dynareFindSteadyState:
Prototype Tools for
Finding DSGE Model Steady States

Gary S. Anderson

November 15, 2007

Abstract

Dynare provides many useful tools for solving and estimating Dynamic Stochastic General Equilibrium(DSGE) Models. Dynare facilitates the computation of solutions to linear rational expectations models and provides perturbation techniques for solving nonlinear stochastic models. However, on occasion, economists express some frustration with the first step in applying these tools: finding a steady state solution for the nonlinear dynamic model.

This note describes a prototype software tool for computing steady states. I am distributing this version in the hope that you will find it useful int its current state and so that you might give me feedback that will help me improve the tool.

Contents

1   Introduction and Summary

The **dynareFindSteadyState** program and related software endeavor to completely automate the process of finding a steady state for a dynare DSGE model. The **dynareFindSteadyState** tools have been written for use in MATLAB along with dynare.

To get steady state solutions for a dynare model one may only need to type:

```
>>dynare myModel
>>dynareFindSteadyState
```

For many models, this will generate a file myModelInitVal with instructions characterizing a steady state satisfying Blanchard-Kahn conditions. Cutting and pasting this file into the myModel.mod file will allow dynare's steady command to quickly recalculate this steady state.

Some models may require more time or more function evaluations than the default limit. In this case one needs to change these default values:

```
>>dynare myModel
>>dynareFindSteadyState([5,1000])
```

It may be necessary to change the default range for allowable values for the model variables.

```
>>dynare myModel
>>dynareFindSteadyState(aRange(-1,1))
```

Finally, it may be necessary to provide appropriate range values for specific variables:

```
>>dynare myModel
>>myRangeSpec=setSomeValues('pie.*',1.0,1.2)+aRange(-1,10)
>>dynareFindSteadyState(myRangeSpec)
```

dynareFindSteadyState generates a collection of starting points to initiate a Newton algorithm for solving the model equations and applies Newton's method to each in turn.(See Section 4 for additional usage options.) **dynareFindSteadyState** allows the user to specify starting values or ranges of starting values for whatever values they choose while providing default initial values for the remainder.

2   Background

dynareFindSteadyState applies global optimization methods to the problem of generating plausible initial starting points for a Newton equation solver. It uses a MATLAB implementation of Direct. **dynareFindSteadyState** uses a modified version of Direct to return a collection of points that potentially minimize $norm(f(x))$. **dynareFindSteadyState**

applies Newton's algorithm to each of these points in turn. In the prototype, **dynareFind-SteadyState** reports success when it has found a convergent steady state that has a unique convergent solution.

To get an idea of the potential usefulness of **dynareFindSteadyState** I applied the tool to a collection of 55 dynare mod files collected from the dynare website and various dynare users at the board. Forty-three required no user intervention to compute a steady state.[1] The calculations typically took several seconds. A couple of models required about a minute. This includes time to run dynare's "check" program.

Several other models needed more global search time. For example after boosting the cpu constraint to 50 seconds **dynareFindSteadyState** was able to find solutions for sedmodel1, or0a by merely increasing the time and iteration constraints.

```
[theMod,xVals,fVals,rc,tm]=dynareFindSteadyState('sedmodel1',true,[50,10000]);
```

In the examples to this point, the **dynareFindSteadyState** tool has not used any user provided information about starting points. When provided with additonal information about the endogenous variables, **dynareFindSteadyState** can compute the steady states of the remaining two example models:

```
>> pgamma=0.7;pbeta=0.95;
forPort4=varRange('C',0.1,1)+varRange('D.',.1,.1)+...
varRange('DOTQ',0.01,1.5)+varRange('V',...
((0.1^(1-pgamma)/(1-pgamma))/(1- pbeta))-1,...
((0.1^(1-pgamma)/(1-pgamma))/(1- pbeta))+1)
[theMod,xVals,fVals,rc,tm]=...
dynareFindSteadyState('portfolio4_norm',forPort4);
```

In the worst case scenario, the user might have to provide starting values for each of the model variables. However, the **setSomeValues** tool makes this relatively easy by providing a simple region specification language employing regular expressions.

3 Future Directions

Many implementation and ease of use issues remain.

Multiple Solutions It's possible to change **dynareFindSteadyState** so that it returns multiple steady states when they exists.

Default Ranges/Times Experience with the tools may suggest a more useful set of defaults

---

[1] These included asset.mod, mod1a.mod, psd_exo3.mod, nnntest7.mod, mod1b.mod, q3a2.mod, dm7.mod, mod1c.mod, q3a50.mod, example1.mod, mod1.mod, t_lag2a.mod, example2.mod, mod2a.mod, ramst.mod, t_lag2b.mod, mod2b.mod, t_lag2_checka.mod, mod2c.mod, sw_euro.mod, t_lag2_check.mod, mod2.mod, test1.mod, t_lag2.mod, mod3a.mod, test2a.mod, t_periods_a.mod, test2.mod, t_periods.mod, lucas78.mod, test3.mod, t_sgu_ex1.mod, portfolio4.mod, test4.mod, variance_0.mod, test5.mod, portfolio.mod, fs2000.mod, m_0_3_0_0_0_0_0_0.mod, m_1_3_0_0_0_0_0_0.mod, judd.mod, judd_norm.mod and test6.mod

Newton method Improvements The code uses a very simple implementation of the Newton algorithm

User Controls Experience with tools may suggest useful quantity and amounts of output and other user controls

Variable Constraints It may prove worthwhile to use Direct's mechanism for implementing constraints among variables

Alternative Global Optimization Strategies Direct implements one of many possible rectangular region methods. Furthermore, it may be useful to explore other global optimization strategies.

Parallel Implementation The algorithm has many embarrassingly parallel components.

4 Installation and Usage

4.1 Installation

1. dynareFindSteadyState calls the program SPAMalg. Unless you already have a version of SPAMalg, you will also need to install the gensysToAMA programs by following the instructions provided with the gensysToAMA.zip file.

2. **unzip the files** into a directory(*someDir*) accessible by Matlab. This will create a directory, **dynareFindSteadyStateDist**, containing the **dynareFindSteadyState** programs and some example .mod files.

3. start **matlab**

4. place the dynareFindSteadyStateDist directory on the Matlab path using

   addpath *someDir*/dynareFindSteadyStateDist

   Be sure to use a fully qualified path name.

5. place the SPAMalg program on your path

   addpath *whateverDir*/gensysToAMADist.

   Be sure to use a fully qualified path name.

6. place a **version 4 dynare** on the Matlab path.[2]

7. during a matlab session, you can run a quick test of the installation:

   type

   >>isDynareFindSteadyStateOK

   to verify the **dynareFindSteadyState** program functions correctly. After a few seconds, you should get a **"SUCCESS"** message.

---

[2]dynareFindSteadyState should work with any version 4 dynare distributed after 4/10/2007. It may not work with earlier versions of dynare version 4. It definitely will not work with dynare version 3.

4.2 dynareFindSteadyState

```
                         dynareFindSteadyState
>> help dynareFindSteadyState
 function [xx,ff,rc]=dynareFindSteadyState(optional args)
 compute steady state generate initval assignments file(<modName>InitValFile)
 inputs(all args optional, order irrelevant)
 a logical value(true,false) determines the displayed
     amount of algorithm iteration info,
     default false, true to display copious amounts of iteration info
 a pair of doubles determines global search iteration limits
     [maxTimeAllowed itmax] default [10 100]
     max cpu secs for global search phase
     max number of iterations for global search phase
 anRSpec -- "global" search region specification
     type help regionSpec for explanation
 outputs
 xx -- approximation to steady state
 ff -- result of evaluating model eqns at approx steady state(should be zeroes)
 rc -- return code 1 for success 0 for failure
 file(<modName>InitValFile)  -- initval assignments to add to
   <modName>.mod file
   prints output of running dynare's check(ignores any steady state without
   a unique convergent solution)
```

4.2.1 Use "active" dynare model and program defaults

```
                        dynareFindSteadyState Example
>>dynareFindSteadyState
Total computing time : 00:00:00
fssInput object:
show func iterations?:false
max time:10
max iterations:100
anRSpec:default range: lower bound:  useDefault -- upper bound:  useDefault
specific variable range(s) if any:
model name:no model specified, use active model


the bounds
Y:  0    1
C:  0    1
K:  0    1
A:  0    1
H:  0    1
B:  0    1
Direct FYI: max fcn evals was binding constraint
113 init vals to try


EIGENVALUES:
         Modulus              Real         Imaginary
      1.184e-019       -1.184e-019                 0
<snip...>
There are 4 eigenvalue(s) larger than 1 in modulus
for 4 forward-looking variable(s)


The rank condition is verified.


ii=1 found start that converges,norm of f(x)=2.482534e-016
also passes unique stable solution test
creating InitValFile


ans =


example1


>>
```

4.2.2 Specify dynare model while using program defaults

This example assumes fs2000.mod is in the current working directory.

```
                  dynareFindSteadyState Example
   >> dynareFindSteadyState('fs2000')
 fssInput object:
 show func iterations?:false
 max time:10
 max iterations:100
 anRSpec:default range: lower bound:  useDefault -- upper bound:  useDefault
 specific variable range(s) if any:
 model name:fs2000
 <snip...>
 The rank condition is verified.


 ii=46 found start that converges,norm of f(x)=2.225990e-015
 also passes unique stable solution test
 creating InitValFile


 fs2000


 >>
```

4.2.3 Specify dynare model while changing default variable range

```
                    dynareFindSteadyState Example
>> dynareFindSteadyState('example1',aRange(-1,3))
fssInput object:
show func iterations?:false
max time:10
max iterations:100
anRSpec:default range: lower bound:  -1 -- upper bound:  3
specific variable range(s) if any:
model name:example1


Total computing time : 00:00:00
the bounds
Y:  -1    3
C:  -1    3
K:  -1    3
A:  -1    3
H:  -1    3
B:  -1    3
Direct FYI: max fcn evals was binding constraint
119 init vals to try


EIGENVALUES:
        Modulus              Real         Imaginary
<snip..>
```

4.2.4   Use "active" model while changing variable ranges

```
                    dynareFindSteadyState Example
>> dynareFindSteadyState(varRange('K',-1,3)+varRange('C',[0.5,7])+aRange(0.1,0.9))
fssInput object:
show func iterations?:false
max time:10
max iterations:100
anRSpec:default range: lower bound:  1.000000e-001 -- upper bound:  9.000000e-001
specific variable range(s) if any:
'K'  -- lower bound:  -1 -- upper bound:  3
'C'  -- lower bound:  5.000000e-001 -- upper bound:  7
model name:no model specified, use active model

the bounds
Y:  1.000000e-001    9.000000e-001
C:  5.000000e-001    7
K:  -1     3
A:  1.000000e-001    9.000000e-001
H:  1.000000e-001    9.000000e-001
B:  1.000000e-001    9.000000e-001
Direct FYI: max fcn evals was binding constraint
125 init vals to try
```

4.2.5   Use "active" model, more ouput different constraints

```
                    dynareFindSteadyState Example
>> dynareFindSteadyState(true,[1,10])
fssInput object:
show func iterations?:true
max time:1
max iterations:10
anRSpec:default range: lower bound:  useDefault -- upper bound:  useDefault
specific variable range(s) if any:
model name:no model specified, use active model

the bounds
Y:  O    1
C:  O    1
K:  O    1
A:  O    1
H:  O    1
B:  O    1
Direct FYI: max fcn evals was binding constraint
Iter:   1   f_min:    0.5657420694    fn evals:       13
13 init vals to try
time so far=          0.02,remaining timeat present rate=          0.19
```

4.3   genBounds

```
                            genBounds
>>help genBounds
 creates and displays matrix of variable bounds delimiting "global" search space for
  findSteadyState function
 inputs
   ()   -- with no args create matrix corresponding to
           default variable ranges for the ''active'' dynare model(M_.fname)
   (regionSpec)  -- create matrix corresponding to
           specified variable ranges for the active dynare model(M_.fname)
   (modName,regionSpec)  -- use dynare command to parse modName
           (making modName the ''active'' model)
           create matrix corresponding to
           specified variable ranges for the model
 outputs
 matrix of variable bounds delimiting "global" search space for
  use by the findSteadyState matlab function
```

4.3.1   view active model variable names and default constraints

```
                          genBounds Example
>> genBounds
active dynare model=example1
the bounds
Y:   0     1
C:   0     1
K:   0     1
A:   0     1
H:   0     1
B:   0     1
```

4.3.2 Specify active model and constraints

```
                          genBounds Example
>> genBounds('portfolio',aRange(-1,15)+varRange('Q.*',2,10))
Total computing time : 00:00:00
active dynare model=portfolio
the bounds
DOTQ:  2     10
Q1  :  2     10
Q2  :  2     10
X1  :  -1     15
X2  :  -1     15
C   :  -1     15
D1  :  -1     15
D2  :  -1     15
```

4.3.3 select "active" model use regular expressions .

```
                          genBounds Example
>> genBounds('portfolio',aRange(-1,15)+varRange('Q.*',2,10))
Total computing time : 00:00:00
active dynare model=portfolio
the bounds
DOTQ:  2     10
Q1  :  2     10
Q2  :  2     10
X1  :  -1     15
X2  :  -1     15
C   :  -1     15
D1  :  -1     15
D2  :  -1     15
```

4.4 setSomeValues

```
                          setSomeValues
>> help setSomeValues
  generate a regionSpec representing
  global seach region variable specifications
  all args optional(order determines scope of assignments)
  input any number of the following
     variable name (or regexp representing collection  of variable names)
           a scalarnumeric
     a scalar value, or range
     a cell array consisting of (names,  values and other cell arrays)
  output
     a regionSpec object characterizing a global search region
  examples:
     setSomeValues('xxx')   -> generates a rangeSpec with xxx at zero
     setSomeValues(99,'xxx')   -> generates a rangeSpec with xxx at 99
     setSomeValues([1 5],'xxx')
     setSomeValues('xxx',[1 5],'yyy','zzz')   ->
           generates a rangeSpec with xxx at 0 and yyy and zzz  between 1,5
     setSomeValues(99,'xxx',{33,'yyy','zzz'},'lll') ->
           generates a rangeSpec with xxx and lll at 99, yyy and zzz at 33
```

4.5 cleanupDynareDebris

```
                       cleanupDynareDebris
>> help cleanupDynareDebris
  cleanupDynareDebris()
  delete files created by dynare and dynareFindSteadyState commands
```

4.6 regionSpec

```
                            regionSpec
>> help regionSpec
 regionSpec object constructor
  regionSpec no args generates default region--vars on interval [0,1].
  regionSpec iterates sequential through its arguments left to right.
  regionSpec(varRange) changes region for specific variable
   type help varRange for more info
  regionSpec(aRange) changes the global "nonspecified" default region
   (ie regionSpec([-3,5]) or regionSpec(aRange(-3,5)))
```

4.7 varRange

```
                            varRange
>> help varRange
 varRange specify a range of values for a specific variable
 varRange no args creates a spec that matches any variable name and
 varRange(aRegExp), creates a spec that a associates
  a [0 1] acceptable range with each variable matching aRegExp.
  (ie varRange('XX_.*'))
 varRange(aRegExp,aRange), creates a spec that a associates
  aRange as the acceptable range for each variable matching aRegExp.
  (ie varRange('XX_.*',[-1,1]),varRange('XX_.*',aRange(-1,1)))
 varRange(aRegExp,lower,upper), creates a spec that a associates
  [lower,upper] as the acceptable range for each variable matching aRegExp.
  (ie varRange('XX_.*',-1,1)
  matches regular expression
 type help regexp for more information on regular expressions
```

4.8 aRange

```
                            aRange
>> help aRange
 aRange class constructor
  constructs an object charactering a range(closed intervals) of values
    inputs
    ()  -- range corresponding to a (0,1)
    (a)  -- range corresponding to a (a,a)
    (a,b)  -- range corresponding to (a,b)
     takes bnkVal's which allow for 'useDefault' in addition to double
 defaultRange=[bndVal(0) bndVal(1)];
```