normal distribution (see equation (2.16)), and the lower bound of the uniform distribution.

6.3.6. *The Prior Parameter 2 Header*

The prior parameter 2 header reports the second parameters for the prior distribution. This parameter is assumed to be the standard deviation of the gamma, beta, and normal distributions. For the inverse gamma distribution it is the q (degrees of freedom) parameter, for the left truncated normal the σ parameter, and the upper bound for the uniform distribution.

6.3.7. *The Lower Bound Header*

The lower bound header is primarily used if the distribution is gamma, inverse gamma, or left truncated normal. For the left truncated normal the lower bound is, as in Section 2.2.6, the c parameter. The entries can for the other prior distributions be either empty or real numbers.

6.3.8. *The Upper Bound Header*

The upper bound header is optional and, when present, is only used by the beta distribution. If the upper bound header is missing or the lower bound is not specified, then the beta prior is assumed to have lower bound 0 and upper bound 1. When the header is present it will be ignored for parameters that do not have a beta prior.

6.4. *Defining Additional Parameters*

Additional parameters can optionally be included as input for YADA. These can be parameters that are calibrated but not specified in the prior distribution file. They can also be parameters, such as β in (6.1), that are defined from other parameters, e.g., steady state parameters. YADA allows for input of two different types of additional parameters. The first is a function that only specifies parameters that should be initialized, while the second is a function with parameters that should be updated along with the estimated parameters. The β example concerns the latter type of additional parameters function.

For both types of additional parameters functions, YADA requires that the function takes as input the structure with model parameters, e.g., `ModelParameters` whose fields have the same names as the names specified in the prior distribution file and the AiM model file. As output, the function must also give the structure with model parameters. In the β example, the AiM code uses a parameter `beta`. Since this is a parameter updated when `rA` receives a new value, the function with parameters to update should include a line such as:

```
ModelParameters.beta = 1/(1+(ModelParameters.rA/400)).
```

See the file `MoreAsParameters.m` in the sub-directory `example\AnSchorfheide` for more details.

There are three names that may not be used for the parameters. They are: `YADAg`, `YADAh`, and `UserVariables`. The first two are used internally to allow for parameters with the names `g` and `h`, respectively. The last is a reserved name that allows the user to pass on information from the parameter function. That information can be reused by the function itself, such as holding initial values for some numerical problem that the function solves (e.g., steady-state calculations). The field `ModelParameters.UserVariables` is viewed as a structure where the user can select his or her own field names. YADA, for its part, simply ignores the `UserVariables` field when dealing with parameters. YADA stores `ModelParameters`, along with a number of other variables, in a mat-file when it has finished the posterior mode estimation routine. The user can therefore access data in `ModelParameters.UserVariables` via that file. To find it, simply look in the `mode` directory that the posterior mode estimation routine creates.

Any additional parameters files that you have specified are always checked for internal consistency before executing, e.g., the actual posterior mode estimation functions.

In Section 2.2 a number of prior distributions were discussed where the function with parameters that need to be updated can be utilized to support additional priors. For example, suppose that we have a parameter α whose prior should be a Weibull distribution with scale parameter $a = 3$ and shape parameter $b = 2$; cf. Section 2.2.2. We may then define the auxiliary parameter α_G with prior distribution $G(1, 1)$. The code

```
ModelParameters.alpha = 3*ModelParameters.alphaG^(1/2)
```

in the file with parameters to update ensures that the parameter α has a $W(3, 2)$ prior distribution.

6.5. *Construction of the AiM Model File*

The AiM model file is simply a text file that is written using a syntax that the AiM parser can interpret. The code used for the An and Schorfheide model is listed in Table 2. In this case, the model has 6 state variables (cf. equation (6.1)), but an additional state variable has been included to account for the need of $\hat{y}_{t-1}$ in the measurement equation for ΔY_t in (6.2). Hence, $p = r = 7$. The model also has $q = 3$ shocks (the $\eta_{i,t}$ variables) which are listed below among the variables, and one constant. The total number of variables (and equation) is therefore `NumEq` $= 11$. The names of the variables given in Table 2 will also appear in the string

matrix that will be sent as input to the measurement equation function, i.e., the string matrix `StateVarNames`.

TABLE 2. The AiM model file code for the An and Schorfheide example in equation (6.1).

```
MODEL> ASmodel
ENDOG>
       yhat       _NOTD
       pihat      _NOTD
       chat       _NOTD
       rhat       _NOTD
       ghat       _NOTD
       zhat       _NOTD
       yhatlag    _NOTD
       one        _DTRM
       etaR       _NOTD
       etaG       _NOTD
       etaZ       _NOTD
EQUATION> EQ1
EQTYPE>   IMPOSED
EQ>       yhat =      LEAD(yhat,1) + ghat - LEAD(ghat,1) - (1/tau)*rhat
                      + (1/tau)*LEAD(pihat,1) + (1/tau)*LEAD(zhat,1)
EQUATION> EQ2
EQTYPE>   IMPOSED
EQ>       pihat =     beta*LEAD(pihat,1) + kappa*yhat
                      - kappa*ghat
EQUATION> EQ3
EQTYPE>   IMPOSED
EQ>       chat =      yhat - ghat
EQUATION> EQ4
EQTYPE>   IMPOSED
EQ>       rhat =      rhoR*LAG(rhat,1) + (1-rhoR)*psi1*pihat + (1-rhoR)*psi2*yhat
                      - (1-rhoR)*psi2*ghat + sigmaR*etaR
EQUATION> EQ5
EQTYPE>   IMPOSED
EQ>       ghat =      rhoG*LAG(ghat,1) + sigmaG*etaG
EQUATION> EQ6
EQTYPE>   IMPOSED
EQ>       zhat =      rhoZ*LAG(zhat,1) + sigmaZ*etaZ
EQUATION> EQ7
EQTYPE>   IMPOSED
EQ>       yhatlag =   LAG(yhat,1)
EQUATION> EQ8
EQTYPE>   IMPOSED
EQ>       one =       0*LAG(one,1)
EQUATION> EQ9
EQTYPE>   IMPOSED
EQ>       etaR =      0*one
EQUATION> EQ10
EQTYPE>   IMPOSED
EQ>       etaG =      0*one
EQUATION> EQ11
EQTYPE>   IMPOSED
EQ>       etaZ =      0*one
END
```

The AiM code in Table 2 is also found in the file `AnSchorfheideModel.aim` located in the subdirectory `example\AnSchorfheide`. The file can be parsed by AiM and, as mentioned above, this is handled by the function `AiMInitialize`.

6.6. *YADA Code*

YADA has a function that checks if the data in the prior distribution file appears to be correct. This function is called `VerifyPriorData`. Apart from checking the validity of the data in this file, it also returns the information it has obtained from the 7 required headers. Table 1 shows how the prior distributions can be setup for the An and Schorfheide model example. The

initial values have all been set to the mode for the beta, gamma, inverse gamma and normal distributions. An exception is the `rA` parameter, whose mean and standard deviation from the gamma distribution are both set to 0.5. This means that the a parameter in equation (2.6) is unity and hence that there does not exist a single mode. In this case, the mean is used as initial value.

7. Parameter Transformations

If some of the parameters in θ have a gamma, inverse gamma, or left truncated normal prior distribution, then the support for these parameters is bounded from below. Similarly, if some of the θ parameters have a (standardized) beta or uniform prior distribution, then the support is bounded from below and above. Rather than maximizing the log posterior of θ subject to these bounds on the support, it is common practise to transform the parameters of θ such that the support of the transformed parameters is unbounded.

For the p_1 parameters with a gamma, inverse gamma, or left truncated normal prior distribution, denoted by θ_1, the transformation function that is typically applied is the natural logarithm

$$\phi_{i,1} = \ln\big(\theta_{i,1} - c_{i,1}\big), \quad i = 1, \ldots, p_1, \tag{7.1}$$

where $c_{i,1}$ is the lower bound. Letting θ_2 denote the p_2 parameters of θ with a beta or uniform prior distribution, the transformation function is the logit

$$\phi_{i,2} = \ln\left(\frac{\theta_{i,2} - a_i}{b_i - \theta_{i,2}}\right), \quad i = 1, \ldots, p_2, \tag{7.2}$$

where $b_i > a_i$ gives the upper and the lower bounds. The remaining, p_0 untransformed parameters are given by $\phi_0 = \theta_0$ while $\phi = [\phi_0' \ \phi_1' \ \phi_2']'$ and $\theta = [\theta_0' \ \theta_1' \ \theta_2']'$. The overall transformation of θ into ϕ may be expressed as $\phi = f(\theta)$.[15]

We can likewise define a transformation from ϕ into θ by inverting the above relations. That is,

$$\theta_{i,1} = \exp\big(\phi_{i,1}\big) + c_{i,1}, \quad i = 1, \ldots, p_1, \tag{7.3}$$

and

$$\theta_{i,2} = \frac{a_i + b_i \exp\big(\phi_{i,2}\big)}{1 + \exp\big(\phi_{i,2}\big)}, \quad i = 1, \ldots, p_2, \tag{7.4}$$

while $\theta_0 = \phi_0$. The full transformation can be expressed as $\theta = g(\phi)$.

When the ϕ parameters are used for computing the posterior mode, it should be noted that the log-likelihood function is not numerically affected by the transformation. This is the usual invariance property of this function. Similarly, the value of the joint prior density $p(\theta) = p(g(\phi))$ is also equivalent. Hence, the value of the log posterior is also not affected by the choices ϕ or $\theta = g(\phi)$. However, the slope of the log posterior depends on whether we consider the θ or the ϕ parameters. In particular, when computing the posterior mode, $\tilde{\theta} = g(\tilde{\phi})$, we need to take the Jacobian in the transformation from ϕ back to the θ parameters into account.

For the parameters that have a gamma, inverse gamma, or left truncated normal distribution, the log of the Jacobian is simply

$$\ln\left(\frac{d\theta_{i,1}}{d\phi_{i,1}}\right) = \phi_{i,1}, \quad i = 1, \ldots, p_1. \tag{7.5}$$

For the parameters with a beta or a uniform prior, the log of the Jacobian is

$$\ln\left(\frac{d\theta_{i,2}}{d\phi_{i,2}}\right) = \ln\big(a_i - b_i\big) + \phi_{i,2} - 2\ln\big(1 + \exp(\phi_{i,2})\big), \quad i = 1, \ldots, p_2. \tag{7.6}$$

In order to properly account for the slope effect when using the transformation $\phi = f(\theta)$, the sum of all these log Jacobians need to be added to the log posterior of θ for the posterior mode

[15] There is no need to order the parameters according to their prior distribution. YADA knows from reading the prior distribution input which parameters have a beta, a gamma, etc, prior distribution.

calculation. At the same time, all the inequality restrictions that θ_1, and θ_2 must obey, can be dropped from the optimization problem for ϕ.

7.1. *YADA Code*

YADA has four functions that handle the parameter transformations discussed above. The $\phi = f(\theta)$ mapping is handled by `ThetaToPhi`, the $\theta = g(\phi)$ mapping by `PhiToTheta`, while `logJacobian` takes care of calculating the log of the Jacobian. In addition, there is a function (`PartialThetaPartialPhi`) that computes the matrix with partial derivatives of θ with respect to ϕ.

7.1.1. `ThetaToPhi`

The function `ThetaToPhi` calculates the mapping $\phi = f(\theta)$. It requires the inputs `theta`, the θ vector; `thetaIndex`, a vector with the same length as θ with unit entries for all parameters that have a gamma, and inverse gamma, or a left truncated normal prior distribution, with zero entries for all parameters with a normal prior, with 2 for the beta prior, and 3 for the uniform prior; `UniformBounds` a matrix with lower and upper bounds of any uniformly and beta distributed parameters (for all other parameters the elements are 0 and 1); and `LowerBound`, a vector of the same length as θ with the lower bound parameters $c_{i,1}$; see also Section 8.2 regarding the function `VerifyPriorData`. The output is given by `phi`.

7.1.2. `PhiToTheta`

The function `PhiToTheta` calculates the mapping $\theta = g(\phi)$. It requires the inputs `phi`, the ϕ vector; `thetaIndex`; `UniformBounds`; and `LowerBound`. The output is given by `theta`.

7.1.3. `logJacobian`

The function `logJacobian` calculates the sum of the log of the Jacobian for the mapping $\theta = g(\phi)$. It requires the inputs `phi`, the ϕ vector; `thetaIndex`; and `UniformBounds`. The output is given by `lnjac`.

7.1.4. `PartialThetaPartialPhi`

The function `PartialThetaPartialPhi` calculates the partial derivatives of θ with respect to ϕ. The required input is `phi`, `thetaIndex`, and `UniformBounds`. The output is given by the diagonal matrix `ThetaPhiPartial`. This function is used when approximating the inverse Hessian at the posterior mode of θ with the inverse Hessian at the posterior mode of ϕ.[16]

8. Computing the Posterior Mode

The estimation of the posterior mode of θ is always performed using the transformed parameters ϕ. Letting m be the dimension of ϕ, the posterior mode of ϕ can be expressed as:

$$\tilde{\phi} = \arg\max_{\phi \in \mathbb{R}^m} \left(\ln L\big(Y; g(\phi)\big) + \ln p\big(g(\phi)\big) + \ln J\big(\phi\big) \right). \tag{8.1}$$

The matrix Y represents the observed data, $L(\cdot)$ is the likelihood function, $\theta = g(\phi)$, while $J(\phi)$ is the determinant of the (diagonal) Jacobian. The posterior mode of θ is then given by $\tilde{\theta} = g(\tilde{\phi})$.

The actual optimization is performed numerically. The user can choose between Christopher Sims' `csminwel` routine, and Matlab's `fminunc`. Both these routines provide an estimate of the inverse Hessian at the mode, denoted by $\tilde{\Sigma}$.[17] This inverse Hessian is one candidate for the

[16] The reason why we compute the derivative of θ with respect to ϕ is related to the fact that it is equal to the inverse of the derivative of ϕ with respect to θ. Specifically, the inverse Hessian with respect to θ is approximated by pre- and postmultiplying the inverse Hessian with respect to ϕ by the inverse of the diagonal matrix with partial derivatives of ϕ with respect to θ. The latter is equal to the diagonal matrix with partial derivatives of θ with respect to ϕ.

[17] The Matlab function `fminunc` actually produces an estimate of the Hessian at the mode.

covariance matrix of the proposal density that the Metropolis algorithm discussed in Section 9 needs for generating candidate draws from the posterior distribution of ϕ.

Note that the YADA specific version of `fminunc` is not supplied with the public version of YADA. Moreover, the original matlab version of `fminunc` is not supported by the posterior mode estimation routine in YADA since it uses an edited version of the function (named `YADAfminuncx`, where `x` should be replaced with 5 or 7). The YADA specific version has some additional output fields and also supports a progress dialog. To make it possible for users that have matlab's *Optimization Toolbox* installed to use `fminunc` for posterior mode estimation in YADA, I plan to release diff-file type of specifications that instruct users how to edit their own copies of `fminunc` and related files.

8.1. *Checking the Optimum*

In order to check if the value $\tilde{\phi}$ is a local optimum, YADA makes use of some tools suggested and originally coded by Mattias Villani. For each element ϕ_i of the vector ϕ a suitable grid with d elements is constructed from the lower and upper bounds $(\tilde{\phi}_i - c\tilde{\Sigma}_{i,i}^{1/2}, \tilde{\phi}_i + c\tilde{\Sigma}_{i,i}^{1/2})$, where $c > 0$ and $\tilde{\Sigma}_{i,i}$ is element (i,i) of $\tilde{\Sigma}$, the inverse Hessian of the log posterior at the mode. Let ϕ_{-i} be a vector with all elements of ϕ except element i. For each ϕ_i in the grid, the log posterior is evaluated at $(\phi_i, \tilde{\phi}_{-i})$. For parameter ϕ_i this provides us with d values of the log posterior of ϕ_i conditional on $\tilde{\phi}_{-i}$.

One proposal density that YADA can use for posterior sampling is $N(\phi, \tilde{\Sigma})$, where the value of ϕ is determined from the previous draw from the posterior. Since the computed values of the log posterior of ϕ_i are conditional on all parameters being equal to their values at the posterior mode, it is natural to compare them to a conditional proposal density for ϕ_i. For the grid values of ϕ_i that were used to calculate the conditional log posterior values of the parameter, one such density is the log of the normal density with mean $\tilde{\phi}_i$ and variance $\tilde{\Omega}_{i|-i} = \tilde{\Sigma}_{i,i} - \tilde{\Sigma}_{i,-i}\tilde{\Sigma}_{-i,-i}^{-1}\tilde{\Sigma}_{-i,i}$. The vector $\tilde{\Sigma}_{i,-i}$ is equal to the i:th row of $\tilde{\Sigma}$ with element i removed. Similarly, the matrix $\tilde{\Sigma}_{-i,-i}$ is obtained by removing row and column i from $\tilde{\Sigma}$.

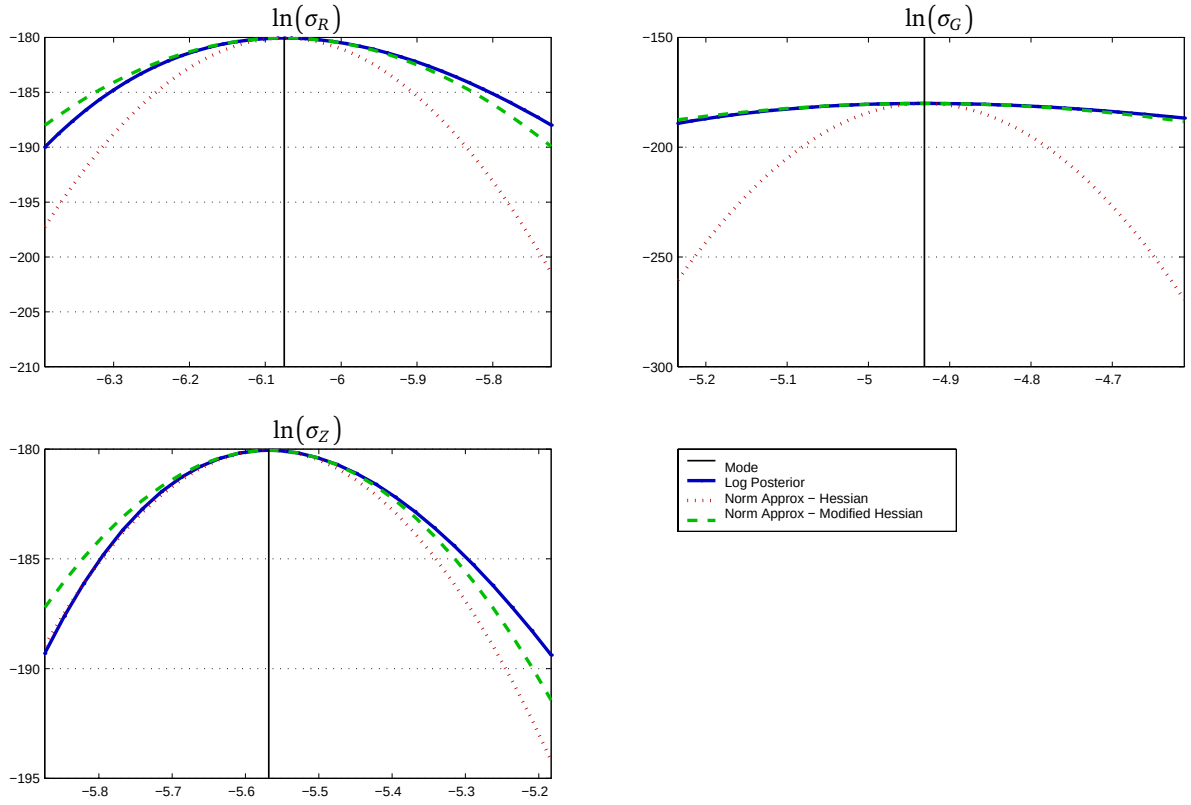
We can also estimate the variance of a conditional proposal density by running a regression of the log posterior values on a constant, ϕ_i, and ϕ_i^2. The estimated variance is now given by the inverse of the absolute value of two times the coefficient on ϕ_i^2. A modified proposal density can now be evaluated for each ϕ_i by using the log of the normal density with mean $\tilde{\phi}_i$ and variance given by the estimate in question.[18]

Using these ideas, Figure 2 provides graphs of the conditional log posterior density (blue solid line) of the parameters σ_R, σ_G, and σ_Z from the An and Schorfheide model in Section 6. The transformed (ϕ) space for the parameters is used here, i.e., the log of the parameters. Since the support for ϕ is the Euclidean space, it is seems a priori more likely that a normal distribution can serve well as a proposal density for ϕ than for θ, where, for instance, the σ parameters are restricted to be positive; cf. Section 7. The red dotted line shows the log normal approximation of the log posterior using the posterior mode as mean and conditional variance based on the inverse Hessian at the mode. The green dashed line is similarly based on the log normal approximation with the same mean, but with the conditional variance estimated as discussed in the previous paragraph.

It is worth noticing from the Figure that the normal approximation based on the modification is close to the log posterior for all these parameters, while the normal approximation based on the inverse Hessian typically lies inside the log posterior. In the case of σ_G, these differences

[18] Such an estimated conditional variance can also be transformed into a marginal variance if, e.g., we are willing to use the correlation structure from the inverse Hessian at the posterior mode. Let $C = \Sigma \ominus \varsigma\varsigma'$, where $\ominus$ denotes element-by-element division, and ς is the square root of the diagonal of Σ. Let $\Omega_{i|-i}$ be the conditional variance, while $\Sigma_{i,i}$ is the marginal variance. For a normal distribution we know that $\Omega_{i|-i} = \Sigma_{i,i} - \Sigma_{i,-i}\Sigma_{-i,-i}^{-1}\Sigma_{-i,i}$. This relationship can also be expressed through the correlation structure as $\Omega_{i|-i} = (1 - C_{i,-i}C_{-i,-i}^{-1}C_{-i,i})\Sigma_{i,i}$. Hence, if we have an estimate of the conditional variance $\Omega_{i|-i}$ and the correlation structure C, we can compute the marginal variance $\Sigma_{i,i}$ by inverting this expression.

FIGURE 2. Plot of the conditional log posterior density around the estimated posterior mode along with two conditional proposal densities for the parameters σ_R, σ_G, and σ_Z.



seem to be particularly severe and indicates that if the proposal density has $\tilde{\Sigma}$ as its covariance matrix, then it may take a long time before the support of σ_G is sufficiently covered.

8.2. *YADA Code*

The posterior mode is computed in YADA by the function `PosteriorModeEstimation`. The main inputs for this function are the structures `DSGEModel`, `CurrINI`, and `controls`. The first contains paths to the user files that specify the log-linearized DSGE model, the prior distribution of its parameters, the data, the measurement equations, and any parameter functions that should be dealt with. It also contains information about options for the Kalman filter, the sample to use, names of observed variables, of exogenous variables, of state variables, and names of the state shocks, your choice of optimization routine, the tolerance value, the maximum number of iterations to consider, as well as some other useful features.

The `CurrINI` structure contains data on initialization information needed by YADA. This structure contains non-model related information, while the `DSGEModel` structure contains the model related information. Finally, the `controls` structure holds handles to all the controls on the main GUI of YADA.

8.2.1. `VerifyPriorData`

Based on the input that `PosteriorModeEstimation` receives, the first task it performs is to check the data in the prior distribution file. This is handled by the function `VerifyPriorData`. Given that the prior distribution data is complete (cf. Section 6.3), this function returns the prior distribution data in various variables. These variables are given by `theta`, `thetaIndex`, `thetaDist`, `LowerBound`, `ModelParameters`, `thetaPositions`, `PriorDist`, `ParameterNames`, and `UniformBounds`.

The vector `theta` contains the initial values of the parameters to be estimated, i.e., θ. The vectors `thetaIndex` and `thetaDist` have the same length as `theta` with integer entries indicating the type of prior distribution that is assumed for each element of `theta`. The difference between these two vectors is that `thetaIndex` indicates the type of transformation that should be applied to obtain ϕ, while `thetaDist` gives the prior distribution. The vector `LowerBound` gives the lower bound specified for the parameters in the prior distribution file. This bound is, for example, used by the parameter transformation function discussed in Section 7.1.

The structure `ModelParameters` has fields given by the parameter names assigned in the prior distribution file; see, e.g., Table 1. Each field is assigned a value equal to the initial value for that parameter. Both estimated and calibrated parameters are given a field in the `ModelParameters` structure. Similarly, the vector structure `thetaPositions` has dimension given by m, the dimension of θ. Each element in the vector structure has a field `parameter` that contains a string with the name of the parameter. The vector structure is constructed such that `thetaPositions(i).parameter` gives the name of the parameter in position i of θ.

The structure `PriorDist` has 6 fields: `beta`, `gamma`, `normal`, `invgamma`, `truncnormal`, and `uniform`. Each such field contains a matrix whose number of rows depends on the number of parameters assigned a given prior distribution. The number of columns is 2 for the normal and the uniform, for the gamma, the inverse gamma, and the left truncated normal the matrix has 3 columns; the third column holds the lower bound. For the beta distribution, finally, it has 4 columns, where the last two hold the lower and upper bounds. Columns 1 and 2 have the values of prior parameter 1 and 2 that were given in the prior distribution file. The `PosteriorModeEstimation` function later creates two new fields in the `PriorDist` structure. These fields are `beta_ab` and `gamma_ab`, containing the (a, b) parameters for the beta (eq. (2.11)) and gamma (eq. (2.6)) distributions, respectively. The functions `MomentToParamStdbetaPDF` and `MomentToParamGammaPDF`, mentioned at the end of Section 2.3, deal with this transformation.

The structure `ParameterNames` has fields `all`, `calibrated`, `beta`, `gamma`, `normal`, `invgamma`, `truncnormal`, `uniform`, and `estimated`. Each field returns a string matrix with the parameter names. And, finally, the matrix `UniformBounds` has dimension $m \times 2$. For all prior distributions but the uniform and the beta each row has 0 and 1, while for the uniform and beta the rows have the lower and the upper bounds.

8.2.2. `logPosteriorPhiDSGE`

Since both `csminwel` and `fminunc` are minimization routines, the function to minimize is given by minus the expression within parenthesis on the right hand side of (8.1). In YADA, this function is called `logPosteriorPhiDSGE`. Before this function is minimized, YADA runs a number of checks on the user defined functions. First of all, all user defined Matlab functions are copied to the `tmp` directory to make sure they are visible to Matlab.

Second, YADA attempts to run the user defined Matlab functions. The first group contains any functions with additional parameters that the user has included in the model setup; cf. Section 6.4. Given that such functions exist, the order in which they are executed depends on the user input on the DSGE Data tab on the YADA GUI; see Figure 1. If both types of additional parameter files exist, the order is determined by the data in the checkbox "Run file with parameters to initialize before file with parameters to update". The execution of additional parameters updates the `ModelParameters` structure with fields and values.

Given that these files are executed without errors (or that they do not exist), the following step is to check the validity of the measurement equation function; cf. Section 6.2. Assuming this function takes the necessary input and provides the necessary output (without errors), YADA then tries to solve the DSGE model with AiM by executing the function `AiMSolver`; see Section 4.2. If the model has a unique convergent solution at the initial values (see Section 4.1), YADA proceeds with the final preparations for running the optimization routine. If not, YADA returns an error message, reporting which problem AiM discovered.

The final preparations first involves collecting additional parameters into the `ParameterNames` structure in the field `additional`. Next, the initial values of the additional parameters as

well as the initial values of the calibrated parameters are located and stored in the vectors `thetaAdditional` and `thetaCalibrated`, respectively. These two tasks are handled by the functions `ReadAdditionalParameterNames` and `ReadAdditionalParameterValues`. Next, the actual sample to use is determined by running the function `CreateSubSample`. This sub-sample does not take into account that the user may have selected a value for the `StartPeriod` variable different from $t_m = 1$; see Section 3.9. The choice of `StartPeriod` is determined by the choice of "First observation after Kalman filter training sample" on the Settings tab of the YADA dialog.

The last task before the chosen minimization routine is called is to check if the function that calculates the log posterior returns a valid value at the initial parameter values. The `logPosteriorPhiDSGE` function takes 10 input arguments. The first is `phi`, the transformed parameters ϕ. Next, 6 inputs originally created by `VerifyPriorData` are required. They are: `thetaIndex`, `UniformBounds`, `LowerBound`, `thetaPositions`, `thetaDist`, and the structure `PriorDist`. Finally, the structure with model parameters `ModelParameters`, the DSGE model structure `DSGEModel`, and the `AIMData` structure are needed. The latter structure is created when the DSGE model is parsed through the `AiMInitialize` function. That function saves to a mat-file the outputs from the `compute_aim_data` function. When this mat-file is loaded into YADA it creates the `AIMData` structure with fields having names equal to the output variables of `compute_aim_data`.

Based on this input the log posterior evaluation function `logPosteriorPhiDSGE` first transforms ϕ into θ by calling `PhiToTheta`. Next, it makes sure that `ModelParameters` is correctly updated for the parameters that are estimated. This is achieved through the function `ThetaToModelParameters`. Apart from `ModelParameters` it needs `theta` and `thetaPositions` to fulfill its task. With `ModelParameters` being updated for the estimated parameters, any remaining parameters are reevaluated next, i.e., the user defined function with parameters to update.

With the parameter point determined, the log-likelihood function is examined and evaluated if the parameter point implies a unique convergent solution for the DSGE model and the state vector is stationary (the largest eigenvalue of F is inside the unit circle). The function `logLikelihoodDSGE` deals with the calculation. The inputs for the function are the three structures `ModelParameters`, `DSGEModel`, and `AIMData`. The function returns three variables: `logLikeValue`, the value of the log-likelihood; `mcode`, indicating if the DSGE model has a unique convergent solution or not; and `status`, a boolean variable, indicating if the F matrix in the state equation (3.2) has all eigenvalues inside the unit circle or not. Given that `mcode` is 1 and that `status` is 0, the value of the log-likelihood is considered valid. If either of these variables returns a different value, the function returns 1000000, otherwise it proceeds with the evaluation of the log of the prior density at $\theta = g(\phi)$ through `logPriorDSGE` and, if the prior density value is not a `NaN`, the log of the Jacobian. The latter function is given by `logJacobian`, presented in Section 7.1. If the log prior density value is `NaN`, then `logPosteriorPhiDSGE` again returns 1000000. Otherwise, the function returns minus the sum of the log-likelihood, the log prior density, and the log Jacobian.

8.2.3. `logLikelihoodDSGE`

The function `logLikelihoodDSGE` directs the main tasks when using the DSGE model for computing the log-likelihood via the Kalman filter. The inputs are, as already mentioned, the three structures `ModelParameters`, `DSGEModel`, and `AIMData`.

First, `logLikelihoodDSGE` runs the `AiMSolver` function. Given that it returns an `mcode` equal to unity, the `AiMtoStateSpace` function is executed. This provides us with the data to determine F and Q that the Kalman filter needs. Next, the measurement equation function is executed to obtain the A, H, and R matrices for the current value of the parameters. Once this task is completed, the `KalmanFilter` function is executed, yielding the outputs `logLikeValue`, and `status`.

8.2.4. logPriorDSGE

The function logPriorDSGE computes the log height of the joint prior density function at a given value of θ. It requires the four inputs theta, thetaDist, PriorDist, and LowerBound. If the value of log prior density is not a real number, logPriorDSGE returns NaN.

8.2.5. YADAcsminwel

Given that the user has chosen to use Christopher Sims' csminwel function, the YADA implementation YADAcsminwel is utilized to minimize the function logPosteriorPhiDSGE. If the optimization algorithm converges within the maximum number of iterations that the user has selected, YADA makes use of the main output variables from this function. First of all, the vector phiMode, with $\tilde{\phi}$, is collected. Furthermore, the value of (minus) the log posterior at the mode is saved into LogPostDensity, while the inverse Hessian is located in the variable InverseHessian. The mode of θ, denoted by $\tilde{\theta}$, is obtained from the parameter transformation function PhiToTheta.

If, for some reason, csminwel is unable to locate the mode, YADA presents the return code message of csminwel indicating what the problem may be.

8.2.6. YADAfminunc*

YADA has three versions of Matlab's fminunc at its disposal. For Matlab versions prior to version 7, an older fminunc function is called: it is named YADAfminunc5 and its original version is dated October 12, 1999. For version 7 and later, YADAfminunc7 is used, and for Matlab versions prior to 7.5 it is originally dated April 18, 2005, while for Matlab version 7.5 and later it is dated December 15, 2006. The YADAfminunc* function attempts to minimize the function logPosteriorPhiDSGE, and if it is successful the vector phiMode, $\tilde{\phi}$, minus the value of the log posterior at the mode, and the Hessian at the mode are provided. This Hessian is inverted by YADA and the results is stored in the variable InverseHessian. The mode of θ, denoted by $\tilde{\theta}$, is obtained from the parameter transformation function PhiToTheta.

Again, if YADAfminunc* fails to locate the posterior mode within the maximum number of iterations, the return code message of fminunc is presented.

YADAfminunc* is only available in the version of YADA that is exclusive to the New Area-Wide Model team at the European Central Bank. As mentioned in Section 8, the publicly available version of YADA does not include these functions.

9. The Random Walk Metropolis Algorithm

The Random Walk Metropolis (RWM) algorithm is a special case of the class of Markov Chain Monte Carlo (MCMC) algorithms popularly called Metropolis-Hastings algorithms; see, Hastings (1970) and Chib and Greenberg (1995) for an overview.

The Metropolis version of this MCMC algorithm is based on a symmetric proposal density, i.e., $q(\theta^*, \theta|Y) = q(\theta, \theta^*|Y)$, while the random walk part follows when the proposal density is symmetric around zero, $q(\theta^*, \theta|Y) = q(\theta^* - \theta, 0|Y)$. The random walk version of the algorithm was originally suggested by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953) and was first used to generate draws from the posterior distribution of DSGE model parameters by Schorfheide (2000) and Otrok (2001).

The description of the RWM algorithm below follows An and Schorfheide (2007) closely.

(1) Compute the posterior mode of ϕ; cf. (8.1). The mode is denoted by $\tilde{\phi}$.
(2) Let $\tilde{\Sigma}$ be the inverse of the Hessian evaluated at the posterior mode $\tilde{\phi}$. YADA actually allows this matrix to be estimated in three different ways and, when used for the RWM algorithm, that its correlations are scaled in a joint fashion towards zero.
(3) Draw $\phi^{(0)}$ from $N_m(\tilde{\phi}, c_0^2\tilde{\Sigma})$, where m is the dimension of ϕ, and c_0 is a non-negative constant. Alternatively, a certain starting value $\phi^{(0)}$ can be specified.

(4) For $s = 1, \ldots, N$, draw φ from the proposal distribution $N_m(\phi^{(s-1)}, c^2\tilde{\Sigma})$, where c is a positive constant. The jump from $\phi^{(s-1)}$ is accepted ($\phi^{(s)} = \varphi$) with probability $\min\{1, r(\phi^{(s-1)}, \varphi|Y)\}$ and rejected ($\phi^{(s)} = \phi^{(s-1)}$) otherwise. The probability $r(\phi, \varphi|Y)$ is determined by

$$r(\phi, \varphi|Y) = \frac{L(Y;\varphi)p(\varphi)J(\varphi)}{L(Y;\phi)p(\phi)J(\phi)}, \tag{9.1}$$

where $J(\phi)$ is the determinant of the Jacobian of $\theta = g(\phi)$; cf. Section 7. Hence, when the value of the numerator in (9.1) is greater than the denominator, then the jump from $\phi^{(s-1)}$ to $\phi^{(s)} = \varphi$ is always accepted.

The expected value of any real valued function $h(\phi)$ is approximated by $N^{-1}\sum_{s=1}^{N} h(\phi^{(s)})$. Hence, the posterior mean and covariance can be calculated directly from the RWM output. Furthermore, the numerical standard error of the posterior mean can be calculated using the Newey and West (1987) estimator.

The numerical standard error of the posterior mean $\bar{\phi}$ is computed as follows. Let $\bar{N} \leq N$ be an integer such that the autocorrelation function for the posterior draws of ϕ tapers off. Consider the matrix

$$\hat{\Sigma}_\phi = \frac{1}{N}\sum_{s=-\bar{N}}^{\bar{N}} \frac{\bar{N} + 1 - |s|}{\bar{N} + 1}\Gamma(s), \tag{9.2}$$

where

$$\Gamma(s) = \frac{1}{N}\sum_{n=s+1}^{N} \left(\phi^{(n)} - \bar{\phi}\right)\left(\phi^{(n-s)} - \bar{\phi}\right)', \quad \text{if } s \geq 0,$$

and $\Gamma(s) = \Gamma(|s|)'$ when $s \leq -1$. The numerical standard error of $\bar{\phi}$ is given by the square root of the diagonal elements of $\hat{\Sigma}_\phi$. The matrix $\hat{\Sigma}_\phi$ has the usual property that it converges to 0 in probability as $N \to \infty$ when the Markov chain that has generated the $\phi^{(s)}$ sequence is ergodic; see, e.g. Tierney (1994).

9.1. *YADA Code*

9.1.1. `NeweyWestCovMat`

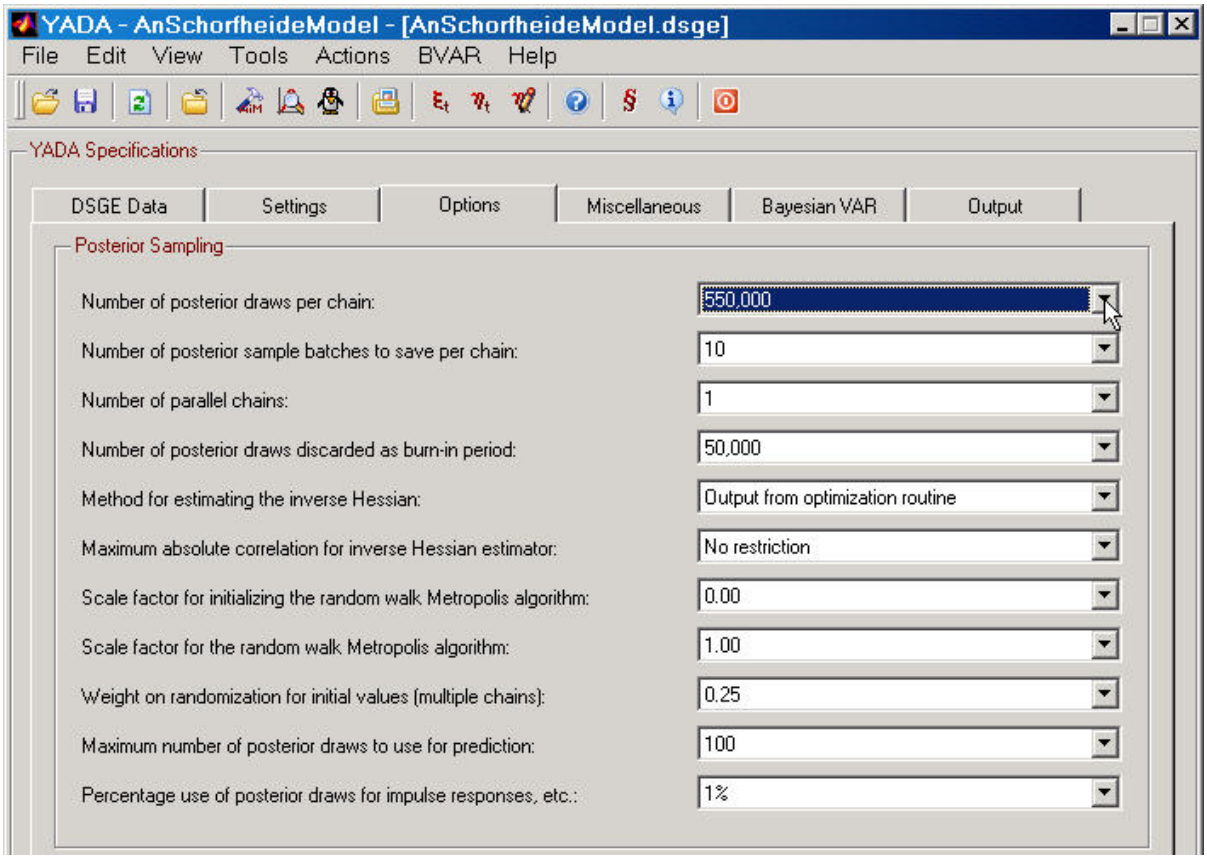
The function `NeweyWestCovMat` computes the numerical standard error of any sampled real valued vector using the Newey and West (1987) corrected covariance matrix. The only required input argument is `PostSample`, a matrix of dimension $N \times m$, with N being the number of sample points and m the dimension of the vector. The function also accepts the input `BarN`, representing $\bar{N}$ in (9.2). If this input is either missing or assigned an empty value, the function sets $\bar{N} = [N^{(1/2.01)}]$, unless $[N^{(1/2.01)}] < 100$ and $N > 200$ when $\bar{N} = 100$ is selected. This ensures that $\lim_{N\to\infty} \bar{N} = +\infty$, while $\lim_{N\to\infty} \bar{N}^2/N = 0$; see, e.g., Geweke (2005, Theorem 4.7.3). The output is given by `StdErr`, an $m \times m$ covariance matrix such that the numerical standard errors are available as the square root of the diagonal elements.

9.1.2. `DSGEPosteriorSampling`

The function `DSGEPosteriorSampling` handles the actual run of the random walk Metropolis algorithm. It uses the same inputs as `PosteriorModeEstimation`, i.e., `DSGEModel`, `CurrINI`, `controls`. One important difference relative to the posterior mode estimation function is that some fields in the `DSGEModel` structure are ignored in favor of values saved to disk while running the posterior mode estimation routine. In particular, YADA stores data about the prior distribution and sample dates information to disk and `DSGEPosteriorSampling` makes sure that the same prior and dates are used when sampling from the posterior as when estimating the posterior mode.

Before starting up the RWM algorithm, the function performs a number of tests. First, it attempts to execute the additional parameter functions that are present. If they run without giving any errors, the measurement equation function is executed next and, thereafter, YADA

FIGURE 3. The Posterior Sampling frame on the Options tab in YADA.



attempts to solve the DSGE model at the posterior mode estimate. Given that all checks return positive results, the log posterior function `logPosteriorPhiDSGE` is executed for the posterior mode estimate of ϕ.

The RWM algorithm in YADA has been designed to be flexible and to avoid computing things more often than necessary. The *Posterior Sampling* frame on the Options tab in displayed in Figure 3. The total number of draws from the posterior per sampling chain can be selected as well as the number of sample batches per chain. YADA always stores the data to disk when a sample batch has been completed. The number of draws in each batch depends on the total number of draws per chain and the number of sample batches per chain. This allows the user to abort a run and later on restart the posterior sampler from the last saved position.

Furthermore, the number of sampling chains can be selected as well as the length of the burn-in period for the sampler. The draws obtained during the burn-in period are later on discarded from the total number of posterior draws per chain, although they will still be saved to disk.

The selections thereafter turns to the proposal density. First, the method for estimating the inverse Hessian at the posterior mode can be selected. YADA makes use of the output from the selected optimization routine by default. Alternatively, YADA can fit a quadratic to the evaluated log posterior to estimate the diagonal of the inverse Hessian; cf. Section 8.1. In addition, when the option "Transform conditional standard deviations for modified Hessian to marginal using correlations from Hessian" on the *Miscellaneous* tab is check marked, then these estimates are scaled up accordingly. For both possibilities, the correlation structure is thereafter taken from the inverse Hessian that the optimization routine provides. Third, a finite difference estimator can be applied. Here, the step length is determined by the user and this selection is located on the *Miscellaneous* tab. Finally, a user specified parametere covariance matrix for the proposal density is also supported. Such a matrix may, for instance, be estimated using old draws from the posterior distribution. YADA supports this feature from the *View* menu, but any user defined

matrix is also allowed provided that it is stored in a mat-file and that the matrix has the name `ParameterCovarianceMatrix`.

The estimator of the inverse Hessian can be influenced through a parameter that determines its maximum absolute correlation. This parameter is by default set to 1 (no restriction), but values between 0.95 and 0 can also be selected. This parameter interacts with the largest absolute correlation for the inverse Hessian such that the off-diagonal elements of the new inverse Hessian are given by the old off-diagonal elements times the minimum of 1 and the ratio between the desired maximum absolute correlation and the estimated maximum absolute correlation. Hence, if the desired maximum absolute correlation is greater than the estimated maximum absolute correlation then the inverse Hessian is not affected. On the other hand, if the ratio between these correlations is less than unity, then all correlations are scaled towards zero by with the ratio. At the extreme, all correlations can be set to zero by selecting a maximum absolute correlation of zero.

The following two selections concern the c_0 and c scale parameters for the initial density and for the proposal density, respectively. The selection of the c parameter influences greatly the sample acceptance probability. If you consider this probability to be too low or high, then changing c will often help; see, e.g., Adolfson, Lindé, and Villani (2007a). The c_0 parameter gives the user a possibility to influence the initial value $\phi^{(0)}$. For instance, if $c_0 = 0$, then $\phi^{(0)} = \tilde{\phi}$, i.e., the posterior mode.

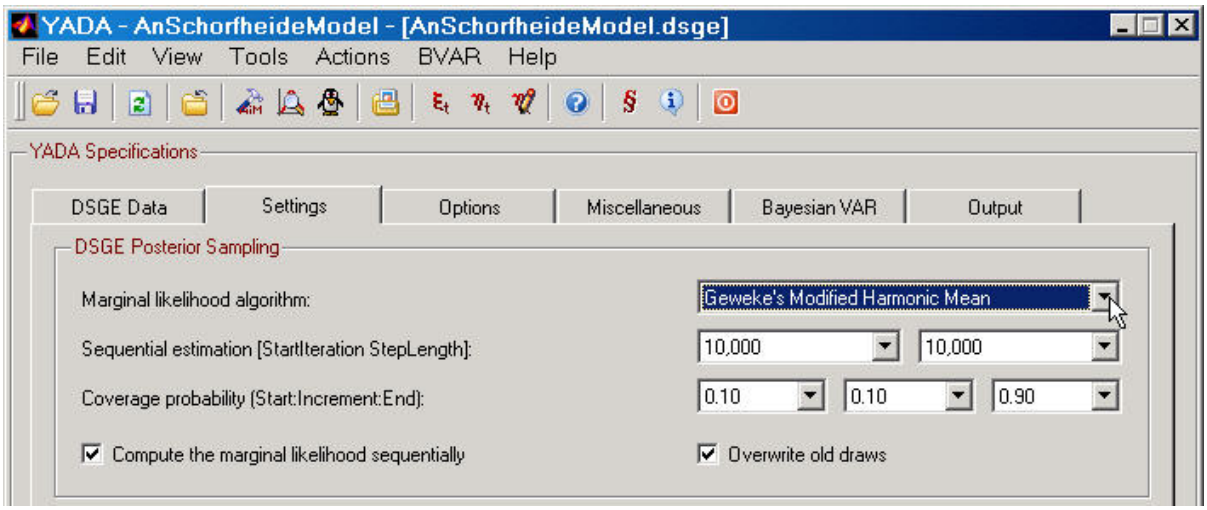
The next parameter in the *Posterior Sampling* frame is only used under multiple sampling chains. The weight on randomization refers to the weight given to a randomly drawn ϕ, determined as in the case of the single chain but with $c_0 = 4$, relative to the posterior mode when setting up $\phi^{(0)}$. Hence, if the weight on randomization is 1, then each sampling chain uses $\phi^{(0)}$ from $N_m(\tilde{\phi}, 16\tilde{\Sigma})$ and if the weight on randomization is 0, then each sampling chain starts from the posterior mode. Hence, the weight on randomization is identical to c_0 except that it is restricted to the 0-4 interval. Since multiple chains are used to check, for instance, convergence related issues, it is not recommended to start all chains from the posterior mode.

The following parameter on this frame determines the maximum number of draws from the posterior that will be used in prediction exercises. When comparing the number of posterior draws minus the length of the burn-in period to the desired maximum number of draws to use in such exercises YADA selects the smallest of these two numbers. Among the stored posterior draws the parameter values used in prediction are obtained by either using a fixed interval (default) or by drawing randomly from the available draws using a uniform distribution. The length of the fixed interval is the maximum possible to ensure desired number of parameter draws is possible. The option "Randomize draws from posterior distributions" on the *Miscellaneous* tab determines which option is used.

The final parameter on the Posterior Sampling frame is the percentage use of posterior draws for impulse responses, variance decompositions, etc. It allows the user to consider only a share of the available posterior draws when computing the posterior distributions of such functions. It may be useful to make use of a small share of the number of available draws when preliminary results are desired or when the user is mainly concerned with point estimates, such as the posterior mean. As above for the prediction case, if fewer than 100 percent are used, the user can choose between the largest fixed interval between draws (default) or uniform draws from the posterior draws.

One additional user determined parameter influences how the RWM algorithm is executed. This parameter in located on the *DSGE Posterior Sampling* frame on the Settings tab; see Figure 4. If the checkbox *Overwrite old draws* is check marked, then previous draws will be overwritten. Conversely, when this box is not checked, then old draws will be used. This allows the user to recover from a previously aborted run of the RWM algorithm provided that the number of sample batches is greater than 1 and that at least one batch was saved to disk.

FIGURE 4. The DSGE Posterior Sampling frame on the Settings tab in YADA.



There is also another case when YADA can recover previously saved posterior draws. Given that the checkbox *Overwrite old draws* is check marked and the posterior draws are only obtained from one sampling chain, YADA will check if, for your current selection about the number of sample batches, posterior draws exist on disk such that the number of posterior draws is lower than what you have currently selected. For instance, suppose you have selected to save 10 sample batches per chain and that you currently consider 100,000 posterior draws. YADA will then check if you have previous run the posterior sampler with less than 100,000 draws for 10 sample batches. The highest such number of posterior draws will then be considered as a candidate. Supposing that you have already run the posterior sampler for 75,000 draws with 10 sample batches, YADA will ask you if you would like to make use of these 75,000 draws.

When the RWM algorithm has finished, YADA first allows for (but does not force) estimation of the marginal likelihood. The alternative estimators are discussed in Section 11 below. The choice of algorithm is stated on the *DSGE Posterior Sampling* frame, where certain other parameters that are needed by the marginal likelihood estimation function can also be selected.

Before `DSGEPosteriorSampling` has completed its mission it sends the results to a function that writes a summary of them to file. This file is finally displayed for the user.

10. MARKOV CHAIN MONTE CARLO CONVERGENCE

This section is concerned with assessing the convergence of the MCMC sampler. A fundamental problem of inference from Markov chain simulation is that there will always be areas of the posterior (target) distribution that have not been covered by the finite chain. As the simulation progresses the ergodic property of the Markov chain causes it eventually to cover all the target distribution but, in the short run, the simulations cannot, generally, tell us about areas where they have not been. As pointed out by, e.g., Gelman (1996), this is a general problem whenever convergence is slow, even in a distribution that has a single mode; see, also, Gelman and Rubin (1992).

This section is divided into two parts. The first deals with tools for evaluating convergence using a single chain, while the following is concerned with multiple chains. For a review of various tools for evaluating MCMC convergence see, e.g., Cowles and Carlin (1996).

10.1. *Single Chain Convergence Statistics*

The simplest tool for assessing if a MCMC chain has converged or not is to view graphs of the raw draws; see, e.g., Adolfson, Laséen, Lindé, and Villani (2007b). If the draws are trending this is a strong indication that the sampler has not converged. In fact, the reason for the trending

may be that the DSGE model is misspecified. The raw draws may also be graphed directly from multiple chains to check if they cover the same region of the parameter space.

Next, sequential estimates of various location parameters may be examined. YADA allows for sequential estimation of the posterior mean and the posterior median. If the sequential estimates are trending this is indicative of poor convergence. Furthermore, sequential estimates of the marginal likelihood may also be studied; see Section 11.

A common tool in time series analysis for studying parameter stability is the partial sum or cusum. Yu and Mykland (1998) proposed to use this tool for evaluating convergence. Like the sequential estimates of the posterior mean, median, and marginal likelihood, the cusum estimates depend a first determining a burn-in period. Let $S(X)$ be a chosen summary statistic of the N post burn-in posterior draws. With $\hat{\mu}$ being the average of $S(X)$ for all N draws, the observed cusum is

$$\hat{C}_i = \sum_{j=1}^{i} \big(S(X_j) - \hat{\mu}\big), \quad i = 1, \ldots, N. \tag{10.1}$$

The cusum path plot is obtained by plotting $\{\hat{C}_i\}$ against $i = 1, \ldots, N$. If N is very large it may be practical to plot the statistic against, say, $i = N_0, 2N_0, \ldots, N$ instead for some suitable integer N_0.

The cusum statistic in (10.1) is zero for $i = N$. In YADA the value of $\hat{\mu}$ is added to $\hat{C}_i$ and the summary statistics are either the log posterior $(\ln L(Y; g(\phi)) + \ln p(g(\phi)) + \ln J(\phi))$, the original parameters (θ), or the transformed parameters (ϕ). Moreover, YADA calculates moving window cusum paths for a fixed window size of $N_1 = N/10$, i.e.,

$$\bar{C}_i = \sum_{j=i+1-N_1}^{i} \big(S(X_j) - \hat{\mu}\big), \quad i = N_1, \ldots, N, \tag{10.2}$$

where, again, $\hat{\mu}$ is added to $\bar{C}_i$.

A separated partial means test for a single MCMC chain has been suggested by Geweke; see, e.g., Geweke (2005, Theorem 4.7.4). Let N be the number of draws and suppose that $N_p = N/2p$ and p are positive integers. For instance, with $N = 10,000$ and $p = 5$ we have that $N_5 = 1,000$. Define the p separated partial means:

$$\hat{S}_{j,p}^{(N)} = \frac{1}{N_p} \sum_{m=1}^{N_p} S\big(\phi^{(m+N_p(2j-1))}\big), \quad j = 1, \ldots, p, \tag{10.3}$$

where S is some summary statistic of the transformed parameters ϕ (such as the original parameters θ). Let $\hat{\tau}_{j,p}$ be the Newey and West (1987) numerical standard error for $j = 1, \ldots, p$. Define the $(p-1)$ vector $\hat{S}_p^{(N)}$ with typical element $\hat{S}_{j+1,p}^{(N)} - \hat{S}_{j,p}^{(N)}$ and the $(p-1) \times (p-1)$ tridiagonal matrix $\hat{V}_p^{(N)}$ where

$$\hat{V}_{j,j}^{(N)} = \hat{\tau}_{j,p}^2 + \hat{\tau}_{j+1,p}^2, \quad j = 1, \ldots, p-1$$

and

$$\hat{V}_{j,j+1}^{(N)} = \hat{V}_{j+1,j}^{(N)} = \hat{\tau}_{j+1,p}^2, \quad j = 1, \ldots, p-1.$$

The statistic

$$G_p^{(N)} = \hat{S}_p^{(N)\prime} \big[\hat{V}_p^{(N)}\big]^{-1} \hat{S}_p^{(N)} \xrightarrow{\text{d}} \chi^2(p-1), \tag{10.4}$$

as $N \to \infty$ under the hypothesis that the MCMC chain has converged with the separated partial means being equal.

10.2. *Multiple Chain Convergence Statistics*

One approach for monitoring convergence using draws from multiple MCMC chains is to use analysis of variance. The approach outlined below is based on Brooks and Gelman (1998), which generalizes the ideas in Gelman and Rubin (1992); see also Gelman (1996) and Gelman et al. (2004).

For a univariate scalar summary S we may assume that we have N draws from M chains. Let S_{ij} denote draw i from chain j. We may then define the average of S for chain j as $\bar{S}_j = (1/N)\sum_{i=1}^{N} S_{ij}$, while the overall average is $\bar{S} = (1/M)\sum_{j=1}^{M} \bar{S}_j$. The between-chain variance B and the within-chain variance W are now given by

$$B = \frac{N}{M-1}\sum_{j=1}^{M}\left(\bar{S}_j - \bar{S}\right)^2, \tag{10.5}$$

and

$$W = \frac{1}{M(N-1)}\sum_{j=1}^{M}\sum_{i=1}^{N}\left(S_{ij} - \bar{S}_j\right)^2. \tag{10.6}$$

The between-chain variance B contains a factor of N because it is based on the variance of the within-chain means, $\bar{S}_j$, each of which is an average of N draws S_{ij}.

From the two variance components two estimates of the variance of S in the target distribution, Σ_S, can be constructed. First

$$\hat{\Sigma}_S = \frac{N-1}{N}W + \frac{1}{N}B, \tag{10.7}$$

is an unbiased estimate of the variance under the assumption of stationarity, i.e., when the starting points of the posterior draws are actually draws from the target distribution. Under the more realistic assumption that the starting points are overdispersed, then (10.7) is an *overestimate* of the variance of S.

Second, for any finite N, the within-chain variance W in (10.6) should *underestimate* the variance of S. In the limit as $N \to \infty$, both $\hat{\Sigma}_S$ and W approach Σ_S, but from opposite directions. Accounting for sampling variability of the estimator $\bar{S}$ yields a pooled posterior variance estimate of $\hat{V} = \hat{\Sigma}_S + B/(MN)$.

To monitor convergence of the posterior simulation Gelman and Rubin (1992) therefore suggest estimating the ratio of the upper and the lower bounds of the variance of S through:

$$\hat{R} = \sqrt{\frac{\hat{V}}{W}} = \sqrt{\frac{N-1}{N} + \frac{(M+1)B}{MNW}}. \tag{10.8}$$

As the simulation converges, the *potential scale reduction factor* in (10.8) declines to 1. This means that the M parallel Markov chains are essentially overlapping.

The scalar summary S can here be individual parameters of the DSGE model. A multivariate version of the potential scale reduction factor is suggested by Brooks and Gelman (1998). Now S is, e.g., a vector of all the model parameters, with B and W being covariance matrix estimators of the between-chain and within-chain covariance. The multivariate potential scale reduction factor (MPSRF) is now:

$$\hat{R} = \sqrt{\frac{N-1}{N} + \frac{M+1}{M}\lambda_1}, \tag{10.9}$$

where λ_1 is the largest eigenvalue of the positive definite matrix $(1/N)W^{-1}B$; see Brooks and Gelman (1998, Lemma 2). The *multivariate potential scale reduction factor* in (10.9) declines towards 1 as the simulation converges. Gelman et al. (2004) suggests that values for $\hat{R}$ less than 1.1 may be regarded as an indication that the MCMC sampler has converged.

In addition to monitoring the MPSRF in (10.9) Gelman et al. (2004) also suggest to monitor the determinants of W and B. This allows the user to also check if both the within-chain covariance matrix W and the between-chain covariance matrix B stabilize as functions of N.

10.3. *YADA Code*

10.3.1. `CUSUM`

The function `CUSUM` computes the cusum paths in equation (10.1) and (10.2) for the values of the log posterior from the MCMC output, the draws of the original parameters (θ), or of the transformed parameters (ϕ). The function takes 4 input variables: `X`, `NumBurnin`, `PlotType`, and

CurrINI. The matrix X has dimension NumIter × NumStat, where NumIter is the total number of draws from the posterior, and NumStat is the number of summary statistics. The integer NumBurnin gives the number of burn-in draws from the MCMC chain, while the string vector PlotType can be either log posterior, original parameters, or transformed parameters. Finally, the structure CurrINI contains initialization information.

The function provides 2 output arguments: CUSUMPost and CUSUMAverage. The former has dimension (NumIter − NumBurnin) × NumStat and holds the cusum statistic in (10.1) plus the mean of the input statistics over the post-burn-in sample. The second output variables has dimension (0.9NumIter − NumBurnin) × NumStat with the moving window cusum statistic in (10.2).

10.3.2. SeparatedPartialMeansTest

The function SeparatedPartialMeansTest calculates the separated partial means test in (10.4). The function needs 4 input variables: PostSample, p, PlotType, and CurrINI. The matrix PostSample has dimension NumIter × NumParam, where NumIter is now the number of post-burn-in period draws from the posterior while NumParam is the number of summary statistics. The integer p gives the number of separated partial means to evaluate. The last two input arguments are the same as in the case of the CUSUM function.

As output the function gives a matrix GewekeStat that has dimension NumParam × 2. The first column holds the values for the separated partial means test for the NumParam summary statistics, and the second column the asymptotic p-values based on applying the $\chi^2(p-1)$ distribution.

10.3.3. MultiANOVA

The function MultiANOVA computes the multivariate potential scale reduction factor in (10.9) as well as the determinants of $\hat{V}$ and W sequentially. The function takes 5 input variables: NumChains, ComputeSequential, DSGEModel, CurrINI, and controls. The last three input variables are also used by the functions for estimating the posterior mode and running the random walk Metropolis algorithm. The first input simply gives the number of parallel MCMC chains to examine (M in Section 10.2), while the second variables is boolean and takes on the value 1 if the output should be calculated sequentially and 0 otherwise. The output statistics are calculated from posterior draws of the original parameters θ, which are loaded from disk, one chain at a time.

As output the function gives two variables: DoneCalc and MPSRF. The first is a boolean variable which indicates if the calculations were finished (1) or not (0). The matrix MPSRF has 4 columns, where the first holds the number of draws used in the calculation, the second the $\hat{R}$ value, the third the determinant of $\hat{V}$, while the fourth holds the determinant of W. The rows correspond to the sample sizes used for the sequential estimates, as can be read from the first column.

11. Computing the Marginal Likelihood

11.1. *The Laplace Approximation*

The Laplace (or, more precisely, the Laplace-Metropolis) approximation of the log marginal likelihood, $\ln p(Y)$, was originally suggested by Tierney and Kadane (1986); see, also, Raftery (1996) for discussions. It requires an estimate of the posterior mode of the parameters and of the inverse Hessian at the mode. The Laplace estimator of the marginal likelihood makes use of a normal approximation and YADA considers only the transformed parameters ϕ. With the inverse Hessian evaluated at the mode being given by $\tilde{\Sigma}$ and $\tilde{\phi}$ being the posterior mode of the transformed parameters, the Laplace approximation of the log marginal likelihood is given by:

$$\ln \hat{p}_L(Y) = \ln L\big(Y; g(\tilde{\phi})\big) + \ln p\big(g(\tilde{\phi})\big) + \ln J\big(\tilde{\phi}\big) + \frac{m\ln(2\pi) + \ln\left|\tilde{\Sigma}\right|}{2}, \tag{11.1}$$

where m is the dimension of ϕ. Notice that since YADA calculates the posterior mode by minimizing minus the log of the posterior, the inverse Hessian does not need to be multiplied by minus 1 in (11.1). For the Bayes factor, i.e., the ratio of the marginal likelihoods of two models, the relative error in (11.1) was shown by Tierney and Kadane (1986) to be $O(T^{-1})$ for regular statistical models.

It is also interesting to note that if the inverse Hessian of the log posterior for the θ parameter is approximated through the delta method, then the Laplace approximation of the log marginal likelihood using $\tilde{\theta}$ is identical to the expression on the right hand side of (11.1). This follows directly from noting that the determinant of the inverse Hessian is now $J(\tilde{\phi})^2|\tilde{\Sigma}|$, while the log posterior is $\ln L(Y;\tilde{\theta}) + \ln p(\tilde{\theta})$.

11.2. *The Modified Harmonic Mean Estimator*

Harmonic mean estimators are based on the identity:

$$p(Y) = \left[\int \frac{f(\theta)}{p(Y|\theta)p(\theta)} p(\theta|Y)d\theta\right]^{-1}, \tag{11.2}$$

where $f(\theta)$ is such that $\int f(\theta)d\theta = 1$; see Gelfand and Dey (1994). Given a choice for $f(\theta)$, the marginal likelihood $p(Y)$ can then be estimated using:

$$\hat{p}_H(Y) = \left[\frac{1}{N}\sum_{s=1}^{N} \frac{f\left(\theta^{(s)}\right)}{L\left(Y;\theta^{(s)}\right)p\left(\theta^{(s)}\right)}\right]^{-1}, \tag{11.3}$$

where $\theta^{(s)}$ is a draw from the posterior distribution and N is the number of draws. As noted by, e.g., An and Schorfheide (2007), the numerical approximation is efficient if $f(\theta)$ is selected such that the summands are of equal magnitude.

Geweke (1999) suggested to use the density of a truncated multivariate normal distribution in (11.3). That is, for $0 < p < 1$

$$f(\theta) = \frac{\exp\left[-(1/2)(\theta-\bar{\theta})'\Sigma_\theta^{-1}(\theta-\bar{\theta})\right]}{p(2\pi)^{m/2}|\Sigma_\theta|^{1/2}} \left\{(\theta-\bar{\theta})'\Sigma_\theta^{-1}(\theta-\bar{\theta}) \leq \chi_p^2(m)\right\}, \tag{11.4}$$

where $\bar{\theta}$ is the mean and Σ_θ the covariance matrix from the output of the posterior simulator, i.e., $\bar{\theta} = N^{-1}\sum_{s=1}^{N}\theta^{(s)}$ and $\Sigma_\theta = N^{-1}\sum_{s=1}^{N}\theta^{(s)}\theta^{(s)\prime} - \bar{\theta}\bar{\theta}'$. The expression $\{a \leq b\}$ is 1 if true and 0 otherwise, while $\chi_p^2(m)$ is the $100p$ percent quantile value of the χ^2 distribution with m degrees of freedom.

Since YADA works internally with the transformed ϕ parameters, the expression for the modified harmonic mean estimator of the marginal likelihood is slightly different. Specifically,

$$\hat{p}_{MH}(Y) = \left[\frac{1}{N}\sum_{s=1}^{N} \frac{f\left(\phi^{(s)}\right)}{L\left(Y;g(\phi^{(s)})\right)p\left(g(\phi^{(s)})\right)J\left(\phi^{(s)}\right)}\right]^{-1}, \tag{11.5}$$

where $f(\phi)$ is the same as in equation (11.4) but with $\bar{\theta}$ and Σ_θ replaced with the posterior mean and covariance of ϕ, i.e., $\bar{\phi}$ and Σ_ϕ.

11.3. *The Chib and Jeliazkov Estimator*

The Chib and Jeliazkov (2001) estimator of the marginal likelihood starts from the so called marginal likelihood identity

$$p\left(Y\right) = \frac{L\left(Y;\theta\right)p\left(\theta\right)}{p\left(\theta|Y\right)} = \frac{L\left(Y;g(\phi)\right)p\left(g(\phi)\right)J\left(\phi\right)}{p\left(\phi|Y\right)}. \tag{11.6}$$

This relation holds for any value of ϕ (and θ), but a point with high posterior density should preferably be selected, e.g., the posterior mode $\tilde{\phi}$.

The numerator of (11.6) can be determined directly once the posterior mode has been found. The denominator, however, requires a numerical approximation. Hence,

$$\hat{p}_{CJ}\left(Y\right) = \frac{L\left(Y; g(\tilde{\phi})\right)p\left(g(\tilde{\phi})\right)J\left(\tilde{\phi}\right)}{\hat{p}\left(\tilde{\phi}|Y\right)}, \tag{11.7}$$

where $\hat{p}(\tilde{\phi}|Y)$ remains to be calculated.

Based on the definition of $r(\phi, \varphi|Y)$ in equation (9.1), let

$$\alpha\left(\phi, \varphi|Y\right) = \min\left\{1, r\left(\phi, \varphi|Y\right)\right\}. \tag{11.8}$$

Let $q(\phi, \tilde{\phi}|Y)$ be the proposal density for the transition from ϕ to $\tilde{\phi}$. For the random walk Metropolis algorithm we have used the multivariate normal density as the proposal, i.e.,

$$q\left(\phi, \tilde{\phi}|Y\right) = \left(2\pi\right)^{-m/2}\left|c^2\tilde{\Sigma}\right|^{-1/2}\exp\left\{-\frac{1}{2c^2}\left(\tilde{\phi}-\phi\right)'\tilde{\Sigma}^{-1}\left(\tilde{\phi}-\phi\right)\right\}. \tag{11.9}$$

This density is symmetric, i.e., $q(\phi, \tilde{\phi}|Y) = q(\tilde{\phi}, \phi|Y)$.[19] The posterior density at the mode can now be approximated by

$$\begin{aligned}\hat{p}\left(\tilde{\phi}|Y\right) &= \frac{N^{-1}\sum_{s=1}^{N}\alpha\left(\phi^{(s)}, \tilde{\phi}|Y\right)q\left(\phi^{(s)}, \tilde{\phi}|Y\right)}{J^{-1}\sum_{j=1}^{J}\alpha\left(\tilde{\phi}, \phi^{(j)}|Y\right)} \\ &= \frac{N^{-1}\sum_{s=1}^{N}q\left(\phi^{(s)}, \tilde{\phi}|Y\right)}{J^{-1}\sum_{j=1}^{J}\alpha\left(\tilde{\phi}, \phi^{(j)}|Y\right)},\end{aligned} \tag{11.10}$$

where $\phi^{(s)}$, $s = 1, \ldots, N$ are sampled draws from the posterior distribution with the RWM algorithm, while $\phi^{(j)}$, $j = 1, \ldots, J$, are draws from the proposal density (11.9). The second equality stems from the fact that $\alpha(\phi^{(s)}, \tilde{\phi}|Y) = 1$ for all $\phi^{(s)}$ when $\tilde{\phi}$ is the posterior mode, i.e., the transition from $\phi^{(s)}$ to $\tilde{\phi}$ is always accepted by the algorithm. Hence, the Chib and Jeliazkov estimator of $p(\tilde{\phi}|Y)$ is simply the sample average of the proposal density height for the accepted draws relative to the posterior mode, divided by the sample average of the acceptance probability, evaluated at the posterior mode.

The parameter J is always equal to N in YADA. In contrast with the modified harmonic mean estimator of the marginal likelihood, the Chib and Jeliazkov estimator requires J additional draws. When J is large and the parameter space is high dimensional, the Chib and Jeliazkov estimator will be considerably slower to compute since the log posterior function on the right hand side of equation (8.1) needs to be evaluated an additional J times.

The numerical standard error of the log of the marginal likelihood estimate $\hat{p}(\tilde{\phi}|Y)$ in (11.10) can be computed from the vectors

$$h^{(s,j)} = \begin{bmatrix} h_1^{(s,j)} \\ h_2^{(s,j)} \end{bmatrix} = \begin{bmatrix} q\left(\phi^{(s)}, \tilde{\phi}|Y\right) \\ \alpha\left(\tilde{\phi}, \phi^{(j)}|Y\right) \end{bmatrix}.$$

The average of $h^{(s,j)}$ is denoted by $\hat{h}$. This means that

$$\ln\hat{p}\left(\tilde{\phi}|Y\right) = \ln\hat{h}_1 - \ln\hat{h}_2,$$

and the numerical standard error of the log marginal likelihood estimate can be calculated from the sample variance of $h^{(s,j)}$ via the delta method. The sample variance of the latter can be computed using the Newey and West (1987) estimator.

11.4. *YADA Code*

Geweke's (1999) modified harmonic mean estimator of the marginal likelihood is computed in YADA with the function `MargLikeModifiedHarmonic`. This function can also be used to estimate the marginal likelihood sequentially. The nature of the sequential estimation is quite

[19] Notice that $\alpha(\phi, \varphi|Y) = \min\{1, r(\phi, \varphi|Y)q(\varphi, \phi|Y)/q(\phi, \varphi|Y)\}$ in Chib and Jeliazkov (2001); see, e.g., Section 2.1, above equation (7).

flexible, where a starting value and an incremental value for the sequence can be selected on the Settings tab. By default, YADA sets the starting value to 100 and the increment value to 100. For a posterior sample with 10000 draws, this means that the marginal likelihood is estimated for the sample sizes $100, 200, \ldots, 9900, 10000$. The selection of sequential estimation sample is determined on the *DSGE Posterior Sampling* frame on the Settings tab; cf. Figure 4.

Similarly, the function `MargLikeChibJeliazkov` calculates the Chib and Jeliazkov (2001) estimator of the marginal likelihood as well as its numerical standard error. Like the modified harmonic mean estimator function, `MargLikeModifiedHarmonic`, the calculations can be performed sequentially using the same sample grid.

The Laplace approximation is calculated by `MargLikeLaplace`. It is only run towards the end of the posterior mode estimation routine and should only be viewed as a quick first order approximation when comparing models.

11.4.1. `MargLikeLaplace`

The function `MargLikeLaplace` takes 2 inputs. First, it requires the value of the log posterior at the mode of ϕ, `LogPost`, and second minus the inverse of the Hessian at the mode, `InvHessian`. Based on equation (11.1) the log marginal likelihood is calculated and provides as the output `LogMarg`.

11.4.2. `MargLikeModifiedHarmonic`

The function `MargLikeModifiedHarmonic` takes 5 inputs. First of all, `PostSample`, an $N \times m$ matrix with N posterior draws of the parameters. Next, the values of the log of the posterior are needed. These are assumed to be given by the N dimensional vector `LogPost`. Third, the function accepts a boolean variable `ComputeSequential` that is 1 if the marginal likelihood should be estimated sequentially and 0 otherwise. Fourth, a vector with coverage probabilities are needed. This vector, denoted by `CovProb`, can be empty or contain numbers between 0 and 1. Finally, the function requires the structure `DSGEModel` with model related information. This structure should contain the fields `SequentialStartIterationValue` and `SequentialStepLengthValue`, where the former gives the starting value of the sequential estimates and the latter gives the increment value. In case `CovProb` is empty, the structure should also include the fields `CovStartValue`, `CovIncValue`, and `CovEndValue`. These fields determine the starting probability value, the increment and the upper bound of the coverage probability p in (11.4); cf. the DSGE Posterior Sampling frame on the Settings tab in Figure 4.

The output of `MargLikeModifiedHarmonic` is given by the matrix `LogMargs` and the vector `CovProb`. The dimension of the former is given by the number of successful computations of the marginal likelihood for the given coverage probabilities times the number of coverage probabilities plus 1. The first column of this matrix contains the number of draws used for the computations, while the remaining columns contains the marginal likelihood values for each given coverage probability. The second output argument is simply the vector with coverage probabilities that was used by the function.

11.4.3. `MargLikeChibJeliazkov`

The function `MargLikeChibJeliazkov` needs 17 input arguments. The first two are the matrix `PostSample` and the vector `LogPost` that are also used by `MargLikeModifiedHarmonic`. Next, the posterior mode $\tilde{\phi}$ and the inverse Hessian at the posterior mode $\tilde{\Sigma}$ are needed, along with the scale factor c that is used by the proposal density. These inputs are denoted by `phiMode`, `SigmaMode`, and `c`, respectively. Furthermore, the function takes the inputs `logPostMode` (the value of the log posterior at the mode), `NumBurnin` (number of burn-in draws), and the boolean variable `ComputeSequential` that is also used by `MargLikeModifiedHarmonic`.

The remaining 9 input arguments are the last 9 inputs for the `logPosteriorPhiDSGE` function, used to compute the $\alpha(\tilde{\phi}, \phi^{(j)}|Y)$ term in the denominator of equation (11.10). The function `MargLikeChibJeliazkov` always sets $J = N$ in (11.10).

The output matrix is given by `LogMargs`. The first column gives the number of draws used for estimating the marginal likelihood, the second column the estimated value of the log marginal likelihood, while the numerical standard error of the log marginal likelihood is provided in the third column. The standard error is computed using the Newey and West (1987) estimator, with $\bar{N} = N^{(1/2.01)}$.

12. Bayesian VAR Analysis

Bayesian VAR models can be analysed with YADA. Their main purpose is to provide a competitor when performing multistep out-of-sample forecasts with a DSGE model. Let x_t be a p dimensional covariance stationary vector stochastic process which satisfies the dynamic relation:

$$x_t = \Psi d_t + \sum_{l=1}^{k} \Pi_l \left(x_{t-l} - \Psi d_{t-l} \right) + \varepsilon_t, \quad t = 1, \ldots, T. \tag{12.1}$$

The vector d_t is deterministic and assumed to be of dimension q. The residuals ε_t are assumed to be i.i.d. Gaussian with zero mean and positive definite covariance matrix Ω. The Π_l matrix is $p \times p$ for all lags, while Ψ is $p \times q$ and measures the expected value of x_t conditional on the parameters and other information available at $t = 0$. All Bayesian VAR models that are supported by YADA have an informative prior on the Ψ parameters, the steady state of x_t. Moreover, the elements of the vector x_t (d_t) are all elements of the vector y_t (x_t) in the measurement equation of the DSGE model. It is hoped that this notational overlap will not be confusing for you.

12.1. *The Prior*

The setup of the VAR model in (12.1) is identical to the stationary VAR process in mean-adjusted form that Villani (2007) examines. The prior on the steady state Ψ is also the same as that considered in his paper and used by, e.g., Adolfson, Anderson, Lindé, Villani, and Vredin (2005). That is, with $\psi = \text{vec}(\Psi)$ I assume that the marginal prior is given by

$$\psi \sim N_{pq}\left(\theta_\psi, \Sigma_\psi\right), \tag{12.2}$$

where Σ_ψ is positive definite. YADA allows the user to select any values for θ_ψ and for the diagonal of Σ_ψ. The off-diagonal elements of the prior covariance matrix are assumed to be zero.

Let $\Pi = [\Pi_1 \ \cdots \ \Pi_k]$ be the $p \times pk$ matrix with parameters on lagged x. The prior distributions for these parameters that YADA supports are as follows:

(i) a Minnesota-style prior similar to the one considered by Villani (2007);
(ii) a normal conditional on the covariance matrix of the residuals (see, e.g., Kadiyala and Karlsson, 1997); and
(iii) a diffuse prior.

Below I will address the details about each prior distribution.

First, for the Minnesota-style prior the marginal prior distribution of Π is given by:

$$\text{vec}(\Pi) \sim N_{p^2k}\left(\theta_\pi, \Sigma_\pi\right), \tag{12.3}$$

where the prior mean θ_π need not be unity for the first own lagged parameters and zero for the remaining. In fact, the general setup considers x_t to be stationary with steady state determined by the prior mean of Ψ and d_t.

Let $\theta_\pi = \text{vec}(\mu_\pi)$, where $\mu_\pi = [\mu_{\Pi_1} \ \cdots \ \mu_{\Pi_k}]$ is a $p \times pk$ matrix with the prior mean of Π. The assumption in YADA is that $\mu_{\Pi_l} = 0$ for $l \geq 2$, while μ_{Π_1} is diagonal. The diagonal entries are determined by two hyperparameters, γ_d and γ_l. With $\Pi_{ii,1}$ being the i:th diagonal element of Π_1, the prior mean of this parameter, denoted by μ_{ii,Π_1}, is equal to γ_d if variable i in the x_t vector is regarded as being first differenced (e.g., output growth), and γ_l if variable i is in levels (e.g., the nominal interest rate).

The Minnesota feature of this prior refers to the covariance matrix Σ_π. Let $\Pi_{ij,l}$ denote the element in row (equation) i and column (on variable) j for lag l. The matrix Σ_π is here assumed

to be diagonal with

$$\text{Var}\big(\Pi_{ij,l}\big) = \begin{cases} \dfrac{\lambda_o}{l^{\lambda_h}}, & \text{if } i = j, \\ \dfrac{\lambda_o \lambda_c \Omega_{ii}}{l^{\lambda_h} \Omega_{jj}}, & \text{otherwise.} \end{cases} \tag{12.4}$$

The parameter Ω_{ii} is simply the variance of the residual in equation i and, hence, the ratio $\Omega_{ii} / \Omega_{jj}$ takes into account that variable i and variable j may have different scales.

Formally, this parameterization is inconsistent with the prior being a marginal distribution since it depends on Ω. YADA tackles this in the standard way by replacing the Ω_{ii} parameters with the maximum likelihood estimate. The hyperparameter $\lambda_o > 0$ gives the overall tightness of the prior around the mean, while $0 < \lambda_c < 1$ is the cross-equation tightness hyperparameter. Finally, the hyperparameter $\lambda_h > 0$ measures the harmonic lag decay.

Second, when the prior of Π is no longer marginal but condition on the covariance matrix of the residuals we use the following:

$$\text{vec}(\Pi)|\Omega \sim N_{p^2 k}\big(\theta_\pi, \Omega_\pi \otimes \Omega\big), \tag{12.5}$$

where Ω_π is a positive definite $pk \times pk$ matrix, while θ_π is determined in exactly the same way as for the Minnesota-style prior above. A prior of this generic form is, for instance, examined by Kadiyala and Karlsson (1997), where it is also discussed relative to, e.g., the Minnesota prior. The matrix Ω_π is assumed to be block diagonal in YADA, where block $l = 1, \ldots, k$ (corresponding to Π_l) is given by

$$\Omega_{\pi_l} = \frac{\lambda_o}{l^{\lambda_h}} I_p. \tag{12.6}$$

Hence, the overall tightness as well as the harmonic lag decay hyperparameter enter this prior, while the cross-equation hyperparameter cannot be included. This is the price for using the Kronecker structure of the prior covariance matrix. At the same time, different scales of the variables are now handled by conditioning on Ω instead of using sample information.

Finally, in the case of the diffuse prior we simply assume that the prior density $p(\Pi) = 1$.

The marginal prior distribution for Ω is either assumed to be diffuse or inverse Wishart. Let the marginal prior density of Ω be denoted by $p(\Omega)$. In the former case, we simply make use of the standard formula (see, e.g., Zellner, 1971)

$$p(\Omega) \propto |\Omega|^{-(p+1)/2}. \tag{12.7}$$

When Ω is assumed to be inverse Wishart, denoted by $\Omega \sim IW_p(A, \nu)$, the density of Ω is given by

$$p(\Omega) = \frac{|A|^{\nu/2}}{2^{\nu p/2} \pi^{p(p-1)/4} \Gamma_p(\nu)} |\Omega|^{-(\nu+p+1)/2} \exp\left(-\frac{1}{2} \text{tr}\big[\Omega^{-1} A\big] \right), \tag{12.8}$$

where $\Gamma_b(a) = \prod_{i=1}^{b} \Gamma([a-i+1]/2)$ for positive integers a and b, with $a \geq b$, and $\Gamma(\cdot)$ being the gamma function in (2.3). The hyperparameters of this prior are given by A and ν, where ν is the number of degrees of freedom. The mode of the inverse Wishart is given by $(1/(p+\nu+1))A$, while the mean exist if $\nu \geq p+2$ and is then given by $E[\Omega] = (1/(\nu-p-1))A$. If $\nu = 0$ (and $A = 0$), then the density of Ω is proportional to that in equation (12.7); see, e.g., Zellner (1971).

The hyperparameter A can be selected in two ways in YADA. The first route is to let A equal the maximum likelihood estimate of Ω. This was suggested by, e.g., Villani (2005). The alternative is to let $A = \lambda_A I_p$, where the hyperparameter λ_A gives the joint marginal prior residual variance; see, e.g., Warne (2006). By selecting the degrees of freedom as small as possible (given finite first moments) the impact of the parameterization for A is minimized, i.e., by letting $\nu = p + 2$.[20]

[20] Given that the inverse Wishart prior has been selected for Ω and the normal conditional on Ω prior for Π, it follows by standard distribution theory that the marginal prior of Π is matricvariate t.

Finally, it should be pointed out that the joint prior distribution of (Ψ, Π, Ω) satisfies certain independence conditions. Specifically, Ψ is assumed to be independent of Π and Ω. Under the Minnesota-style prior for Π it is also assumed that Π is a prior independent of Ω.

12.2. *Posterior Mode*

Before we turn to the estimation of the posterior mode we need to introduce some additional notation. Let x be a $p \times T$ matrix with x_t in column t, while ε is constructed in the same way. Similarly, let d be a $q \times T$ matrix with d_t in column t. Furthermore, let D be a $q(k+1) \times T$ matrix where column t is given by $[d_t' \; -d_{t-1}' \; \cdots \; -d_{t-k}']'$. Also, let $y = x - \Psi d$, while Y is a $pk \times T$ matrix where column t is given by $[y_{t-1}' \; \cdots \; y_{t-k}']'$ with $y_t = x_t - \Psi d_t$. Next, let X be a $pk \times T$ matrix where column t is $[x_{t-1}' \; \cdots \; x_{t-k}']'$. From this we can define $z = x - \Pi X$.

Hence, the stacked VAR may be expressed as either:

$$y = \Pi Y + \varepsilon, \tag{12.9}$$

or

$$z = \Theta D + \varepsilon, \tag{12.10}$$

where $\Theta = [\Psi \;\; \Pi_1 \Psi \;\; \cdots \;\; \Pi_k \Psi]$. Applying the vec operator on Θ gives us:

$$\begin{aligned} \operatorname{vec}(\Theta) &= \begin{bmatrix} I_{pq} \\ (I_q \otimes \Pi_1) \\ \vdots \\ (I_q \otimes \Pi_k) \end{bmatrix} \operatorname{vec}(\Psi), \\ &= U \operatorname{vec}(\Psi). \end{aligned} \tag{12.11}$$

The nonlinearity of the VAR model means that an analytical solution for the mode of the joint posterior distribution of (Ψ, Π, Ω) is not available. However, from the first order conditions we can express three systems of equations that the mode must satisfy, and by iterating on these equations it is possible to quickly solve for the posterior mode. Naturally, the choice of prior influences the three systems of equation.

First, the choice of prior for Π and Ω does not have any effect on the equations that Ψ has to satisfy at the mode conditional on Π and Ω. Here we find that

$$\psi = \left[U' \left(DD' \otimes \Omega^{-1} \right) U + \Sigma_\psi^{-1} \right]^{-1} \left(U' \operatorname{vec}\left(\Omega^{-1} z D' \right) + \Sigma_\psi^{-1} \theta_\psi \right). \tag{12.12}$$

Second, in case the Minnesota-style prior is applied to Π, the posterior mode estimate must satisfy the system of equations

$$\operatorname{vec}(\Pi) = \left[\left(YY' \otimes \Omega^{-1} \right) + \Sigma_\pi^{-1} \right]^{-1} \left(\operatorname{vec}\left(\Omega^{-1} y Y' \right) + \Sigma_\pi^{-1} \theta_\pi \right). \tag{12.13}$$

Similarly, when a normal conditional on the residual covariance matrix prior is used for Π, then the posterior mode must satisfy:

$$\Pi = \left(y Y' + \mu_\pi \Omega_\pi^{-1} \right) \left[YY' + \Omega_\pi^{-1} \right]^{-1}. \tag{12.14}$$

The system of equations that Π needs to satisfy when a diffuse prior is used on these parameters is, for instance, obtained by letting $\Omega_\pi^{-1} = 0$ in (12.14), i.e., $\Pi = yY'(YY')^{-1}$.

Third, in case a Minnesota-style prior is used on Π, then the posterior mode of Ω must satisfy:

$$\Omega = \frac{1}{T + p + \nu + 1} \left(\varepsilon \varepsilon' + A \right). \tag{12.15}$$

If the prior on Ω is diffuse, i.e., given by (12.7), we simply set $\nu = 0$ and $A = 0$ in (12.15).

Similarly, when the prior on Π is given by (12.5), then the posterior mode of Ω satisfies

$$\Omega = \frac{1}{T + p(k+1) + \nu + 1} \left(\varepsilon \varepsilon' + A + \left(\Pi - \mu_\pi \right) \Omega_\pi^{-1} \left(\Pi - \mu_\pi \right)' \right). \tag{12.16}$$

If the prior on Ω is diffuse we again let $\nu = 0$ and $A = 0$. Similarly, if the prior on Π is diffuse, we set $k = 0$ and $\Omega_\pi^{-1} = 0$ in (12.16).

12.3. *Gibbs Samplers for a Bayesian VAR*

The posterior samplers used by YADA for drawing from the posterior distribution of the parameters of the Bayesian VAR models that it supports are simple Gibbs samplers; see, e.g., Casella and George (1992), Tierney (1994), or Geweke (1999, 2005). This means that the full conditional posterior distributions are needed for (Ψ, Π, Ω).

The full conditional posterior distribution of Ψ is given by Villani (2007, Proposition 2.1). Let $\mathfrak{D}_T = \{x_{1-k}, \ldots, x_0, x_1 \ldots, x_T, d_{1-k}, \ldots, d_0, d_1 \ldots, d_T\}$. We can now express this distribution as:

$$\psi|\Pi, \Omega, \mathfrak{D}_T \sim N_{pq}\big(\bar{\theta}_\psi, \bar{\Sigma}_\psi\big), \tag{12.17}$$

where $\bar{\Sigma}_\psi^{-1} = U'(DD' \otimes \Omega^{-1})U + \Sigma_\psi^{-1}$ and $\bar{\theta}_\psi = \bar{\Sigma}_\psi(U'\mathrm{vec}(\Omega^{-1} z D') + \Sigma_\psi^{-1}\theta_\psi)$. Notice that the mean of this conditional distribution has the same general construction as the first order condition expression for Ψ in (12.12).

The full conditional posterior distribution of Π when a Minnesota-style prior is used is also given by Villani (2007, Proposition 2.1). Given our notation, this distribution can be expressed as

$$\mathrm{vec}\big(\Pi\big)|\Psi, \Omega, \mathfrak{D}_T \sim N_{p^2k}\big(\bar{\theta}_\pi, \bar{\Sigma}_\pi\big), \tag{12.18}$$

where $\bar{\Sigma}_\pi^{-1} = (YY' \otimes \Omega^{-1}) + \Sigma_\pi^{-1}$, while $\bar{\theta}_\pi = \bar{\Sigma}_\pi(\mathrm{vec}(\Omega^{-1} yY') + \Sigma_\pi^{-1}\theta_\pi)$. In this case the full conditional distribution of Ω is:

$$\Omega|\Psi, \Pi, \mathfrak{D}_T \sim IW_p\big(\varepsilon\varepsilon' + A, T + \nu\big). \tag{12.19}$$

If the prior on Ω is assumed to be diffuse, then we simply let $\nu = 0$ and $A = 0$ in (12.18). This results in the full conditional posterior of Ω in Villani (2007).

The case when the prior distribution of Π is normal conditional on Ω instead implies that the full conditional posterior of the autoregressive parameters is given by:

$$\mathrm{vec}\big(\Pi\big)|\Psi, \Omega, \mathfrak{D}_T \sim N_{p^2k}\big(\mathrm{vec}\big(\bar{\mu}_\pi\big), \big[\bar{\Omega}_\pi \otimes \Omega\big]\big), \tag{12.20}$$

where $\bar{\Omega}_\pi^{-1} = YY' + \Omega_\pi^{-1}$ and $\bar{\mu}_\pi = (yY' + \mu_\pi\Omega_\pi^{-1})\bar{\Omega}_\pi$. For this case we find that the full conditional posterior of Ω is:

$$\Omega|\Psi, \Pi, \mathfrak{D}_T \sim IW_p\big(\varepsilon\varepsilon' + A + \big(\Pi - \mu_\pi\big)\Omega_\pi^{-1}\big(\Pi - \mu_\pi\big)', T + pk + \nu\big). \tag{12.21}$$

If the prior on Ω is assumed to be diffuse, then we simply let $\nu = 0$ and $A = 0$ in (12.21).

Finally, in case a diffuse prior is assumed for Π, then the full conditional distribution of Π is given by (12.20), with $\Omega_\pi^{-1} = 0$. Similarly, the full conditional distribution of Ω is now given by (12.21), with $k = 0$ and $\Omega_\pi^{-1} = 0$. Again, if the prior on Ω is assumed to be diffuse, then we simply let $\nu = 0$ and $A = 0$ in (12.21).

12.4. *Marginal Likelihood*

The marginal likelihood for the Bayesian VAR model can be computed using the ideas in Chib (1995); see, also, Geweke (1999, 2005). That is, we first note that

$$p\big(\mathfrak{D}_T\big) = \frac{p\big(\mathfrak{D}_T|\Psi, \Pi, \Omega\big)p\big(\Psi, \Pi, \Omega\big)}{p\big(\Psi, \Pi, \Omega|\mathfrak{D}_T\big)}, \tag{12.22}$$

by applying Bayes rule. Chib (1995) refers to this expression as the *basic marginal likelihood identity* and it holds for any parameter point (Ψ, Π, Ω) in the support.

The numerator in (12.22) is simply the likelihood times the joint prior density, while the denominator is the joint posterior density. The numerator can be directly evaluated at any valid parameter point, such as the posterior mode. The density of the joint posterior is, however, unknown, but can be estimated.

The obtain such an estimate we first factorize the joint posterior density as follows:

$$p\big(\Psi, \Pi, \Omega|\mathfrak{D}_T\big) = p\big(\Omega|\Psi, \Pi, \mathfrak{D}_T\big)p\big(\Psi|\Pi, \mathfrak{D}_T\big)p\big(\Pi|\mathfrak{D}_T\big). \tag{12.23}$$

The first term on the right hand side of (12.23) is the density of the inverse Wishart and can therefore be evaluated directly at the selected parameter point $(\Psi, \Pi, \Omega) = (\tilde{\Psi}, \tilde{\Pi}, \tilde{\Omega})$; cf. equation (12.19) or (12.21).

Let $(\Psi^{(i)}, \Pi^{(i)}, \Omega^{(i)})$ be N posterior draws using the relevant Gibbs sampler in Section 12.3 for the full conditional posterior. The third term on the right hand side of (12.23) can now be estimated as

$$\hat{p}\big(\tilde{\Pi}|\mathfrak{D}_T\big) = N^{-1}\sum_{i=1}^{N} p\big(\tilde{\Pi}|\Psi^{(i)}, \Omega^{(i)}, \mathfrak{D}_T\big). \tag{12.24}$$

The density on the right hand side of (12.24) is normal and parameterized as shown in equation (12.18) or (12.20).

There remains to estimate the conditional posterior density $p(\Psi|\Pi, \mathfrak{D}_T)$ at the selected parameter point. In this case we cannot, as Chib (1995) explains, use the posterior draws from the Gibbs sampler using the full conditional posteriors above. Instead, we can use Gibbs samplers for (Ψ, Ω) for the fixed value of $\Pi = \tilde{\Pi}$. That is, we draw $\Psi^{(j)}$ from (12.17) with $\Pi = \tilde{\Pi}$. Similarly, we draw $\Omega^{(j)}$ from either (12.19) or from (12.21) with $\Pi = \tilde{\Pi}$. The gives us N posterior draws $(\Psi^{(j)}, \Omega^{(j)})$ that are all based on a fixed value of Π. We can now estimate $p(\Psi|\Pi, \mathfrak{D}_T)$ at $(\tilde{\Psi}, \tilde{\Pi})$ using

$$\hat{p}\big(\tilde{\Psi}|\tilde{\Pi}, \mathfrak{D}_T\big) = N^{-1}\sum_{j=1}^{N} p\big(\tilde{\Psi}|\Omega^{(j)}, \tilde{\Pi}, \mathfrak{D}_T\big). \tag{12.25}$$

The density on the right hand side is normal with parameters given by equation (12.17).

There are, of course, alternative ways of estimating the marginal likelihood $p(\mathfrak{D}_T)$ for the Bayesian VAR model; see, e.g., Geweke (1999). The approach advocated by Chib (1995) is generally viewed as reliable when a parameter point with a high posterior density is used. The posterior mode, discussed in Section 12.2, is one such point, but one may also consider an estimate of the joint posterior mean. YADA always makes use of the posterior mode, thus explaining why the posterior mode must be estimated prior to running the posterior sampler for the Bayesian VAR. Moreover, when the Gibbs sampler based on the full conditional posteriors are applied, then the point of initialization is given by the posterior mode.

The chosen order of factorization influences how the Chib estimator of the marginal likelihood is carried out. Since Π generally has (a lot) more parameters than Ψ or Ω is useful to fix Π first. The additional Gibbs steps for Ψ and Ω can be carried out much more quickly than the more time consuming Π step. The choice between using the full condition posterior for Ψ or Ω is not so important. From a computational perspective it should generally not matter much if we estimate $p(\Psi|\Pi, \mathfrak{D}_T)$ or $p(\Omega|\Pi, \mathfrak{D}_T)$ since the dimensions of Ψ and Ω are generally fairly low.

12.5. *YADA Code*

YADA contains a wide range of functions for the Bayesian VAR analysis. In this section I will limit the discussion to the four main topics above, i.e., the prior, the posterior mode estimation, the Gibbs sampler, and the marginal likelihood calculation.

12.5.1. *Functions for Computing the Prior*

This section concerns two functions, `MinnesotaPrior` and `NormalConditionPrior`. These functions both compute elements needed for the prior covariance matrix of the Π parameters. The first functions is used when the Minnesota-style prior is assumed for these parameters, i.e., when Σ_π in (12.3) is determined as in equation (12.4). Similarly, the second function is applied when the normal condition on Ω prior is assumed. In this case, the matrix Ω_π in (12.5) is determined as in equation (12.6).

12.5.1.1. MinnesotaPrior

The function MinnesotaPrior requires 5 inputs: OverallTightness (λ_o), CrossEqTightness (λ_c), HarmonicLagDecay (λ_h), OmegaVec, and k. While the first three inputs are hyperparameters, the fourth is a vector with the diagonal elements of Ω, i.e., with the residual variances. YADA always uses the maximum likelihood estimate of Ω to generate these residual variances. Finally, the fifth input is the lag order of the Bayesian VAR. The function provides the $p^2k \times p^2k$ matrix SigmaPi (Σ_π) as output.

12.5.1.2. NormalConditionPrior

The function NormalConditionPrior takes 4 inputs: OverallTightness, HarmonicLagDecay, p, and k. The first two are the same hyperparameters as the MinnesotaPrior function uses, while the third input is the number of endogenous variables, and the fourth the lag order. As output the function provides the $pk \times pk$ matrix OmegaPi (Ω_π).

12.5.2. *Functions for Estimating the Mode of the Joint Posterior*

The function BVARPosteriorModeEstimation is used to estimate the posterior mode of the Bayesian VAR parameters. It handles all the types of priors discussed in Section 12.1. The main inputs for this function are the structures DSGEModel and CurrINI; see Section 8.2.

From the perspective of analysing a Bayesian VAR model, the DSGEModel structure contains information about the type of prior to use for VAR parameters, their hyperparameters, the lag order, as well as which endogenous and exogenous variables to use, for which sample, the data, the maximum number of iterations to consider, and the tolerance value of the convergence criterion. This information allows the function to compute the maximum likelihood estimates of Ψ, Π, and Ω and fully set up the prior. The maximum likelihood estimates are used as initial values for the posterior mode estimation algorithm. The maximum likelihood estimate of Ω is adjusted to take the prior into account. For example, if a diffuse prior is used for Ω and for Π, then the maximum likelihood estimate of Ω is multiplied by $T/(T+p+1)$. Similarly, if the inverse Wishart prior in (12.8) is assumed with $\nu \geq p+2$, then the maximum likelihood estimate is multiplied by T, A is added to this, and everything is divided by $T+p+\nu+1$.

As discussed in Section 12.2, it is not possible to solve for the posterior mode analytically. Instead, it is possible to iterate on the first order conditions until a set of values that satisfy these conditions can be found. The posterior mode estimation routine in YADA first evaluates the log posterior at the initial values. For each iteration i YADA computes:

- $\Pi^{(i)}$ given $\Psi^{(i-1)}$ and $\Omega^{(i-1)}$ as shown in equation (12.13) or (12.14);
- $\Psi^{(i)}$ given $\Pi^{(i)}$ and $\Omega^{(i-1)}$ as shown in equation (12.12); and
- $\Omega^{(i)}$ given $\Psi^{(i)}$ and $\Pi^{(i)}$ as shown in equation (12.15) or (12.16).

With a new set of parameter values, the log posterior is recalculated and if then absolute change is not sufficiently small, the algorithm computes iteration $i+1$. Otherwise it exits.

YADA has three functions for calculating the log posterior. First, if a diffuse prior on Π is assumed, then BVARLogPosteriorDiffuse is called. Second, if the prior on Π is of the Minnesota-style, then the function BVARLogPosteriorMinnesota is used. Finally, the function BVARLogPosteriorNormalCond is considered when a normal conditional on the residual covariance matrix prior on Π is assumed.

Furthermore, YADA has 5 functions for dealing with the computations in (12.12)–(12.16). These functions are: BVARPsiMean (equation 12.12) for Ψ, BVARPiMeanMinnesota (equation 12.13) or BVARPiMeanNormalCond (equation 12.14) for Π, and BVAROmegaMinnesota (equation 12.15) or BVAROmegaNormal (equation 12.16) for Ω. The functions are also used by the Gibbs sampling routine for the Bayesian VAR; cf. Section 12.5.3.

12.5.2.1. BVARLogPosteriorDiffuse

The function BVARLogPosteriorDiffuse needs 11 inputs. First of all it requires the parameter values Omega (Ω), Pi (Π), and Psi (Ψ). Next, the hyperparameters of the prior of the residual covariance matrix are needed as Amatrix (A) and qDF (ν), and the hyperparameters of the

steady state prior `thetaPsi` (θ_ψ) and `SigmaPsi` (Σ_ψ). Finally, the function needs information about the endogenous and exogenous variables in terms of the matrices `x` (x), `X` (X), `d` (d), and `dLag`. The latter matrix is given by d_{-1} in $D = [d'\ d'_{-1}]'$, i.e., the final qk rows of D.

The function provides the output scalar `logPost` which is the sum of the log-likelihood, the log-prior of Ψ and the log-prior of Ω. All proper log-densities include their integration constants. In case the Ω prior is diffuse (improper), it simply uses the log of the right hand side of equation (12.7).

12.5.2.2. BVARLogPosteriorMinnesota

The function `BVARLogPosteriorMinnesota` requires 13 inputs. All the inputs that the function `BVARLogPosteriorDiffuse` accepts are included as well as two additional inputs. These extra inputs appear as argument eight and nine and are called `thetaPi` (θ_π) and `SigmaPi` (Σ_π).

As output the function provides the scalar `logPost` which is the sum of the log-likelihood, the log-prior of Ψ, the log-prior of Π, and the log-prior of Ω.

12.5.2.3. BVARLogPosteriorNormalCond

The function `BVARLogPosteriorNormalCond` requires 13 inputs. All the inputs that the function `BVARLogPosteriorDiffuse` accepts are included as well as two additional inputs. These extra inputs appear as argument eight and nine and are called `muPi` (μ_π) and `OmegaPi` (Ω_π).

As output the function provides the scalar `logPost` which is the sum of the log-likelihood, the log-prior of Ψ, the log-prior of Π, and the log-prior of Ω.

12.5.2.4. BVARPsiMean

The function `BVARPsiMean` is used to compute the value of $\bar{\theta}_\psi$ is equation (12.17). Optionally, it can also calculate the $pq \times pq$ matrix $\bar{\Sigma}_\psi$ as well. A total of 8 inputs are needed by the function. These are given by: `Omega`, `Pi`, `thetaPsi`, `invSigmaPsi`, `x`, `X`, `d`, and `dLag`. Apart from `invSigmaPsi` all these inputs are discussed for the `BVARLogPosteriorDiffuse` function. The matrix `invSigmaPsi` is simply the inverse of Σ_ψ.

The required output from the function is `ThetaBarPsi`, a $p \times q$ matrix. If we apply the vec operator on this matrix we obtain $\bar{\theta}_\psi$. The optional output is called `SigmaBarPsi`. While only the required output is needed by the posterior mode estimation routine, the optional output is needed by the Gibbs sampler for drawing from the posterior distribution.

12.5.2.5. BVARPiMeanMinnesota

The function `BVARPiMeanMinnesota` accepts 8 inputs: `Omega`, `Psi`, `thetaPi`, `invOmegaPi`, `x`, `X`, `d`, and `dLag`. Apart from `invOmegaPi` all these inputs are described above for the log posterior function `BVARLogPosteriorMinnesota`. The input `invOmegaPi` is simply the inverse of Σ_π.

The output `ThetaBarPi` is required and is a $p \times pk$ matrix. The vector $\bar{\theta}_\pi$ in (12.18) is obtained by applying the vec operator to this output. Optionally, the matrix `SigmaBarPi` is provided. This output is equal to $\bar{\Sigma}_\pi$ in the same equation.

12.5.2.6. BVARPiMeanNormalCond

The function `BVARPiMeanNormalCond` needs 7 inputs: `Psi`, `muPi`, `invOmegaPi`, `x`, `X`, `d`, and `dLag`. All these inputs apart from `invOmegaPi` are discussed above for the log posterior function `BVARLogPosteriorNormalCond`. The input `invOmegaPi` is, of course, the inverse of Ω_π.

The output `muBarPi` is required and is the $p \times pk$ matrix $\bar{\mu}_\pi$ in (12.20). Optionally, the matrix `OmegaBarPi` is provided. This output is equal to $\bar{\Omega}_\pi$ in the same equation.

12.5.2.7. BVAROmegaMinnesota

The function `BVAROmegaMinnesota` needs 8 inputs: `Pi`, `Psi`, `A`, `qDF`, `x`, `X`, `d`, and `dLag`. These inputs are also required by `BVARLogPosteriorMinnesota`. As output the function provides `Omega`, the $p \times p$ matrix in equation (12.15).

12.5.2.8. BVAROmegaNormal

The function BVAROmegaNormal accepts 10 inputs: Pi, Psi, A, qDF, muPi, invOmegaPi, x, X, d, and dLag. These inputs are discussed above; see the BVARLogPosteriorNormalCond and the BVARPiMeanNormalCond functions. As output the function provides Omega, the $p \times p$ matrix in equation (12.16).

12.5.3. *Gibbs Sampling*

The function BVARPosteriorSampling controls the events regarding the posterior sampling algorithm. The function takes exactly the same inputs as the posterior mode estimation function BVARPosteriorModeEstimation. The Bayesian VAR estimation routines follow the same type of logic as the DSGE model estimation routines. This means that you have to run the posterior mode estimation before the posterior sampling function can be run. The reason is simply that the posterior sampling routine for the Bayesian VAR uses the posterior mode estimates of the parameters to initialize the Gibbs sampler.

As in the case of posterior sampling for the DSGE model, the sampling function for the Bayesian VAR model reads a number of variable entries from the posterior mode estimation output file. These data are read from file to ensure that exactly the same data is used for posterior sampling as was used for posterior mode estimation. Hence, if you have changed some hyperparameter after running the posterior mode estimation part, the new value will not be used by YADA. Similarly, changes to the sample will be ignored as well as any other changes to the data.

The posterior sampling function can look for sampling data that you have already generated and can load this data. These features operate in exactly the same way for the posterior sampling function for the DSGE and the Bayesian VAR model.

The precise Gibbs sampler for the Bayesian VAR depends on the prior you are using for the Ψ, Π, and Ω parameters. The posterior mode estimates are always used as initial values for the sampler. To generate draw number i of the parameters YADA first draws $\Omega^{(i)}$ conditional on $\Psi^{(i-1)}$ and $\Pi^{(i-1)}$ with the function InvWishartRndFcn. Since the marginal likelihood is also estimated as described in Section 12.4 YADA also draws a value for Ω conditional on Π fixed at the posterior mode and a theoretically consistent previous value of Ψ, i.e., one that is also conditioned on Π at the posterior mode. Next, YADA draws $\Pi^{(i)}$ conditional on $\Psi^{(i-1)}$ and $\Omega^{(i)}$. Here it utilizes the function MultiNormalRndFcn. To finish the i:th draw, a value of $\Psi^{(i)}$ conditional on $\Pi^{(i)}$ and $\Omega^{(i)}$ is obtained. Again, YADA makes use of the function MultiNormalRndFcn.

As in the case of Ω YADA also draws a value of Ψ conditional on Π fixed at the posterior mode and the draw obtained for Ω when Π is fixed at the mode. This value of Ψ is used for the next draw of Ω conditional on Π fixed at the posterior mode. In this fashion YADA generates two sets of Gibbs sampler draws. The first full set $(\Psi^{(i)}, \Pi^{(i)}, \Omega^{(i)})$ are draws from the joint posterior and are also used to estimate the density $p(\tilde{\Pi}|\mathfrak{Y}_T)$ in equation (12.24). The second partial set $(\Psi^{(j)}, \Omega^{(j)})$ is only used to estimate the conditional density $p(\tilde{\Psi}|\tilde{\Pi}, \mathfrak{Y}_T)$ in equation (12.25).

12.5.3.1. InvWishartRndFcn

The function InvWishartRndFcn requires two inputs to generate a draw from the inverse Wishart distribution. These inputs are A and df, representing the location parameter and the degrees of freedom parameter respectively. As output the function provides Omega.

12.5.3.2. MultiNormalRndFcn

The function MultiNormalRndFcn generates a desired number of draws from the multivariate normal distribution. As input the function needs mu, Sigma, and NumDraws. These inputs provide the mean, the covariance matrix and the number of desired draws, respectively. The last input is optional and defaults to 1.

As output the function gives z, a matrix with as the same number of rows as the dimension of the mean and number of columns gives by NumDraws.

12.5.4. *Marginal Likelihood of the Bayesian VAR*

Estimation of the marginal likelihood is handled by the function `MargLikeChib` in YADA. Before discussing this function in more detail it is worthwhile to keep Bartlett's or (Lindley's) paradox in mind; see Bartlett (1957) and Lindley (1957). That is, as a rule of thumb we should only compare the marginal likelihood value across two models if the prior is proper in the dimensions where they differ. By proper we mean that the prior density should integrate to unity (a finite constant) over these parameters. For instance, if the Minnesota-style prior is assumed for Π, then we can compare the marginal likelihood for models that differ in terms of the lag order given that the same sample dates and variables are covered by x_t. If, instead, the diffuse prior $p(\Pi) \propto 1$ is used for these parameters, then the marginal likelihoods should not be compared in this dimension. The paradox here states that the model with fewer lags will have a greater marginal likelihood value regardless of the information in the data.

12.5.4.1. `MargLikeChib`

The function `MargLikeChib` computes the log marginal likelihood using Chib's marginal likelihood identity. As input the function requires 9 inputs (and accepts a 10th). First of all it takes two vectors with `NumIter` elements of values of the log densities $p(\tilde{\Pi}|\Psi^{(i)}, \Omega^{(i)}, \mathfrak{D}_T)$ (`LogPiDensity`) and $p(\tilde{\Psi}|\tilde{\Pi}, \Omega^{(j)}, \mathfrak{D}_T)$ (`LogPsiDensity`). Next, the scalars `LogPosterior` and `LogOmegaDensity` are accepted, where the first measures the sum of the log-likelihood and the log-prior, while the second measures the log of $p(\tilde{\Omega}|\tilde{\Psi}, \tilde{\Pi}, \mathfrak{D}_T)$. The fifth input is `q`, giving the number of exogenous variables.

The following and sixth input is the integer `NumBurnin` which provides the number of burn-in draws to be removed from the top of the log density vector. Next, `MargLikeChib` accepts the boolean input `ComputeSequential`. The function will compute sequential estimates of the log marginal likelihood is this input is unity, and only the final estimate if it is zero. The first of the remaining inputs is `SequentialStartIterationValue`. This integer determines after how many draws the first sequential estimate shall be performed. Similarly, the last required input `SequentialStepLengthValue` determines how many draws to use as increment.

The function provides the matrix `LogMargs` as output. The number of rows of this matrix is equal to the number of sequential estimates. The first column contains the number of draws used for a particular estimate, the second column the estimated log marginal likelihood, and the third column the numerical standard error of the estimate based on the Newey and West (1987) correction for autocorrelation.

13. A Bayesian Approach to Forecasting

13.1. *Unconditional Forecasting*

13.1.1. *The State-Space Model*

To analyse the predictions of a set of future observations $y_{T+1}, \ldots, y_{T+h}$ conditional on the data that can be observed at time T and a path for future values of the exogenous variables, we need to determine its predictive distribution. Letting this distribution be denoted by $p(y_{T+1}, \ldots, y_{T+h}|x_{T+1}, \ldots, x_{T+h}, \mathbf{y}_T)$, we note that it can be described in terms of the distribution for the future observations conditional on the data and the parameters and the full posterior distribution. That is,

$$p\left(y_{T+1}, \ldots, y_{T+h}|x_{T+1}, \ldots, x_{T+h}, \mathbf{y}_T\right) = \int_{\theta\in\Theta} p\left(y_{T+1}, \ldots, y_{T+h}|x_{T+1}, \ldots, x_{T+h}, \mathbf{y}_T; \theta\right) \times p\left(\theta|\mathbf{y}_T\right)d\theta, \tag{13.1}$$

where Θ is the support of θ. The problem of numerically determining the predictive distribution is examined in the context of a cointegrated VAR by Villani (2001). Although the full predictive distribution cannot be obtained analytically, a procedure suggested by Thompson and Miller (1986) may be applied. This procedure may be used both for the state-space representation of the DSGE model and for the Bayesian VAR model.

The procedure is based on a double simulation scheme, where S draws of θ from its full posterior are first obtained. In the second stage, prediction paths are simulated for $x_{T+1}, \ldots, x_{T+h}$ conditional on the data and θ. From the conditional predictive distribution we have that

$$p\left(y_{T+1}, \ldots, y_{T+h} | x_{T+1}, \ldots, x_{T+h}, \mathbf{y}_T; \theta\right) = \prod_{i=1}^{h} p\left(y_{T+i} | x_{T+i}, \mathbf{y}_{T+i-1}; \theta\right). \tag{13.2}$$

The right hand side of equation (13.2) is obtained by the usual conditioning formula and by noting that y_{T+i} conditional on $x_{T+i}, \mathbf{y}_{T+i-1}$ and the parameters is independent of x_{t+j} for all $j > i$. Moreover, the density for this conditional expression is for the state-space model given by a multivariate normal with mean $y_{T+i|T+i-1}$ and covariance $(H' P_{T+i|T+i-1} H + R)$.

The approach suggested by Thompson and Miller (1986) may be implemented for the state-space model as follows. For a given draw of θ from the posterior distribution, the DSGE model is solved and the matrices A, H, R, F, and Q are calculated. A value for y_{T+1} is now generated by drawing from the normal distribution with mean $y_{T+1|T}$ and covariance matrix $(H' P_{T+1|T} H + R)$ using the expressions in Section 3.

For period $T + 2$ we treat the draw y_{T+1} as given. This allows us to compute $y_{T+2|T+1}$ and $(H' P_{T+2|T+1} H + R)$ and draw a value of y_{T+2} from the multivariate normal using these values as the mean and the covariance. Proceeding in the same way until we have drawn y_{T+h} we have obtained one possible future path for $y_{T+1}, \ldots, y_{T+h}$ conditional on $x_{T+1}, \ldots, x_{T+h}$, $\mathbf{y}_T$ and the given draw θ from the posterior distribution.

We may now continue and draw a total of P paths for $y_{T+1}, \ldots, y_{T+h}$ conditional on this information. Once all these paths have been drawn, we pick a new value of θ from its posterior and recalculate everything until a total of PS draws from the density in (13.2) have been drawn.

Alternatively, and as suggested by Adolfson, Lindé, and Villani (2007b), we can directly utilize the state-space form. Specifically, for a given draw of θ from its posterior distribution, the period T state vector can be drawn from $N(\xi_{T|T}, P_{T|T})$, where $\xi_{T|T}$ and $P_{T|T}$ are obtained from the final step of the Kalman filter computations; cf. Section 3. Next, a sequence of future states $\xi_{T+1}, \ldots, \xi_{T+h}$ can be simulated from the state equation (3.2) given draws of the state shocks $v_{T+1}, \ldots, v_{T+h}$. The latter are drawn from the normal distribution with mean zero and covariance matrix $Q = B_0 B_0'$. Next, vectors of measurement errors $w_{T+1}, \ldots, w_{T+h}$ are drawn from a normal distribution with mean zero and covariance matrix R. Adding the sequence of state variables and the measurement errors a path for $y_{T+1}, \ldots, y_{T+h}$ is obtained via the measurement equation (3.1). For the given value of θ, we can now generate P paths of the observed variables and by repeating this for S draws from the posterior distribution of θ and total of PS paths from the predictive density of $y_{T+1}, \ldots, y_{T+h}$ may be obtained.

The Adolfson, Lindé, and Villani (2007b) approach is faster than the first approach since the underlying computations are more direct. For this reason, the Adolfson, Lindé, and Villani approach has been implemented in YADA. Moreover, this procedure highlights the fact that the uncertainty in the forecasts stems from four sources: parameter uncertainty (θ), uncertainty about the current state (ξ_T), uncertainty about future shocks (v), and measurement errors (w).

For instance, the forecast uncertainty for y_{T+i} can be decomposed as follows:

$$Cov\left(y_{T+i} | \mathbf{y}_T\right) = E_T\left[Cov\left(y_{T+i} | \mathbf{y}_T; \theta\right)\right] + Cov_T\left[E\left(y_{T+i} | \mathbf{y}_T; \theta\right)\right], \tag{13.3}$$

where E_T and Cov_T denotes the expectation and covariance with respect to the posterior of θ at time T and where, for notational simplicity, the sequence of exogenous variables $x_{T+1}, \ldots, x_{T+h}$ have been suppressed from the expressions. Adolfson, Lindé, and Villani (2007b) show that the first term on the right hand side of (13.3) is given by

$$E_T\left[Cov\left(y_{T+i} | \mathbf{y}_T; \theta\right)\right] = E_T\left[H' F^i P_{T|T} \left(F^i\right)' H\right] + E_T\left[H' \left(\sum_{j=1}^{i} F^{j-1} Q \left(F^{j-1}\right)'\right) H\right] + E_T\left[R\right],$$

providing the uncertainties regarding the current state, the future shocks, and the measurement errors. Similarly, for the second term in (13.3) we have that

$$Cov_T\left[E\left(y_{T+i} | \mathbf{y}_T; \theta\right)\right] = Cov_T\left[A' x_{T+i} + H' F^i \xi_{T|T}\right],$$

which thus reflects the influence of parameter uncertainty on forecast uncertainty.

13.1.2. *The VAR Model*

The Thompson and Miller (1986) procedure may also be applied to the Bayesian VAR in Section 12. In this case we let $\theta = (\Psi, \Pi, \Omega)$ where the draws are obtained using the Gibbs samplers discussed in Section 12.3. For a given draw θ from its posterior distribution we may first draw residuals $\varepsilon_{T+1}, \ldots, \varepsilon_{T+h}$ from a normal distribution with mean zero and covariance matrix Ω. Next, we simulate the $x_{T+1}, \ldots, x_{T+h}$ by feeding the residual draws into the VAR system in (12.1). Repeating this P times for the given θ gives us P sample paths conditional on θ. By using S draws of θ from its posterior we end up with PS paths of $x_{T+1}, \ldots, x_{T+h}$ from its predictive density.

For the Bayesian VAR we may decompose the prediction uncertainty into two components, the error uncertainty and the parameter uncertainty. That is,

$$Cov\left(x_{T+i}|\mathfrak{D}_T\right) = E_T\left[Cov\left(x_{T+i}|\mathfrak{D}_T;\theta\right)\right] + Cov_T\left[E\left(x_{T+i}|\mathfrak{D}_T;\theta\right)\right], \tag{13.4}$$

where the deterministic process $d_{T+1}, \ldots, d_{T+h}$ has been suppressed from the expressions to simplify notation. The first term on the right hand side measures the error uncertainty, while the second measures parameter uncertainty. To parameterize these two terms, we first rewrite the VAR model in (12.1) in first order form:

$$Y_{T+i} = BY_{T+i-1} + J_k\varepsilon_{T+i}, \quad i = 1, \ldots, h, \tag{13.5}$$

where J_k is a $pk \times p$ matrix with I_p on top and a zero matrix below. This means that $y_{T+i} = J_k'Y_{T+i}$. Furthermore, the $pk \times pk$ matrix B is given by

$$B = \begin{bmatrix} \Pi_1 & \cdots & \Pi_{k-1} & \Pi_k \\ I_p & & 0 & 0 \\ & \ddots & & \\ 0 & & I_p & 0 \end{bmatrix}. \tag{13.6}$$

Using these well known expressions we first find that the error uncertainty term is:

$$E_T\left[Cov\left(x_{T+i}|\mathfrak{D}_T;\theta\right)\right] = E_T\left[\sum_{j=0}^{i-1} J_k'B^jJ_k\Omega J_k'\left(B^j\right)'J_k\right],$$

while the parameter uncertainty term is given by:

$$Cov_T\left[E\left(x_{T+i}|\mathfrak{D}_T;\theta\right)\right] = Cov_T\left[\Psi d_{T+i} + J_k'B^iY_T\right].$$

13.2. *Conditional Forecasting with the State-Space Model*

Conditional forecasting concerns forecasts of endogenous variables conditional on a certain path and length of path for some other endogenous variables; see, e.g., Waggoner and Zha (1999). In this section I will discuss conditional forecasting as it has been implemented in YADA for a DSGE model. Specifically, it is assumed that the conditioning information satisfies *hard conditions*, i.e., a particular path, rather than *soft conditions* (a range for the path).

Let K_1 and K_2 be known $n \times q_m$ matrices with $n > q_m$ such that $\text{rank}(K_1) = q_m$. Furthermore, consider the following relation:

$$z_{T+i} = K_1'y_{T+i} + \sum_{j=1}^{i-1} K_2'y_{T+i-j} + u_T, \quad i = 1, \ldots, g. \tag{13.7}$$

The specification in equation (13.7) is general enough to satisfy our purposes. In the special case where $K_2 = 0$ and $u_T = 0$ the vector z_{T+i} is determined directly from y_{T+i}, e.g., one particular observed variable. Although such a specification covers many interesting cases it does not allow us to handle the case when y includes the real exchange rate and the first differences of the domestic and foreign prices, but where z is the nominal exchange rate. Let p_t and p_t^* denote the domestic and foreign prices, respectively, while s_t denotes the nominal exchange

rate. We may then let K_1 be defined such that $K_1'y_{T+i} = (s_{T+i} + p^*_{T+i} - p_{T+i}) + \Delta p_{T+i} - \Delta p^*_{T+i}$, whereas $K_2'y_{T+i-j} = \Delta p_{T+i-j} - \Delta p^*_{T+i-j}$ and $u_T = p_T - p^*_T$.

To keep the values in z_{T+i} fixed over the given horizon, the method we consider requires that a subset of the economic shocks are adjusted to take on certain values. The selection of shocks is defined by the user while the values are calculated by taking equation (13.7) into account. The selection of economic shocks is determined by the $q \times q_m$ matrix M where $q > q_m$ and $\text{rank}(M) = q_m$. Let $M_\perp$ be the $q \times (q - q_m)$ orthogonal matrix, i.e., $M_\perp'M = 0$. It now follows that $N = [M_\perp \; M]$ is a full rank $q \times q$ matrix while

$$N'\eta_t = \begin{bmatrix} M_\perp' \\ M' \end{bmatrix} \eta_t = \begin{bmatrix} \eta_t^{(q-q_m)} \\ \eta_t^{(q_m)} \end{bmatrix}. \tag{13.8}$$

The shocks $\eta_t^{(q_m)}$ will be adjusted over the time interval $t = T+1, \ldots, T+g$ to ensure that (13.7) is met for all forecast paths of the observed variables over this time interval.

Let $\bar{M} = M(M'M)^{-1}$ while $\bar{M}_\perp = M_\perp(M_\perp'M_\perp)^{-1}$.[21] We can now express the state equation (3.2) as

$$\xi_t = F\xi_{t-1} + B_0\bar{M}_\perp\eta_t^{(q-q_m)} + B_0\bar{M}\eta_t^{(q_m)}. \tag{13.9}$$

Turning first to period $T+1$ we know that if we substitute for y_{T+1} using the measurement equation (3.1) and the rewritten state equation (13.9), the conditioning on the vector z_{T+1} in (13.7) implies that:

$$z_{T+1} = K_1'A'x_{T+1} + K_1'w_{T+1} + K_1'H'F\xi_T + K_1'H'B_0\bar{M}_\perp\eta_{T+1}^{(q-q_m)} + K_1'H'B_0\bar{M}\eta_{T+1}^{(q_m)} + u_T. \tag{13.10}$$

Provided that the $q_m \times q_m$ matrix $K_1'H'B_0\bar{M}$ has full rank, the economic shocks $\eta_{T+1}^{(q_m)}$ can be uniquely specified such that a fixed value for z_{T+1} is obtained.

With $\eta_{T+1}^{(q_m)}$ being computed such that (13.10) holds, it follows from the state equation (13.9) that the state vector at $T+1$ conditional on the path for z_{T+i} is determined by

$$\xi_{T+1}^{(q_m)} = F\xi_T + B_0\bar{M}_\perp\eta_{T+1}^{(q-q_m)} + B_0\bar{M}\eta_{T+1}^{(q_m)},$$

while the measurement equation implies that y_{T+1} is given by:

$$y_{T+1}^{(q_m)} = A'x_{T+1} + H'\xi_{T+1}^{(q_m)} + w_{T+1}.$$

We may now continue with period $T+2$ where the shocks $\eta_{T+2}^{(q_m)}$ can, for given parameter values, be determined from z_{T+2}, x_{T+2}, w_{T+2}, $\xi_{T+1}^{(q_m)}$, $\eta_{T+2}^{(q-q_m)}$, $y_{T+1}^{(q_m)}$, and u_T. In fact, it is now straightforward to show that the values for the economic shocks which guarantee that the conditioning path $z_{T+1}, \ldots, z_{T+g}$ is always met are:

$$\begin{aligned} \eta_{T+i}^{(q_m)} = \left(K_1'H'B_0\bar{M}\right)^{-1} \Bigg[& z_{T+i} - K_1'A'x_{T+i} - K_1'w_{T+i} - K_1'H'F\xi_{T+i-1}^{(q_m)} \\ & - K_1'H'B_0\bar{M}_\perp\eta_{T+i}^{(q-q_m)} - K_2'\sum_{j=1}^{i-1} y_{T+i-j}^{(q_m)} - u_T \Bigg], \quad i = 1, \ldots, g, \end{aligned} \tag{13.11}$$

while the states and the observed variables evolve according to:

$$\xi_{T+i}^{(q_m)} = F\xi_{T+i-1}^{(q_m)} + B_0\bar{M}_\perp\eta_{T+i}^{(q-q_m)} + B_0\bar{M}\eta_{T+i}^{(q_m)}, \quad i = 1, \ldots, g, \tag{13.12}$$

and

$$y_{T+i}^{(q_m)} = A'x_{T+i} + H'\xi_{T+i}^{(q_m)} + w_{T+i}, \quad i = 1, \ldots, g, \tag{13.13}$$

while $\xi_T^{(q_m)} = \xi_T$.

For $i > g$ there are not any direct restrictions on the possible paths for the observed variables other than that the state vector at $T+g$ needs to be taken into account.

[21] In many situations we will have that $\bar{M} = M$ since M is typically a 0-1 matrix with only one unit element per column. Similarly, we can always select $M_\perp$ such that $\bar{M}_\perp = M_\perp$.

The procedure described here makes it straightforward to calculate conditional predictive distributions. For a given draw of θ from its posterior distribution, the period T state vector ξ_T is drawn from $N(\xi^{(z)}_{T|T+g}, P^{(z)}_{T|T+g})$. YADA sets $\xi^{(z)}_{T|T+g} = \xi_{T|T}$ and $P^{(z)}_{T|T+g} = P_{T|T}$ and, hence, ignores the conditioning assumptions.

Next, the economic shocks $\eta^{(q-q_m)}_{T+i}$ are drawn from $N(0, M'_\perp M_\perp)$ and w_{T+i} from $N(0, R)$. Given the conditioning information the economic shocks $\eta^{(q_m)}_{T+i}$ are calculated from (13.11), the state vector from (13.12), and the observed variables from (13.13) in a sequential manner until $i = g+1$, when the economic shocks $\eta^{(q_m)}_{T+i}$ are drawn from $N(0, M'M)$ until $i = h$. This provides one path for $y_{T+1}, \ldots, y_{T+h}$. Given the value of θ we may next repeat this procedure yielding P paths. By considering S draws of θ we can calculate a total of PS sample paths for the observed variables that take the conditioning into account.

13.3. *Modesty Statistics for the State-Space Model*

Conditional forecasting experiments may be subject to the well known Lucas (1976) critique. Leeper and Zha (2003) introduced the concept of *modest* policy interventions along with a simple metric for evaluating how unusual a conditional forecast is relative to the unconditional forecast. Their idea has been further developed by Adolfson, Laséen, Lindé, and Villani (2005). Below I shall present three modesty statistics, two univariate and one multivariate.

The general idea behind the modesty statistics is to compare the conditional and the unconditional forecast. The forecasts are subject to uncertainty concerning which shocks will hit the economy during the prediction period. For the conditional forecasts some shocks have to take on certain values over the conditioning period to ensure that forecasts are consistent with the conditioning information. If these restricted shocks behave as if they are drawn from their distributions, then the conditioning information is regarded as modest. But if the behavior of the shocks over the conditioning period is different from the assumed, then the agents in the economy may be able to detect this change. In this case, the conditioning information need no longer be modest and might even be subject to the famous Lucas (1976) critique.

Within the context of the state-space representation of the DSGE model, the univariate statistic suggested by Leeper and Zha (2003) is based on setting $\eta^{(q-q_m)}_{T+i} = 0$ and $w_{T+i} = 0$ for $i = 1, \ldots, g$ in equations (13.11)–(13.13). The alternative univariate statistic suggested by Adolfson et al. (2005) does not force these shocks to be zero over the conditioning horizon.

For both approaches the difference between the conditional and the unconditional forecasts at $T+g$ for a given θ is:

$$\begin{aligned}\Phi_{T,g}(\bar{\eta}) &= y_{T+g}(\bar{\eta};\theta) - E\left[y_{T+g}|\underset{\sim}{Y}_T;\theta\right] \\ &= H'F^g\left(\xi_T - \xi_{T|T}\right) + H'\sum_{j=0}^{g-1} F^j B_0\left(\bar{M}\eta^{(q_m)}_{T+g-j} + \bar{M}_\perp \eta^{(q-q_m)}_{T+g-j}\right) + w_{T+g},\end{aligned} \tag{13.14}$$

where $\bar{\eta} = \{\eta^{(q_m)}_t, \eta^{(q-q_m)}_t\}_{t=T+1}^{T+g}$. Under the Leeper and Zha approach the measurement errors and the other shocks $\{\eta^{(q-q_m)}_t\}_{t=T+1}^{T+g}$ are set to zero, while under the Adolfson et al. approach these shocks and errors are drawn from their distributions.

For the latter approach we have that the forecast error variance at $T+g$ is

$$\Omega_{T+g} = H'P_{T+g|T}H + R, \tag{13.15}$$

where as already shown in equation (3.21)

$$P_{T+i|T} = FP_{T+i-1|T}F' + B_0B_0', \quad i = 1, \ldots, g.$$

Under the hypothesis that ξ_T is can be observed at T, the matrix $P_{T|T} = 0$. This assumption is used by Adolfson et al. (2005) and is also imposed in YADA.

The assumption that the state vector in T can be observed at T in the modesty analysis is imposed to make sure that it is consistent with the assumptions underlying the DSGE model. That is, the agents know the structure of the model, all parameters, and all past and present

shocks. Hence, there cannot be any state uncertainty when evaluating the current state. This also has an implication for equation (13.14) where we set $\xi_T = \xi_{T|T}$.

The multivariate modesty statistic can now be defined as:

$$\mathcal{M}_{T,g}(\bar{\eta}) = \Phi_{T,g}(\bar{\eta})'\Omega_{T+g}^{-1}\Phi_{T,g}(\bar{\eta}). \tag{13.16}$$

Under the hypothesis that the conditioning shocks are modest, i.e., $\{\eta_t^{(q_m)}\}_{t=T+1}^{T+g}$ can be viewed as being drawn from a multivariate standard normal distribution, this statistic is $\chi^2(n)$. Instead of using the chi-square as a reference distribution for the multivariate modesty statistic, one may calculate the statistic in (13.16) using the shocks $\eta = \{\eta_t\}_{t=T+1}^{T+g}$ in (13.16) that are drawn from the $N_q(0, I_q)$ distribution and thereafter compute the tail probability $\Pr[\mathcal{M}_{T,g}(\eta) \geq \mathcal{M}_{T,g}(\bar{\eta})]$ to determine if the conditioning information is modest.

One univariate statistic suggested by Adolfson et al. is the following

$$\mathcal{M}_{T,g}^{(i)}(\bar{\eta}) = \frac{\Phi_{T,g}^{(i)}(\bar{\eta})}{\sqrt{\Omega_{T+g}^{(i,i)}}}, \quad i = 1, \ldots, n, \tag{13.17}$$

where $\Phi_{T,g}(\bar{\eta})$ is calculated with the other shocks and the measurement errors drawn from a normal distribution. This statistic has a standard normal distribution under the assumption of modest conditioning shocks.

For the alternative Leeper-Zha related statistic we let $\Phi_{T,g}(\bar{\eta})$ be computed for zero measurement errors and other shocks, while

$$\begin{aligned}\Omega_{T+g} &= H'P_{T+g|T}H,\\ P_{T+i|T} &= FP_{T+i-1|T}F' + B_0\bar{M}M'B_0'.\end{aligned}$$

The alternative $\Phi_{T,g}^{(i)}(\bar{\eta})$ and $\Omega_{T+g}^{(i,i)}$ values may now be used in equation (13.17).

13.4. *Conditional Forecasting with the VAR Model*

Conditional forecasting with a reduced form Bayesian VAR can be conducted in YADA using the ideas from Waggoner and Zha (1999). That is, reduced form shocks ε_{T+i} are drawn over the conditioning sample from a normal distribution with a mean and covariance matrix which guarantees that the assumptions are satisfied. The approach of fixing certain shocks is not used for the VAR since the individual reduced form shocks do not have any particular interpretation other than being 1-step ahead forecast errors when the parameters are given.

The conditioning assumptions for the VAR model are expressed as

$$\tilde{z}_{T+i} = \tilde{K}_1'x_{T+i} + \sum_{j=1}^{i-1}\tilde{K}_2'x_{T+i-j} + \tilde{u}_T, \quad i = 1, \ldots, g, \tag{13.18}$$

where $\tilde{z}_{T+i}$ is m dimensional with $m < p$, and $\tilde{u}_T$ is an m dimensional vector of known constants. The matrices $\tilde{K}_1$ and $\tilde{K}_2$ are $p \times m$ with $\text{rank}(\tilde{K}_1) = m$.

To derive the moments of the distribution for the shocks over the conditioning sample it is convenient to write the VAR model on state-space form. Hence, let the measurement equation be given by

$$x_t = \Psi d_t + J_k'Y_t, \tag{13.19}$$

while the state equation is

$$Y_t = BY_{t-1} + J_k\varepsilon_t, \tag{13.20}$$

where J_k, Y_t, and B are defined in Section 13.1.2.

For convenience and, as we shall see below, without loss of generality, let

$$\varepsilon_t = \Omega^{1/2}\tilde{\eta}_t, \tag{13.21}$$

where $\Omega^{1/2}$ is any matrix such that $\Omega = \Omega^{1/2}\Omega^{1/2\prime}$. The shocks $\tilde{\eta}_t$ are i.i.d. $N(0, I_p)$. The first step shall be to derive the mean and the covariance matrix for these normalized shocks which guarantee that the conditioning assumptions in (13.18) are satisfied. We shall then translate

these moments into the ones that apply for the reduced form shocks ε_t. Given that the moments do not involve a specific choice for $\Omega^{1/2}$, the above claim is implied.

The state equation for period $T+i$ can be expressed relative to Y_T and the normalized shocks from periods $T+1,\dots,T+i$ by

$$Y_{T+i} = B^i Y_T + \sum_{j=0}^{i-1} B^j J_k \Omega^{1/2} \tilde{\eta}_{T+i-j}, \quad i = 1,\dots,g.$$

Substituting this into the measurement equation for x_{T+i} we obtain

$$x_{T+i} = \Psi d_{T+i} + J_k' B^i Y_T + \sum_{j=0}^{i-1} J_k' B^j J_k \Omega^{1/2} \tilde{\eta}_{T+i-j}, \quad i = 1,\dots,g.$$

For periods $T+1,\dots,T+g$ we can stack these equations as

$$\begin{bmatrix} x_{T+g} \\ x_{T+g-1} \\ \vdots \\ x_{T+1} \end{bmatrix} = \begin{bmatrix} \Psi d_{T+g} \\ \Psi d_{T+g-1} \\ \vdots \\ \Psi d_{T+1} \end{bmatrix} + \begin{bmatrix} J_k' B^g \\ J_k' B^{g-1} \\ \vdots \\ J_k' B \end{bmatrix} Y_T + \\ + \begin{bmatrix} \Omega^{1/2} & J_k' B J_k \Omega^{1/2} & \cdots & J_k' B^{g-1} J_k \Omega^{1/2} \\ 0 & \Omega^{1/2} & & J_k' B^{g-2} J_k \Omega^{1/2} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & & \Omega^{1/2} \end{bmatrix} \begin{bmatrix} \tilde{\eta}_{T+g} \\ \tilde{\eta}_{T+g-1} \\ \vdots \\ \tilde{\eta}_{T+1} \end{bmatrix},$$

or

$$X_{T+g} = \Psi_{T+g} + G Y_T + D \tilde{N}_{T+g}. \tag{13.22}$$

The conditioning assumptions in (13.18) can be stacked as:

$$\begin{bmatrix} \tilde{z}_{T+g} \\ \tilde{z}_{T+g-1} \\ \vdots \\ \tilde{z}_{T+1} \end{bmatrix} = \begin{bmatrix} \tilde{K}_1' & \tilde{K}_2' & \cdots & \tilde{K}_2' \\ 0 & \tilde{K}_1' & & \tilde{K}_2' \\ \vdots & & \ddots & \\ 0 & 0 & & \tilde{K}_1' \end{bmatrix} \begin{bmatrix} x_{T+g} \\ x_{T+g-1} \\ \vdots \\ x_{T+1} \end{bmatrix} + \begin{bmatrix} \tilde{u}_T \\ \tilde{u}_T \\ \vdots \\ \tilde{u}_T \end{bmatrix},$$

or

$$\tilde{Z}_{T+g} = \tilde{K}' X_{T+g} + \tilde{U}_T. \tag{13.23}$$

Substituting for X_{T+g} from (13.22) in (13.23) and rearranging terms gives us the following linear restrictions that the shocks $\tilde{N}_{T+g}$ must satisfy in order to meet the conditioning assumptions

$$\tilde{K}' D \tilde{N}_{T+g} = \tilde{k}_{T+g}, \tag{13.24}$$

where $\tilde{k}_{T+g} = \tilde{Z}_{T+g} - \tilde{U}_T - \tilde{K}'(\Psi_{T+g} + G Y_T)$.

Like in Waggoner and Zha (1999), the distribution of the shocks $\tilde{N}_{T+g}$ conditional on the restriction (13.24) is normal with mean $\mu_{\tilde{N},T+g}$ and idempotent covariance matrix $\Sigma_{\tilde{N},T+g}$. We here find that

$$\begin{aligned} \mu_{\tilde{N},T+g} &= D' \tilde{K} \left(\tilde{K}' D D' \tilde{K} \right)^{-1} \tilde{k}_{T+g}, \\ \Sigma_{\tilde{N},T+g} &= I_{pg} - D' \tilde{K} \left(\tilde{K}' D D' \tilde{K} \right)^{-1} \tilde{K}' D. \end{aligned} \tag{13.25}$$

This concludes the first step of deriving the mean and the covariance matrix that the standardized shocks should be drawn from to ensure that the conditioning assumptions are satisfied.

For the second step, where we will show that the choice of $\Omega^{1/2}$ does not have any effect on the reduced form shocks subject to the conditioning assumptions, let $\mathcal{E}_{T+g}$ be the stacking of

$\varepsilon_{T+g}, \ldots, \varepsilon_{T+1}$. This means that

$$\boldsymbol{\mathcal{E}}_{T+g} = \big(I_g \otimes \Omega^{1/2}\big)\tilde{N}_{T+g}. \tag{13.26}$$

The restriction (13.24) can now be expressed in terms of $\boldsymbol{\mathcal{E}}_{T+g}$ as

$$\tilde{K}'\tilde{D}\boldsymbol{\mathcal{E}}_{T+g} = \tilde{k}_{T+g}. \tag{13.27}$$

The matrix $\tilde{D} = D(I_g \otimes \Omega^{-1/2})$ and does not depend on $\Omega^{1/2}$. In fact

$$\tilde{D} = \begin{bmatrix} I_p & J_k'BJ_k & \cdots & J_k'B^{g-1}J_k \\ 0 & I_p & & J_k'B^{g-2}J_k \\ \vdots & & \ddots & \vdots \\ 0 & 0 & & I_p \end{bmatrix}.$$

Moreover, the definition in (13.26) also means that the distribution of the reduced form shocks $\boldsymbol{\mathcal{E}}_{T+g}$ conditional on the restriction (13.27) is normal with mean $\mu_{\mathcal{E},T+g}$ and covariance matrix $\Sigma_{\mathcal{E},T+g}$. These moments are equal to

$$\begin{aligned} \mu_{\mathcal{E},T+g} &= \big(I_g \otimes \Omega\big)\tilde{D}'\tilde{K}\Big[\tilde{K}'\tilde{D}\big(I_g \otimes \Omega\big)\tilde{D}'\tilde{K}\Big]^{-1}\tilde{k}_{T+g}, \\ \Sigma_{\mathcal{E},T+g} &= \big(I_g \otimes \Omega\big) - \big(I_g \otimes \Omega\big)\tilde{D}'\tilde{K}\Big[\tilde{K}'\tilde{D}\big(I_g \otimes \Omega\big)\tilde{D}'\tilde{K}\Big]^{-1}\tilde{K}'\tilde{D}\big(I_g \otimes \Omega\big). \end{aligned} \tag{13.28}$$

From these expressions we find that the moments do not depend on a particular choice of $\Omega^{1/2}$ and the claim has therefore been established.

The computation of the conditional predictive distribution can now proceed as follow. For a draw $\theta = (\Psi, \Pi, \Omega)$ from the joint posterior distribution, we may first draw $\boldsymbol{\mathcal{E}}_{T+g}$ from $N(\mu_{\mathcal{E},T+g}, \Sigma_{\mathcal{E},T+g})$ thus yielding a sequence of shocks, $\varepsilon_{T+1}, \ldots, \varepsilon_{T+g}$, which guarantees that the conditioning assumptions (13.18) are met. Next, we draw ε_{T+i} for $i = g+1, \ldots, h$ from $N(0, \Omega)$. With these shocks we can simulate the path $x_{T+1}, \ldots, x_{T+h}$ by feeding the residuals into the VAR system (12.1). Repeating this P times for the given θ gives us P sample paths from the predictive distribution conditional on the historical data, the conditioning assumptions, and θ. Repeating the above procedure for S draws of θ from its joint posterior distribution means that we end up with PS paths of $x_{T+1}, \ldots, x_{T+h}$ from the conditional predictive distribution.

For each draw θ we can also estimate the population mean of $x_{T+1}, \ldots, x_{T+h}$ by letting $\varepsilon_{T+1}, \ldots, \varepsilon_{T+g}$ be equal to $\mu_{\mathcal{E},T+g}$. The shocks ε_{T+i} are next set to zero for $i = g+1, \ldots, h$. By feeding these shock values into the VAR system we obtain a path for $E[x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g}; \theta]$, $i = 1, \ldots, h$. Repeating this S times for the different θ draws we may estimate the population mean of the conditional predictive distribution by taking the average of these S paths.

The modesty analysis can also be performed in the VAR setting. Like in the case of the state-space model we can consider one multivariate and two univariate statistics. These are again based on the ideas of Adolfson et al. (2005) and Leeper and Zha (2003). For a given draw $\bar{\boldsymbol{\mathcal{E}}}_{T+g}$ from $N(\mu_{\mathcal{E},T+g}, \Sigma_{\mathcal{E},T+g})$ the difference between the period $T+g$ simulated conditional forecast value of the endogenous variables and the unconditional forecast (given θ) is

$$\Phi_{T,g}\big(\bar{\boldsymbol{\mathcal{E}}}_{T+g}\big) = x_{T+g}\big(\bar{\boldsymbol{\mathcal{E}}}_{T+g}; \theta\big) - E\big[x_{T+g}|\mathfrak{D}_T; \theta\big] = \sum_{j=0}^{g-1} J_k'B^jJ_k\bar{\varepsilon}_{T+g-j}, \tag{13.29}$$

where $\bar{\boldsymbol{\mathcal{E}}}_{T+g} = [\bar{\varepsilon}_{T+g}' \cdots \bar{\varepsilon}_{T+1}']'$. The forecast error covariance matrix for the unconditional forecast of x_{T+g} is

$$\Omega_{T+g} = \sum_{j=0}^{g-1} J_k'B^jJ_k\Omega\big(J_k'B^jJ_k\big)'. \tag{13.30}$$

From these expressions we can define a multivariate modesty statistic as

$$\mathcal{M}_{T,g}\big(\bar{\boldsymbol{\mathcal{E}}}_{T+g}\big) = \Phi_{T,g}\big(\bar{\boldsymbol{\mathcal{E}}}_{T+g}\big)'\Omega_{T+g}^{-1}\Phi_{T,g}\big(\bar{\boldsymbol{\mathcal{E}}}_{T+g}\big). \tag{13.31}$$

Under the hypothesis that the conditioning shocks are modest this statistic is $\chi^2(p)$. An alternative reference distribution can be generated by computing the same statistic with $\bar{\boldsymbol{\mathcal{E}}}_{T+g}$ replaced with ε_{T+i} drawn from $N(0,\Omega)$ for $i=1,\ldots,g$ and defining this reference statistic as $\mathcal{M}_{T,g}(\boldsymbol{\mathcal{E}}_{T+g})$. The event $\{\mathcal{M}_{T,g}(\boldsymbol{\mathcal{E}}_{T+g}) \geq \mathcal{M}_{T,g}(\bar{\boldsymbol{\mathcal{E}}}_{T+g})\}$ can then be tested for each one of the PS conditional forecast paths that is computed, making it possible to estimate the probability of this event. If the probability is sufficiently small we may say the hypothesis of modest conditioning assumptions is rejected.

Univariate modesty statistics can now be specified by selecting elements from the vector in (13.29) and the matrix in (13.30). Specifically, we let

$$\mathcal{M}_{T,g}^{(i)}\left(\bar{\boldsymbol{\mathcal{E}}}_{T+g}\right) = \frac{\Phi_{T,g}^{(i)}\left(\bar{\boldsymbol{\mathcal{E}}}_{T+g}\right)}{\sqrt{\Omega_{T+g}^{(i,i)}}}, \quad i=1,\ldots,p. \tag{13.32}$$

This statistic has a standard normal distribution under the assumption that the conditioning information is modest and, like the multivariate statistic, it takes into account that there is uncertainty about all shocks.

For a Leeper-Zha type of univariate modesty statistic we set the reduced form shocks equal to the mean $\mu_{\mathcal{E},T+g} = [\bar{\mu}'_{\mathcal{E},T+g} \cdots \bar{\mu}_{\mathcal{E},T+1}]'$ value. The covariance matrix for the forecast errors thus becomes singular and is given by

$$\bar{\Omega}_{T+g} = \sum_{j=0}^{g-1} J_k' B^j J_k \bar{\Sigma}_{\mathcal{E},T+g-j} \left(J_k' B^j J_k\right)'. \tag{13.33}$$

where

$$\bar{\Sigma}_{\mathcal{E},T+j} = \bar{\mu}_{\mathcal{E},T+j}\left(\bar{\mu}'_{\mathcal{E},T+j}\bar{\mu}_{\mathcal{E},T+j}\right)^{-1}\bar{\mu}'_{\mathcal{E},T+j}\Omega\bar{\mu}_{\mathcal{E},T+j}\left(\bar{\mu}'_{\mathcal{E},T+j}\bar{\mu}_{\mathcal{E},T+j}\right)^{-1}\bar{\mu}'_{\mathcal{E},T+j},$$

for $j=1,\ldots,g$. The univariate Leeper-Zha type of modesty statistic is now given by

$$\mathcal{M}_{T,g}^{(i)}\left(\mu_{\mathcal{E},T+g}\right) = \frac{\Phi_{T,g}^{(i)}\left(\mu_{\mathcal{E},T+g}\right)}{\sqrt{\bar{\Omega}_{T+g}^{(i,i)}}}, \quad i=1,\ldots,p. \tag{13.34}$$

This statistic can now be compared with a standard normal distribution.

The prediction uncertainty of the conditional forecasts can be decomposed into error (or residual) uncertainty and parameter uncertainty. The equivalent to equation (13.4) is now

$$Cov\left(x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g}\right) = E_T\left[Cov\left(x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g};\theta\right)\right] + Cov_T\left[E\left(x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g};\theta\right)\right], \tag{13.35}$$

for $i=1,\ldots,h$, where the E_T and Cov_T as in Section 13.1 denotes the expectation and covariance with respect to the posterior of θ at time T. Once again, the deterministic variables over the prediction horizon have been suppressed from the expression.

To parameterize these terms we first note that

$$E\left[x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g};\theta\right] = \Psi d_{T+i} + J_k' B^i Y_T + \sum_{j=0}^{i-1} J_k' B^j J_k \bar{\mu}_{\mathcal{E},T+i-j},$$

where $\bar{\mu}_{\mathcal{E},T+i-j} = 0$ if $i-j>g$. These expected values satisfy the conditioning assumptions for $i=1,\ldots,g$. Moreover, the forecast error for a given θ is

$$x_{T+i} - E\left[x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g};\theta\right] = \sum_{j=0}^{i-1} J_k' B^j J_k\left(\varepsilon_{T+i-j} - \bar{\mu}_{\mathcal{E},T+i-j}\right).$$

Next, the covariance matrix $\Sigma_{\mathcal{E},T+g}$ is invariant to T since $\tilde{D}$ and $\tilde{K}$ are both invariant to T. Partitioning this $gp \times gp$ matrix as follows

$$\Sigma_{\mathcal{E},T+g} = \begin{bmatrix} \bar{\Sigma}^{(g,g)} & \cdots & \bar{\Sigma}^{(g,1)} \\ \vdots & \ddots & \vdots \\ \bar{\Sigma}^{(1,g)} & \cdots & \bar{\Sigma}^{(1,1)} \end{bmatrix},$$

where $\bar{\Sigma}^{(i,j)} = \bar{\Sigma}^{(j,i)\prime}$ is the $p \times p$ covariance matrix for the vector pair $(\varepsilon_{T+i}, \varepsilon_{T+j})$ for all $i, j = 1, \ldots, g$. The forecast error covariance matrix of x_{T+i} conditional on $\mathfrak{D}_T, \tilde{Z}_{T+g}$ and θ is now equal to

$$Cov\left[x_{T+i}|\mathfrak{D}_T, \tilde{Z}_{T+g}; \theta\right] = \sum_{j=0}^{i-1}\sum_{l=0}^{i-1} J_k' B^j J_k \bar{\Omega}^{(i,j)} J_k' \left(B'\right)^l J_k,$$

where

$$\bar{\Omega}^{(i,j)} = \begin{cases} \bar{\Sigma}^{(i,j)}, & \text{if } i, j = 1, \ldots, g, \\ \Omega, & \text{if } i = j, i = g+1, \ldots, h \\ 0 & \text{otherwise.} \end{cases}$$

These covariance matrices also satisfy the conditioning assumptions, meaning that, for instance, $\tilde{K}_1' \bar{\Sigma}^{(1,1)} = 0$.

13.5. *YADA Code*

The main functions for computing unconditional prediction paths for the observed variables of the DSGE model are called `DSGEPredictionPathsTheta` and `DSGEPredictionPaths`. The former works with a fixed value for the model parameters θ, while the latter works for a set of draws from the posterior distribution of θ. The calculation of the conditional prediction paths for the observed variables of the DSGE model are handled by `DSGECondPredictionPathsTheta` and `DSGECondPredictionPaths`.

Unconditional prediction paths can also be calculated for the Bayesian VAR models. The main functions for this objective are `BVARPredictionPathsPostMode` and `BVARPredictionPaths`. The former function uses a fixed value for the model parameters, while the latter uses draws from the posterior distribution of (Ψ, Π, Ω). Hence, the latter function makes it possible to estimate a predictive distribution of the Bayesian VAR that does not depend on the particular values of the model parameters.

13.5.1. `DSGEPredictionPathsTheta`

The function `DSGEPredictionPathsTheta` needs 11 inputs. First of all, a set of values for the parameters θ is supplied through the variable `theta`. To use the values properly the vector structure `thetaPositions` and the structure `ModelParameter` that were discussed in Section 8.2 are also needed. Furthermore, the DSGE model information structure `DSGEModel` and the generic initialization structure `CurrINI` must be supplied to the function. The following input is given by the $k \times h$ matrix `X` with the values of the exogenous variables over the h period long prediction sample. Next, the value of h is accepted since `X` is empty if $k = 0$. The 8th input is called `NumPaths` and specifies how many prediction paths to compute, while the boolean variable `AnnualizeData` indicates if the prediction paths should be annualized or not. Similarly, the boolean variable `TransData` indicates if the data should be transformed or not. The final input is given by `NameStr` which indicates the type of values that are used for θ, e.g., the posterior mode estimate.

The main output from the function is the 3-dimensional matrix `PredPaths` and the matrices `PredEventData` and `YObsEventData`. The dimensions of the `PredPaths` matrix are given by the number of observed variables, the length of the prediction sample, and the number of prediction paths. The matrix `PredEventData` has as many rows as number of observed variables and two columns. The first column gives the number of times that a prediction event is true, while the second column holds the number of times the event has been tested. A prediction event can, for instance, be defined as non-negative inflation for h^* consecutive periods over the prediction period. The h^* integer of always less than or equal to the length of the prediction period. Similarly, the matrix `YObsEventData` has the same dimension as `PredEventData` and holds prediction event data when the mean of the predictive distribution is equal to the realized values for the observed data.

13.5.2. DSGEPredictionPaths

The function DSGEPredictionPaths requires 12 inputs. Relative to the previous function, DSGEPredictionPathsTheta, there is one additional input (the first) and the last input is different. Before the theta input the current function accepts the matrix thetaPostSample with NumDraws rows and NumParam columns. The number of draws from the posterior that are used can vary irrespective of how many draws from the posterior that are available. Typically, the number of draws of θ from its posterior that are sent to this function is a small number, such as 500 or 1,000. The last input used by the function is CurrChain which is an integer that indicates the MCMC chain number.

As output the function provides 6 variables. The first is the boolean DoneCalc that indicates if all calculations were performed or not. The next is the matrix PredEventData with prediction event results. The final 4 variables provides the data on the prediction variance decompositions for the observed variables over the whole prediction horizon. These variables are called StateCov, ShockCov, MeasureCov, and ParameterCov, respectively. Apart from MeasureCov these are all 3D matrices with dimensions $n \times n \times h$, where h is the length of the prediction horizon, while MeasureCov is $n \times n$.

The prediction paths are not directly sent as output from the function. These are instead written to disk in mat-files, one for each parameter draw. In each file the 3D matrix PredPaths is stored. Its dimensions are given by the number of observed variables, the length of the prediction sample, and the number of prediction paths.

13.5.3. DSGECondPredictionPathsTheta

The function DSGECondPredictionPathsTheta needs 13 inputs. The first 6 and the last 5 are the same inputs as the function DSGEPredictionPathsTheta takes. The 2 additional inputs naturally refer to the conditioning information. Specifically, the 7th input is given by Z, an $q_m \times g$ matrix with the conditioning data $[z_{T+1} \ \cdots \ z_{T+g}]$. The 8th input is given by U, an $q_m \times g$ matrix with the initial values $[u_{T-g+1} \ \cdots \ u_T]$ for the conditioning; cf. equation (13.7).

The main output from the function is the 3-dimensional matrix PredPaths, the matrices with prediction event test results, PredEventData and YObsEventData, and the modesty results, MultiModestyStat, UniModestyStat, and UniModestyStatLZ. The dimension of PredPaths is given by the number of observed variables, the length of the prediction sample, and the number of prediction paths. The matrix PredEventData (and YObsEventData) has as many rows as number of observed variables and two columns. The first column gives the number of times that a prediction event is true, while the second column holds the number of times the event has been tested. A prediction event can, for instance, be defined as non-negative inflation for h^* consecutive periods over the prediction period. The h^* integer of always less than or equal to the length of the prediction period. The difference between PredEventData and YObsEventData is that the latter matrix holds prediction event results when the mean of the predictive distribution has been set equal to the realized values of the observed variables.

The modesty statistics are only calculated when AnnualizeData is zero. When this condition is met, MultiModestyStat is a matrix of dimension NumPaths times 2, where the first columns holds the values of $\mathcal{M}_{T,g}(\bar{\eta})$, while the second column gives $\mathcal{M}_{T,g}(\eta)$. The matrix UniModestyStat has dimension NumPaths times n and gives the univariate modesty statistics in equation (13.17), while UniModestyStatLZ is a vector with the n values of the univariate Leeper-Zha related modesty statistic.

13.5.4. DSGECondPredictionPaths

The function DSGECondPredictionPaths for computing the conditional predictive distribution requires 14 inputs. The first 7 and the last 5 input variables are the same as those that the function DSGEPredictionPaths takes. The two additional inputs refer to the same data that the function DSGECondPredictionPathsTheta requires, i.e., to Z and U.

Moreover, as in the case of DSGEPredictionPaths for the unconditional predictive distribution, the majority of the output from this function is not sent through its output arguments, but

are written to disk. For instance, the prediction paths are written to disk in mat-files, one for each parameter draw. In each file the 3D matrix PredPaths is stored. Its dimensions are given by the number of observed variables, the length of the prediction sample, and the number of prediction paths. Moreover, the multivariate and univariate modesty statistics are calculated and saved to disk provided that the AnnualizeData variable is zero (no annualization).

The function provides the same 6 output arguments as the DSGEPredictionPaths function. Moreover, the prediction paths data is also written to disk in mat-files, one for each parameter draw. In each file the 3D matrix PredPaths is stored. Its dimensions are given by the number of observed variables, the length of the prediction sample, and the number of prediction paths.

13.5.5. BVARPredictionPathsPostMode

The function BVARPredictionPathsPostMode requires 9 inputs. The first group is Psi, Pi, and Omega with fixed values for Ψ, Π, and Ω, respectively. Next, the function takes the structures DSGEModel and CurrINI. Furthermore, the $p \times h$ matrix DPred with data on the exogenous variables over the h periods in the prediction sample as well as h, the prediction sample length are needed. Finally, the function requires the integer NumPaths and the boolean AnnualizaData. The former determines the number of prediction paths to compute at the fixed parameter value, while the latter indicates if the prediction paths should be annualized or not.

The number of output variables is equal to 5. The first is the 3-dimensional matrix PredPaths, whose dimensions are given by the number of observed variables, the length of the prediction sample, and the number of prediction paths. The second output variable is PredMean, a matrix with the population mean predictions of the observed variables. The following output variable is the matrix PredEventData, which stores the prediction event results. It has as many rows as there are number of observed variables and two columns. The first column holds the number of times that a prediction event is true, while the second column keeps the number of times the event has been tested. The last two output variables are called KernelX and KernelY, 3-dimensional matrices with kernel density estimates of the marginal predictive densities. The dimensions of both matrices are equal to the number of observed variables, the number of grid points, and the prediction sample length.

13.5.6. BVARPredictionPaths

The function BVARPredictionPaths also requires 9 inputs. The final 6 input variables are identical to the final 6 input variables for the BVARPredictionPathsPostMode function. The first 3, however, are now given by the matrices PsiPostSample, PiPostSample, and OmegaPostSample. The number of rows of these matrices is NumDraws, while the number of columns is equal to the number of parameters of Ψ, Π, and Ω, respectively.

The function gives 4 variables as output. First, the boolean variable DoneCalc indicates if the calculations were finished or not. The second output is PredEventData, a $p \times 2$ matrix with the prediction event results. Furthermore, the prediction uncertainty decomposition into the error uncertainty and the parameter uncertainty is provided through the 3D matrices ShockCov and ParameterCov. The dimensions of these matrices are $p \times p \times h$, where h is the length of the prediction sample. This decomposition is only calculated when the boolean input variable AnnualizeData is zero.

Adolfson, M., Anderson, M. K., Lindé, J., Villani, M., and Vredin, A. (2005). *Modern Forecasting Models in Action: Improving Macroeconomic Analyses at Central Banks.* (Sveriges Riksbank Working Paper Series No. 188.)

Adolfson, M., Laséen, S., Lindé, J., and Svensson, L. E. O. (2008). *Optimal Monetary Policy in an Operational Medium-Sized DSGE Model.* (Manuscript, Sveriges Riksbank.)

Adolfson, M., Laséen, S., Lindé, J., and Villani, M. (2005). "Are Constant Interest Rate Forecasts Modest Policy Interventions? Evidence from a Dynamic Open-Economy Model." *International Finance*, *8*, 509–544.

Adolfson, M., Laséen, S., Lindé, J., and Villani, M. (2007a). "Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through." *Journal of International Economics*, *72*, 481–511.

Adolfson, M., Laséen, S., Lindé, J., and Villani, M. (2007b). "Evaluating an Estimated New Keynesian Small Open Economy Model." *Journal of Economic Dynamics and Control*. (Forthcoming. Also available as Sveriges Riksbank Working Paper Series No. 203.)

Adolfson, M., Lindé, J., and Villani, M. (2007a). "Bayesian Analysis of DSGE Models - Some Comments." *Econometric Reviews*, *26*, 173–185.

Adolfson, M., Lindé, J., and Villani, M. (2007b). "Forecasting Performance of an Open Economy DSGE Model." *Econometric Reviews*, *26*, 289–328.

An, S., and Schorfheide, F. (2007). "Bayesian Analysis of DSGE Models." *Econometric Reviews*, *26*, 113–172.

Anderson, E. W., Hansen, L. P., McGrattan, E. R., and Sargent, T. J. (1996). "Mechanics of Forming and Estimating Dynamic Linear Economies." In H. M. Amman, D. A. Kendrick, and J. Rust (Eds.), *Handbook of Computational Economics* (pp. 171–252). Elsevier.

Anderson, G. (1999). *The Anderson-Moore Algorithm: A MATLAB Implementation.* (Mimeo, Board of Governors of the Federal Reserve System.)

Anderson, G., and Moore, G. (1983). *An Efficient Procedure for Solving Linear Prefect Foresight Models.* (Mimeo, Board of Governors of the Federal Reserve System.)

Anderson, G., and Moore, G. (1985). "A Linear Algebraic Procedure for Solving Linear Perfect Foresight Models." *Economics Letters*, *17*, 247–252.

Arnold, I., W. F., and Laub, A. J. (1984). "Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations." *Proceedings of the IEEE*, *72*, 1746–1754.

Bartlett, M. S. (1957). "A Comment on D. V. Lindley's Statistical Paradox." *Biometrika*, *44*, 533–534.

Bauwens, L., Lubrano, M., and Richard, J. F. (1999). *Bayesian Inference in Dynamic Econometric Models.* Oxford: Oxford University Press.

Bernardo, J. M., and Smith, A. F. M. (2000). *Bayesian Theory.* Chichester: John Wiley.

Beyer, A., and Farmer, R. E. A. (2004). *On the Indeterminancy of New-Keynesian Economics.* (ECB Working Paper Series No. 323.)

Brooks, S. P., and Gelman, A. (1998). "General Methods for Monitoring Convergence of Iterative Simulations." *Journal of Computational and Graphical Statistics*, *7*, 434–455.

Canova, F., and Sala, L. (2006). *Back to Square One: Identification Issues in DSGE Models.* (ECB Working Paper Series No. 583.)

Casella, G., and George, E. I. (1992). "Explaining the Gibbs Sampler." *The American Statistician*, *46*, 167–174.

Chib, S. (1995). "Marginal Likelihood from the Gibbs Output." *Journal of the American Statistical Association*, *90*, 1313–1321.

Chib, S., and Greenberg, E. (1995). "Understanding the Metropolis-Hastings Algorithm." *The American Statistician*, *49*, 327–335.

Chib, S., and Jeliazkov, I. (2001). "Marginal Likelihood from the Metropolis-Hastings Output." *Journal of the American Statistical Association*, *96*, 270–281.

Christiano, L., Eichenbaum, M., and Evans, C. (2005). "Nominal Rigidities and the Dynamic Effects of a Shock to Monetary Policy." *Journal of Political Economy*, *113*, 1–45.

Christoffel, K., Coenen, G., and Warne, A. (2008). *The New Area-Wide Model of the Euro Area: A Micro-Founded Open-Economy Model for Forecasting and Policy Analysis.* (ECB Working Paper Series No. 944.)

Cowles, M. K., and Carlin, B. P. (1996). "Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review." *Journal of the American Statistical Association*, *91*, 883–904.

Fernández-Villaverde, J., Rubio-Ramírez, J. F., Sargent, T. J., and Watson, M. W. (2007). "ABCs (and Ds) of Understanding VARs." *American Economic Review*, *97*, 1021–1026.

Galí, J. (1999). "Technology, Employment, and the Business Cycle: Do Technology Shocks Explain Aggregate Fluctuations?" *American Economic Review*, *89*, 249–271.

Gelfand, A., and Dey, D. (1994). "Bayesian Model Choice: Asymptotics and Exact Calculations." *Journal of the Royal Statistical Society B*, *56*, 501–514.

Gelman, A. (1996). "Inference and Monitoring Convergence." In W. R. Gilks, S. Richardson, and D. J. Spiegelhalter (Eds.), *Markov Chain Monte Carlo in Practice* (pp. 131–143). Boca Raton: Chapman & Hall/CRC.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis* (Second ed.). Boca Raton: Chapman & Hall/CRC.

Gelman, A., and Rubin, D. B. (1992). "Inference from Iterative Simulations Using Multiple Sequences." *Statistical Science*, *7*, 457–511.

Geweke, J. (1999). "Using Simulation Methods for Bayesian Econometric Models: Inference, Development, and Communication." *Econometric Reviews*, *18*, 1–73.

Geweke, J. (2005). *Contemporary Bayesian Econometrics and Statistics.* Hoboken: John Wiley.

Hamilton, J. D. (1994). *Time Series Analysis.* Princeton: Princeton University Press.

Hastings, W. K. (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, *57*, 97–109.

Iskrev, N. (2006). *Identification and Estimation of DSGE Models: An Integrated Approach.* (Manuscript, University of Michigan.)

Iskrev, N. (2007a). "Evaluating the Information Matrix in Linearized DSGE Models." *Economics Letters*, Forthcoming.

Iskrev, N. (2007b). *How Much Do We Learn from the Estimation of DSGE Models? A Case Study of Idenitification Issues in a New Keynesian Business Cycle Model.* (Manuscript, University of Michigan.)

Kadiyala, K. R., and Karlsson, S. (1997). "Numerical Methods for Estimation and Inference in Bayesian VAR-Models." *Journal of Applied Econometrics*, *12*, 99–132.

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems." *Journal of Basic Engineering*, *82*, 35–45.

Kalman, R. E., and Bucy, R. S. (1961). "New Results in Linear Filtering and Prediction Theory." *Journal of Basic Engineering*, *83*, 95–108.

Klein, A., and Neudecker, H. (2000). "A Direct Derivation of the Exact Fisher Information Matrix of Gaussian Vector State Space Models." *Linear Algebra and its Applications*, *321*, 233–238.

Kydland, F. E., and Prescott, E. C. (1982). "Time to Build and Aggregate Fluctuations." *Econometrica*, *50*, 1345–1370.

Leeper, E. M., and Zha, T. (2003). "Modest Policy Interventions." *Journal of Monetary Economics*, *50*, 1673–1700.

Lindley, D. V. (1957). "A Statistical Paradox." *Biometrika*, *44*, 187–192.

Lucas, R. E., Jr. (1976). "Econometric Policy Evaluation: A Critique." In K. Brunner and A. H. Meltzer (Eds.), *Carnegie-Rochester Series on Public Policy, vol. 1* (pp. 19–46). Amsterdam: North-Holland.

Magnus, J. R., and Neudecker, H. (1988). *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Chichester: John Wiley.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). "Equations of State Calculations by Fast Computing Machines." *Journal of Chemical Physics*, *21*, 1087–1091.

Newey, W. K., and West, K. D. (1987). "A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix." *Econometrica*, *55*, 703–708.

Otrok, C. (2001). "On Measuring the Welfare Cost of Business Cycles." *Journal of Monetary Economics*, *47*, 61–92.

Raftery, A. E. (1996). "Hypothesis Testing and Model Selection." In W. R. Gilks, S. Richardson, and D. J. Spiegelhalter (Eds.), *Markov Chain Monte Carlo in Practice* (pp. 163–187). Boca Raton: Chapman & Hall/CRC.

Rothenberg, T. J. (1971). "Identification in Parametric Models." *Econometrica*, *39*, 577-591.

Schorfheide, F. (2000). "Loss Function-Based Evaluation of DSGE Models." *Journal of Applied Econometrics*, *15*, 645–670.

Smets, F., and Wouters, R. (2003). "An Estimated Stochastic Dynamic General Equilibrium Model for the Euro Area." *Journal of the European Economic Association*, *1*, 1123–1175.

Smets, F., and Wouters, R. (2005). "Comparing Shocks and Frictions in U.S. and Euro Area Business Cycles: A Bayesian DSGE Approach." *Journal of Applied Econometrics*, *20*, 161–183.

Thompson, P. A., and Miller, R. B. (1986). "Sampling the Future: A Bayesian Approach to Forecasting from Univariate Time Series Models." *Journal of Business & Economic Statistics*, *4*, 427–436.

Tierney, L. (1994). "Markov Chains for Exploring Posterior Distributions." *Annals of Statistics*, *22*, 1701–1762. (With discussion.)

Tierney, L., and Kadane, J. B. (1986). "Accurate Approximations for Posterior Moments and Marginal Densities." *Journal of the American Statistical Association*, *81*, 82–86.

Villani, M. (2001). "Bayesian Prediction with Cointegrated Vector Autoregressions." *International Journal of Forecasting*, *17*, 585–605.

Villani, M. (2005). "Bayesian Reference Analysis of Cointegration." *Econometric Theory*, *21*, 326–357.

Villani, M. (2007). "Steady State Priors for Vector Autoregressions." *Journal of Applied Econometrics*, Forthcoming.

Waggoner, D. F., and Zha, T. (1999). "Conditional Forecasts in Dynamic Multivariate Models." *Review of Economics and Statistics*, *81*, 639–651.

Warne, A. (2006). *Bayesian Inference in Cointegrated VAR Models: With Applications to the Demand for Euro Area M3*. (ECB Working Paper Series No. 692.)

Yu, B., and Mykland, P. (1998). "Looking at Markov Samplers Through CUSUM Path Plots: A Simple Diagnostic Idea." *Statistics and Computing*, *8*, 275–286.

Zadrozny, P. A. (1989). "Analytical Derivatives for Estimation of Linear Dynamic Models." *Computers and Mathematics with Applications*, *18*, 539–553.

Zadrozny, P. A. (1992). "Errata to 'Analytical Derivatives for Estimation of Linear Dynamic Models'." *Computers and Mathematics with Applications*, *24*, 289–290.

Zagaglia, P. (2005). "Solving Rational-Expectations Models through the Anderson-Moore Algorithm: An Introduction to the Matlab Implementation." *Computational Economics*, *26*, 91–106.

Zellner, A. (1971). *An Introduction to Bayesian Inference in Econometrics*. New York: John Wiley.