The Matlab codes perform robustness checks using high precision arithmetic (henceforth MP) for the results reported in the Tables of Qu and Tkachenko (2015): "**Global Identification in DSGE Models Allowing for Indeterminacy**".

The folders and subfolders must be added to Matlab path for the scripts to work. They have been tested on Matlab version 2015a.

You also need to install the multi-precision toolbox by Advanpix available at http://www.advanpix.com/.

This readme file is structured as follows.

First, we give some general information on how the code files are organized.

Second, we provide some details on the toolbox and the implementation.

Third, we use an example to illustrate the structure of the script file that performs a robustness check on a column in one of the tables.

****
***GENERAL INFORMATION
****

The folder structure is kept similar to that of the replication files for the main paper results. Most importantly, the directory "Replication_scripts" contains subfolders with the scripts reproducing and displaying table output labeled by its number and column, e.g., QT_Tables_4_5_part1_mp.m provides a robustness check for the first column of the Tables 4 and 5 etc. They can all be run independently.

The directory 'General' contains Chris Sims' gensys routine modified to allow for indeterminacy and MP computation. It also contains an auxiliary routine that performs fast row echelon form computation compatible with MP (frref.m).

The rest of subroutines are organized by model folder (i.e., AS_Model and SW_Model). Inside each folder, the 'Constraints' folder collects constraint functions used in the global identification analysis and the 'Objectives' folder contains MP versions of m-files that evaluate the objective function for different cases. The .m files under the

root directory compute the model solution and KL distance in MP. The data files contain the benchmark parameter values, the corresponding spectral densities as well as Gaussian quadrature abscissae and weights for the numerical integral approximation.

****

*****ADVANPIX MULTIPRECISION COMPUTING TOOLBOX: BRIEF DESCRIPTION AND NOTES ON IMPLEMENTATION**

****

This toolbox is a Matlab extension that allows computing with arbitrary precision. It has a much wider range of supported operations and is more efficient than the Matlab's built-in Variable Precision Arithmetic function. A free 2-week trial of the full version can be downloaded from www.advanpix.com .

The main reasons for choosing this toolbox for performing the robustness check considered here are as follows:

1) It can perform the QZ decomposition and hence allows computing the numerical model solution with arbitrary precision.
2) It has extensive support for numerical integration techniques in MP, such as Gaussian quadrature.
3) Finally, it allows performing optimization using Nelder-Mead simplex method (Matlab's 'fminsearch' function) fully in MP.

The toolbox allows setting an arbitrary precision level, with the default being 34 digits (quadruple precision). We use this default level of precision throughout, unless explicitly stated otherwise. Precision level can be changed, e.g., to 40 digits, by executing:

```
mp.Digits(40)
```

Increasing the level of precision increases computation time, as more processing power and RAM is required.

The structure of the code is mostly unchanged from the original replication files, except for changes necessary to ensure all computations are performed in MP, and the numerical integral computation, where the Riemann sum approximation is replaced by the MP Gaussian quadrature.

A scalar or a matrix can be created or declared an MP object via the mp() function.

Example 1: For an existing scalar equal to pi evaluated in double precision:

```
pi2 = mp(pi);
```

declares it an MP object, and all further operations supported by the toolbox (addition, subtraction, matrix operations etc.) will be performed in MP. The caveat is that since pi was initially evaluated in double precision, its accuracy will be limited even when it is an MP object.

Example 2: We can create a scalar equal to pi evaluated directly at the specified precision level, by entering it as a string:

```
pi4 = mp('pi');
```

will produce a fully precise (up to number of specified digits) result.

We use the method in Example 1 to convert the original minimizer to an MP object, and use the method in Example 2 to perform all subsequent computations during the optimization robustness check.

Note that once a vector/scalar is detected as an MP object, all the Matlab functions supported by the toolbox will recognize that automatically and perform the computation in MP, so little if any code modification is needed once the proper MP-format inputs are established.

Example 3: Computing eigenvalues of a magic square matrix in MP:

```
A = mp(magic(3)); % a magic square matrix
EV = eig(A);
```

Thus,to perform our computation in MP, we declare the model parameter vector an MP object, and modify the existing code to ensure all numerical scalars and matrices are created as MP objects and all intermediate operations are supported by the toolbox.

****
****REPLACING RIEMANN SUM INTEGRAL APPROXIMATION WITH GAUSSIAN QUADRATURE**
****

The computation of the KL distance between two DSGE models requires numerical integral approximation. In the original code implementation, the Riemann sum approximation is used to evaluate the integral inside the objective function.

Here, we replace it by Gaussian Quadrature evaluated with higher precision.

The details of implementation are as follows.

The integral for the full spectrum case is evaluated between –pi and pi. We set the order of quadrature to 400, as we found this is the order at which integral value stabilizes and does not change visibly when the order is further increased.

The respective abscissae (x) and quadrature weights (wg) are then computed via:

```
[x,wg] = mp.GaussLegendre(400,-pi,pi);
```

For efficiency of computation, we save x and wg, together with the benchmark model spectral density evaluated at frequencies in x in a mat-file (e.g., true_spectrum0swdmp.mat for the Smets and Wouters model), and load it during the call of the objective function. Furthermore, due to the property of the spectral density that:

```
f(w) = conj(f(-w))
```

only the first 200 points are effectively used for computing the spectrum of the alternative model, as the abscissae x are symmetric about 0.

****
*****NELDER-MEAD ALGORITHM: BRIEF DESCRIPTION AND NOTES ON IMPLEMENTATION**
****

The Nelder-Mead simplex search algorithm is the only optimization algorithm supported fully by the MP toolbox. It is implemented through the Matlab ‘fminsearch’ function.

This algorithm is a direct search method and does not involve numerical or analytical gradients. It can handle discontinuities and is expected to perform well in the local search. It does not allow

for parallel computation, so the computational time, especially for the larger Smets and Wouters (2007) model, can be substantial.

The algorithm performs unconstrained optimization. To handle bounds on parameters, we transform the parameters to lie in [-Inf, Inf] using a method in Lubik in Schorfheide (2004), and convert them back after optimization is completed (see transfmpas2.m and transfmpsw.m). To handle inequality constraints, we apply a large penalty to the objective function if the constraint is violated, in the same way as during Particle Swarm minimization in the original code.

The main options for the algorithm are the various stopping conditions: maximum number of iterations, objective function tolerance level, parameter tolerance level.

We set maximum number of iterations to 20000 for the AS model and 6000 for the SW model. In our experience with these models, this is enough for the algorithm to either converge at the given options, or reach a stage where the objective function does not visibly change for several thousand iterations. The objective function tolerance level is set to 1e-30. This is to allow, in particular, to show that using progressively higher precision computation will result in progressively lower objective function values in observationally equivalent cases (AS model, Tables 4 and 5, columns 1 and 2). The parameter tolerance level is set at 1e-12.

****
*****EXAMPLE OF CODE REPLICATING A COLUMN IN TABLE 2**
****

This example demonstrates the structure of replication scripts found in the directory “Replication_scripts”. This is a constrained KL minimization problem, but files for the unconstrained problems have a nearly identical structure.

The code is broken into sections for convenience (lines/sections not preceded by ‘%’ can be copypasted and run directly in Matlab).

```
%% Load previous results and set up

load tables_23_p1_min %load results from GA+Multistart optimization
```

```
%Here the inputs to the objective function are created (wgtmp,conmp)
or declared (thetaind – the originally obtained minimizer)as MP ob-
jects:

wgtmp=mp(ones(1,numpar)); %set weights as mp object

conmp=mp('[1,0.1]'); %set constraint as mp object

resfilename2=[resfilename,'_mp']; %new result file name

thetamp=mp(thetaind)'; %set starting value to the minimizer computed
by GA+Multistart

%% Additional local optimization using MP version of fminsearch

%define objective function – modified for MP computation
ObjFunmp=@(theta0)kloptas_mp(theta0,conmp,wgtmp,nrm,indp,bc);


%set fminsearch options as discussed above
optfms=optimset('fminsearch');
optfms=optimset(optfms,'Display','iter','MaxFunEvals',50000,'MaxIter'
,20000,'TolFun',1e-30,'TolX',1e-12);


%%perform optimization. Note the input to the objective function is
transformed into the unconstrained form using the transmpas2.m func-
tion:

timefms=tic;
[xestmp,fvalmp,eflagmp,outmp] =
fminsearch(ObjFunmp,transfmpas2(thetamp,1,1,[1:15]),optfms);
timelfms=toc(timefms);


%% Arrange and save results. We transform back the minimizer and save
the results:

thetaindmp=transfmpas2(xestmp,2,1,[1:15])'; %recover the new minimiz-
er

klmp=fvalmp; %recover the Kl distance

save(resfilename2) %save results into a new file


%% Display results

%For comparison, compute the KL value at the original minimizer in MP:

klmp_o=ObjFunmp(transfmpas2(thetamp,1,1,[1:15])); %original KL in mp
```

```
%Display the original and the new minimizers and respective KL dis-
tances:

disp('-----------------------------------------------')

disp 'Table 2. Parameter values minimizing the KL criterion, AS (2007)
model'
disp '                                  (a) All parameters can vary,
c=0.1'
disp('                  Original minimizer
New minimizer')
disp([thetamp',thetaindmp]);

display ('                      Original KL
New KL');
display ([klmp_o,klmp]);
```