Lecture Notes on Shooting for the Global Nonlinear Saddle Path

Manoj Atolia        Edward F. Buffie

November 18, 2006

Contents

1 Introduction 3

2 The Neo-Classical Growth Model 5

3 Reverse Shooting with 2 State Variables 9
   3.1 The Minimum Distance Mapping 16
   3.2 Implementation Procedure: Reverse Shooting Made Easy 19
      3.2.1 “Slow” But Very Easy 20
      3.2.2 Much Faster But Less Easy 22
   3.3 A Primer on Using Reverse Shoot Easy 24
      3.3.1 Interpreting Output of Reverse Shoot Easy 28
   3.4 Using Reverse Shoot Fast 31
   3.5 Complex Eigenvalues and Oscillatory Dynamics 32

4 Reverse Shooting with 3 State Variables 33
   4.1 The Minimum Distance Mapping 33
   4.2 The Circle-Search Algorithm 37
   4.3 Implementation Procedure: Reverse Shoot 3D 39
   4.4 Further Remarks 40
   4.5 An Example with 3 State Variables 41
   4.6 Using Reverse Shoot 3D 42
      4.6.1 Interpreting the Output of Reverse Shoot 3D 43
   4.7 A Picture of the Minimum Distance Mapping 46

5 Reverse Shooting with 4 State Variables 48
   5.1 Searching with Spheres 49
   5.2 An Example with 4 State Variables 50
   5.3 Implementation Procedure: Reverse Shoot 4D 54
   5.4 Using Reverse Shoot 4D 54
      5.4.1 Interpreting Output of Reverse Shoot 4D 55
   5.5 Limitations of Reverse Shooting 56
6 Forward Shooting with the Minimum Distance Mapping .................................................. 56
6.1 Forward Shooting versus Reverse Shooting .............................................................. 59
6.2 Implementation Procedure ......................................................................................... 60
6.3 Solving Systems with 2 Jump variables ....................................................................... 62
6.4 Solving Systems with 3 Jump variables ....................................................................... 62

7 Forward Shooting with Anticipated Shock .................................................................... 62

8 Forward Shooting with Temporary Shock ...................................................................... 64

9 Forward Shooting for Unit Root Problems ................................................................... 66

10 Unit Root and Anticipated or Temporary Shock............................................................. 68

List of Figures

1 Phase diagram of the neoclassical growth model. ......................................................... 6
2 Need for accuracy in finding $c^*$. ............................................................................... 8
3 Transition paths for the time-reversed system. ............................................................... 9
4 Stable Manifold with saddle paths and $\varepsilon$-ring. ....................................................... 11
5 Searching for intersection of the desired trajectory with $\varepsilon$-ring. ............................ 12
6 Using exponential tracking to check accuracy of the computed path. ....................... 13
7 Source of ill-conditioning of the minimum distance mapping. .................................. 16
8 The minimum distance mapping. .................................................................................. 17
9 Failure to find a solution due to lack of sufficient precision. ...................................... 21
10 The idea behind Reverse Shoot Fast. ........................................................................... 23
11 Overview of the Reverse Shoot Easy. ......................................................................... 24
12 Initialization section of Reverse Shoot Easy. .............................................................. 26
13 Calibration and Steady State section of Reverse Shoot Easy. ................................. 27
14 Entering the dynamic system in Reverse Shoot Easy. .................................................. 28
15 The Initialization cell in Reverse Shoot Easy. .............................................................. 29
16 Initial Output from Reverse Shoot Easy. ................................................................ .... 29
17 Output as Reverse Shoot Easy enters Step 2. ............................................................. 30
18 Output as Reverse Shoot Easy enters Step 3. ............................................................. 31
19 Output as Reverse Shoot Easy as current candidate solution fails the accuracy test .... 32
20 Checking accuracy of the new candidate solution. ..................................................... 33
21 Successful completion of search after meeting accuracy check for the new candidate solution. ................................................................................................................. 34
22 Initialization cell for Reverse Shoot Fast ...................................................................... 34
23 Initial output from Reverse Shoot Fast. ...................................................................... 35
24 The Rosenbrock Curve. ............................................................................................... 35
25 The nature of distance mapping in neighborhood of RC curve. .................................. 36
26 The failure of the typical simplex algorithm. ............................................................... 37
27 The Circle Search Algorithm. ....................................................................................... 38
28 Why $T$ matters. .......................................................................................................... 39
29 Entering the dynamic system into the program. .......................................................... 42
1 Introduction

These lecture notes discuss a very general technique for solving for global nonlinear saddle path that can be used by an economist without special knowledge of numerical techniques and without making use of problem-specific information/intuition.

While the user can write his own program to implement this algorithm in any higher level programming language environment such as Mathematica, Matlab etc., we provide Plug and Play programs (written in Mathematica) that expect the user to only specify details relating to his/her particular problem and not worry about the nitty-gritty of the solution technique every time a new model has to be solved. These programs are available at http://mailer.fsu.edu/~matolia.

Despite using these Plug and Play programs, the user may be interested in the idea that underlies the technique used by these programs. For such an interested user, these lectures notes provide the requisite information. They also contain a discussion on how to use the programs beyond what is already outlined in the programs themselves which may be useful in certain situations.

The technique is fairly general and allows one to solve for a large class of dynamic economic systems that arise in economics as a result of intertemporal optimization. Specifically, the method can be used to solve following types of problems:

1. problems with $2 - 4$ state variables and permanent shock.\(^1\)

2. problems with $2 - 3$ jump variables and anticipated shock.

3. problems with $2 - 3$ jump variables and temporary shock.

---

\(^1\)While technique extends at least to systems with 5 state variables, the current algorithm would be computationally infeasible.
4. problems with $2 - 3$ jump variables and a unit root.

5. problems with $2 - 3$ jump variables, anticipated shock, and a unit root.

6. problems with $2 - 3$ jump variables, temporary shock, and a unit root.

All problems other than those with permanent shock require the use of forward shooting as reverse shooting fails when:

- the model has a unit root,
- there is a temporary shock to the model, or
- there is an anticipated shock to the model.

In some rare cases, it switching to forward shooting may save time even when the problem can be solved using reverse shooting.

The lectures are organized as follows. Section 2 introduces the idea of forward and reverse shooting in context of a the neoclassical growth model and provides a preview of the types of problems that may arise in higher order systems. Section 3 describes reverse shooting in models two state variables. Section 4 describes reverse shooting in model with three state variables. These two sections layout the key insights and technical contribution that underpin the shooting approach put forward in these lecture notes. It also shows that as algorithm scarcely relies on problem specific information or intuition, it is possible to write completely automated programs that can be expected to always work. Section 5 describes how the technique naturally extends to systems with 4 state variables.

Section 6 shifts focus and show how the technique naturally extends from reverse shooting to forward shooting. It discusses solution technique for systems with $2 - 3$ jump variables. In Section 7, we extend the idea to the models with anticipated shock which sometimes may need use of homotopy. Section 8, contains the extension to models with temporary shock. Such models necessarily require a homotopic approach to solution. In Section 9, we turn our attention to a large class of problems for which currently there does not exist a method to solve for the global nonlinear saddlepath: problems with a unit root. Standard shooting approach and projection methods fails as one needs to simultaneously solve for the transition and the final steady. With new steady state not known in advance reverse shooting method of Section 3 and 4 does not work either. Unit root
problems abound in realm of international economics where one many times considers a small open
economy operating in the perfect capital market. Section 10 combines a unit root problem with an
anticipated shock. Section 11 combines unit root with temporary shock. Section 12 concludes.

2 The Neo-Classical Growth Model

Consider the following neoclassical (optimal) growth problem:

\[ \max_c \int_0^\infty \frac{c^{1-\frac{1}{\tau}}}{1-\frac{1}{\tau}} e^{-\rho t} dt \]

subject to

\[ \dot{k} = Y(k) - \delta k - c \]

\[ k(0) = k_0. \]

To solve the problem, form the Hamiltonian

\[ H = e^{-\rho t} \left[ \frac{c^{1-\frac{1}{\tau}}}{1-\frac{1}{\tau}} + \lambda_1 (Y(k) - \delta k - c) \right]. \]

The first-order condition for control variable \( c \) is

\[ c^{-\frac{1}{\tau}} = \lambda_1, \]

and the co-state equation for the state variable \( k \) is

\[ \frac{d}{dt} (\lambda_1 e^{-\rho t}) = -\frac{\partial H}{\partial k}, \]

which implies

\[ \lambda_1 = (\rho + \delta - Y_k(k)) \lambda_1. \]

Differentiating the first-order condition and eliminating \( \lambda_1 \) from the resulting equation using the
co-state equation gives the Euler equation for consumption

\[ \dot{c} = \tau (Y'(k) - \delta - \rho) c. \]
Figure 1: Phase diagram of the neoclassical growth model.

The optimal growth problem, therefore, gives rise to the following dynamic system.

\[
\dot{c} = \tau (Y'(k) - \delta - \rho) c
\]
\[
\dot{k} = Y(k) - \delta k - c
\]
given \( k(0) = k_o \) and \( \lim_{t \to \infty} k(t) = k_f \).

\[
TVC: \lim_{t \to \infty} e^{-\rho t} k u'(c) = 0.
\]

The phase diagram for the neoclassical is familiar and shown in Figure 1. It is easy to show that any transition path other than that lies on the stable manifold SS ultimately leads to the violation of the transversality condition, \( TVC \) or the first-order conditions of the problem.\(^2\) In essence, therefore, \( TVC \) implies the boundary condition:

\[
\lim_{t \to \infty} k(t) = k^*.
\]

Thus, for a quantitative solution, we now have the following problem to solve

\[
\dot{c} = \tau (Y'(k) - \delta - \rho) c
\]
\[
\dot{k} = Y(k) - \delta k - c
\]
given \( k(0) = k_o \) and \( \lim_{t \to \infty} k(t) = k_f \),

\(^2\) In passing it may be mentioned that different paths cannot cross as then one set of values of \((c,k)\) would imply two values of \(\dot{c}/\dot{k}\) which is not possible.
which is a two point boundary value problem with one asymptotic boundary condition as \( t \to \infty \).

A shooting algorithm converts this boundary value problem into a sequence of initial value problems with an updating rule linking the transition from one initial value problem to the next one. In context of the neoclassical growth model, consider the following initial value problem:

\[
\dot{c} = \tau \left( Y'(k) - \delta - \rho \right) c \\
\dot{k} = Y(k) - \delta k - c
\]

given \( k(0) = k_o \) and \( c(0) = c^g \),

where \( c^g \) is a guess value for \( c^* \). It is clear from the phase diagram in Figure 1 that if we chose correct value \( c^* \) then asymptotic boundary condition on \( k \) as \( t \to \infty \) will be satisfied.

Having solved the problem for \( c^g \) it is easy to see how the next value should be chosen. If while solving the system, \( k \) turns negative, we know that \( c^g > c^* \) and we choose a value less than \( c^g \). If on the other hand \( c \) turns negative then we choose a new guess value to be larger than \( c^g \). Thus, we have a straightforward updating rule that can be used to update guesses for \( c^* \) and solve the original two point boundary value problem. This basic idea of using shooting to convert an boundary value problem to a sequence of initial value problems can be found in any elementary text on numerical analysis.

However, a economic system is only saddlepath-stable and not globally stable. What that means for shooting algorithm is that small errors in guess value \( c^g \) relative to true value \( c^* \) lead to large errors along transition path. Thus, one needs to estimate \( c^* \) with a very high accuracy to be able to remain close enough to the saddle path for sufficiently long time and hence solve for the transition path with desired accuracy (see Figure 2). This point which right now appears to be a minor irritation becomes as major issue when we go to higher order systems.

It also suggests that another shooting strategy, that has no advantage for typical applications in physics or engineering, will be more fruitful for the economic problems. To distinguish this new strategy, let us call the shooting technique described above as forward shooting because we guess the initial value of \( c \) and start close to the initial point on the transition path. In the alterante method called reverse shooting, we start close to the final point on the transition path \((c_f, k_f)\) and then use the time-reversed system to get back to the initial point of the transition path or the saddlepath of the original system. The transition paths for the time-reversed system corresponding those for original system in Figure 2 are shown in Figure 3. Obviously, the only difference is the
direction in which time flows. However notice that if solve the time reverse system\(^3\)

\[
\begin{align*}
\dot{c} &= -[\tau (Y'(k) - \delta - \rho) c], \\
\dot{k} &= -[z_0 Y(k) - \delta k - c],
\end{align*}
\]

with initial conditions\(^4\)

\[
k(0) = k_f - \varepsilon \quad \text{and} \quad c(0) = c_f,
\]

then the corresponding solution of the time-reversed system will remain close to the saddle path for all time. Thus, reverse shooting does not suffer from the ill-conditioning which plagued forward shooting algorithm. Furthermore, in reverse shooting we need to just solve one initial value problem and not a sequence of problems indexed by the guess values \(c^g\). That reverse shooting reduces the dimensionality of the search process is a general result that is valid for higher order systems.

The reason that reverse shooting dominates forward shooting for economic problems is that reversing time converts \(SS\) from the stable manifold (and hence a strong repeller) of the original system to the unstable manifold (and hence a strong attractor) of the time-reversed system. In fact, for a class of nonlinear systems, there is a result in mathematics literature called ‘exponential tracking’ which states that the deviations of the path found by solving the time-reversed system

\(^3\)Note that to produce the time-reversed system all that is needed is to put a negative sign in front of the right hand side of the differential equations in the system.

\(^4\)If \(k_o > k_f\), we would choose \(k(0) = k_f + \varepsilon\).
from the unstable manifold die down at an exponential rate. This is nothing but a generalization
of the result for the linear systems.\textsuperscript{5} This important property underpins the approach to shooting
developed here.

3 Reverse Shooting with 2 State Variables

Consider following modification to the neoclassical growth model where we introduce money in the
utility function following Sidrauski (1967) and the economy has an exogenous rise in productivity.
So, the representative agent faces the following problem:

\[
\max_{c,m,k} \int_0^\infty \left( \frac{c^{1-\frac{1}{n}}}{1-\frac{1}{n}} + \frac{g^{m^{1-\frac{1}{n}}}}{1-\frac{1}{n}} \right) e^{-\rho t} dt
\]

\textsuperscript{5}To a graduate student in economics this should not be at all surprising. Any discussion of the phase diagram of
neoclassical growth model invariably contains a statement about how small deviations of guess value \(c^g\) from true
value \(c^*\) generates path that quickly and rapidly move away from the saddlepath and the stable manifold. This is
nothing by exponential tracking in action but with time flowing in opposite direction.

For those who are familiar with solving linear systems that are saddlepath-stable, it is easy to see that errors in
guess value of \(c\) result in non-zero coefficients for the terms involving positive eigenvalues and hence to an exponential
explosion away from the stable manifold.
subject to

\[
\begin{align*}
A &= m + b + k, \\
\dot{A} &= Y(k, z) + rb + L - \delta k - c - \pi m, \\
A(0) &= m_o + k_o + b_o.
\end{align*}
\]

The path of productivity is

\[
\begin{align*}
\dot{z} &= \omega (z_f - z), \quad \omega > 0, \\
z(0) &= z_o, \quad z_f > z_o.
\end{align*}
\]

and government’s budget constraint is given by

\[
\dot{m} + \dot{b} = rb + L - \pi m.
\]

From agent’s first order conditions and the co-state equations, one can explicitly solve for \( r \) and \( \pi \) and obtain the Euler equation

\[
\begin{align*}
\dot{r} &= Y_k(k, z) - \delta \equiv r(k, z), \\
\dot{\pi} &= g\left(\frac{m}{c}\right) \frac{\delta}{\tau} - \pi(c, k, z) \equiv \pi(m, c, k, z), \\
\dot{\tau} &= \tau c(r - \rho).
\end{align*}
\]

Assuming stock of bonds is fixed at \( b_0 \), then from government’s budget constraint and agent’s budget constraints one gets

\[
\begin{align*}
\dot{m} &= L + rb - \pi m, \\
\dot{k} &= Y(k, z) - c - \delta k.
\end{align*}
\]

The evolution of the economy is given by the system consisting of \((c, k, z)\) whose evolutions is given by

\[
\begin{align*}
\dot{c} &= \tau c(r(k, z) - \rho), \\
\dot{k} &= Y(k, z) - c - \delta k, \\
\dot{z} &= \omega(z_f - z).
\end{align*}
\]
and the boundary conditions for the system are

\[ z(0) = z_o < z_f, \quad \text{and} \quad \lim_{t \to \infty} z(t) = z_f, \]
\[ k(0) = k_o < k_f, \quad \text{and} \quad \lim_{t \to \infty} k(t) = k_f. \]

Note that while \( k \) was a control variable for the agent’s problem, it becomes a state variable in the general equilibrium: the economy as a whole can accumulate capital only over time by consuming less than the net output. It may also be mentioned that in the current setup it is possible to exclude \( m \) from the system as the model exhibits classical dichotomy.\(^6\)

To illustrate the idea of reverse shooting with two state variables, let us first consider the stable manifold defined by policy function \( c(k, z) \) as shown in Figure 4. For concreteness, let us assume that top of the surface corresponds to the final steady state \( (k_f, z_f) \). If we form a ring of sufficiently small radius \( \varepsilon \) around the final steady state on this stable manifold (see Figure 4), then the desired global nonlinear saddlepath which a curve on the stable manifold will intersect this \( \varepsilon \)-ring at some point.\(^7\) Thus, solving for the saddlepath boils down to finding its intersection with this \( \varepsilon \)-ring. For if we knew this point, all one needs to do is to solve the time-reversed system with the values of \( (c, k, z) \) given by this intersection as the initial conditions.

\(^6\) An astute reader may have noted that since path of \( z \) is exogenous, this problem is essentially same as the previous problem and can be solved using forward shooting by guessing the value of \( c(0) \). (Can we use reverse shooting in one dimension as discussed earlier to solve this problem?) However, we are using this simplified setup to illustrate the idea behind reverse shooting in two dimensions that will apply to a general two state variable problem where there will also be two ‘jump’ or control variables.

\(^7\) Note that unlike the case with one state variable, now saddle path and stable manifold are distinct objects with saddle path being a one-dimensional curve and stable manifold being a two-dimensional surface in three dimensions.
To illustrate the idea of finding this intersection, let us put this $\varepsilon$-ring around the final steady state $(k_f, z_f)$ in the state space as shown in Figure 5.

The projections of various saddlepaths on stable manifold onto the state space (henceforth called trajectories) will intersect the ring. We need to find the intersection of the desired trajectory with the ring. To do so, choose $n$ consecutive points $p_i$, $i = 1, 2, ..., n$, on the ring as initial conditions and solve for the corresponding trajectories by reverse shooting (i.e. by solving the time-reversed system). Then there exists a trajectory with initial conditions corresponding to the point $p_j$ that comes closest to the initial steady steady state. As trajectories do not cross in the state-space, the desired trajectory must cross the ring on the arc $p_{j-1}p_jp_{j+1}$. One can now use the bisection method to obtain a better approximation to the desired trajectory. For example, if $q$ and $r$ are the midpoints of the arcs $p_{j-1}p_j$ and $p_jp_{j+1}$, then one of the three trajectories starting at $q$, $p_j$ and $r$ will come nearest to the initial steady state. This narrows the search to arc $p_{j-1}p_j$, arc $qr$, or arc $p_jp_{j+1}$. The updating process is continued until the distance of the trajectory currently closest to the steady state is within the desired tolerance.

Solving the time-reversed system requires starting values not only of the state variables $(k, z)$ which we know once we choose a point on the $\varepsilon$-ring but also of the jump variable $c$. Since, we
do not the know the stable manifold we do not the corresponding value of $c$. Now, suppose as in the neoclassical model, we put $c(0) = c_f$ while solving the time reversed system. Let $p^*$ be the intersection of the trajectory that meets the tolerance criterion with the $\varepsilon$-ring. What’s the implication of having used $c(0) = c_f$ for the accuracy of the corresponding computed ‘saddlepath’?

A look back at Figure 3 provides the answer. The computed saddlepath will be above or below the stable manifold. However, Once again as the stable manifold of the original system which contains the saddlepath becomes the unstable manifold for the time reversed system, these deviations away from the stable manifold will decay rapidly due to exponential tracking as shown in Figure 3. So, once the exponential tracking had some time to work, the remaining part of the solution will very accurately track the true nonlinear saddlepath.

How does one know when exponential tracking has had enough time to work? Once again intuition from the neoclassical growth model provides the answer. Figure 6 shows how the accuracy check works in the neoclassical growth model. Trajectories for the time-reversed system hit $k_o$, but there is error in the path for the jump variable $c$. Consider $k = k_f - \varepsilon_1$ which is analogous to a larger $\varepsilon$-ring in systems with two state variables. The candidate solution $AJ$ is not accurate. This becomes apparent when it is compared with the candidate solution for a smaller with radius $\varepsilon_2$. Reducing $\varepsilon$ forces the path to move closer to the stable manifold. Since the corridor formed by

Figure 6: Using exponential tracking to check accuracy of the computed path.
AJ and VF is wide, the new candidate solution BN looks significantly different. If the candidate solution for \( \varepsilon_1 \) were accurate, path AJ would be extremely close to the true solution VF. Path BN would be even closer but indistinguishable from path AJ. It is clear that the accuracy test will ratify candidate solutions close to the stable manifold. But might two paths be close to each other yet far away from the stable manifold? No. If the trajectory associated with \( \varepsilon_2 \) had the shape shown by the dashed line, then the error in \( c \) would be larger at \( k_f-\varepsilon_1 \) than at \( k_f-\varepsilon_2 \). Exponential tracking guarantees, however, that the error decreases exponentially in the time-reversed system. Paths AJ and BN cannot be close to each other unless both are close to path VF.

The trapping argument in previous paragraph is special to the neoclassical growth model. It is, however, not too hard to see that exponential tracking guarantees that paths with two different values of \( \varepsilon \) will only be close to each other if they are also close to the stable manifold and the true nonlinear saddlepath. In the present example, we have only one control/jump variable. Will it matter if we had more than one jump variable? No.

For concreteness, let us augment the model with adjustment costs in capital accumulation which will result in system with two state and two jump variables. In this case the agent’s problem becomes

\[
\max_{c,m,i,b} \int_0^\infty \left( \frac{c^{1-\frac{1}{2}}}{1-\frac{1}{2}} + \frac{g m^{1-\frac{1}{2}}}{1-\frac{1}{2}} \right) e^{-\rho t} dt
\]

subject to

\[
A = m + b,
\]

\[
\dot{A} = Y(k,z) + rb + L - c - I - \psi \left( \frac{I}{k} \right) k - \pi m,
\]

\[
\dot{k} = I - \delta k
\]

\[
A(0) = m_o + b_o \quad \text{and} \quad k(0) = k_o,
\]

where \( \psi(x) = n (x - \delta)^2 /2 \) is the quadratic adjustment cost function. The productivity process and the government’s budget constrain remain unchanged.

From agent’s problem, as before, one obtains

\[
g \left( m \right) = \pi + r,
\]

\[
\dot{c} = \tau c (r - \rho),
\]

and in addition, the first order condition for investment on some manipulations can be solved
explicitly for investment as function of \( c, k, \) and \( z \)

\[ I \equiv I (c, k, z), \]

When the first order conditions are now manipulated one gets

\[ r \equiv r (m, c, k, z), \]
\[ \pi = \pi (m, c, k, z). \]

Note that now \( r \) is now a function of \( m, c, k, \) and \( z \) and not just of \( k \) and \( z \) as in case of the model without adjustment costs. It, therefore, becomes necessary to include \( m \) in the core dynamic system.

Assuming stock of bonds is fixed at \( b_0 \), then from government’s budget constraint as before one has

\[ \dot{m} = L + rb - \pi m, \]

and the evolution of the economy is given by

\[ \dot{m} = L + r (m, c, k, z) b - \pi (m, c, k, z) m, \]
\[ \dot{c} = \tau c (r (m, c, k, z) - \rho), \]
\[ \dot{k} = I (c, k, z) - \delta k, \]
\[ \dot{z} = \omega (z_f - z), \]

and the boundary conditions for the system are

\[ z (0) = z_o < z_f \quad \text{and} \quad \lim_{t \to \infty} z (t) = z_f, \]
\[ k (0) = k_o < k_f \quad \text{and} \quad \lim_{t \to \infty} k (t) = k_f. \]

Thus, now the system has two jump variables \( c \) and \( m \). When applying the two state variable algorithm for searching for the intersection of the desired trajectory in the state space with the \( \varepsilon \)-ring, one sets both \( c (0) = c_f \) and \( m (0) = m_f \) for the time-reversed system. Once again, exponential tracking ensures that the corresponding solutions for \( c \) and \( m \) gravitate rapidly toward the corresponding stable manifold. Therefore, once again, one can check for the accuracy of the computed solutions by comparing the solutions for the two \( \varepsilon \)-rings.
3.1 The Minimum Distance Mapping

The preceding argument for searching for the intersection of the desired trajectory with the $\varepsilon$-ring can be cast in more formal terms. What we did implicitly was to define a mapping from the starting point of a candidate trajectory on the $\varepsilon$-ring to the minimum distance of the trajectory from the initial steady state. The fact that trajectories cannot cross in state space implied that the mapping has unique global minimum. By definition, the minimum is zero. Given that the search on the $\varepsilon$-ring is one dimensional, the bracketing algorithm locates the unique global minimum of this mapping. But though the search strategy is simple, the search process is not. The complicating factor is that the distance mapping is extremely ill-behaved. We elaborate on this and on the computational issues it gives rise to next.

We claim that primitive bracketing (i.e., bisection) is the only efficient search strategy. Conventional algorithms fail because the minimum distance mapping is highly ill-conditioned. In a system with two state variables, the saddle point solution for the linearized system is controlled by the system’s two negative eigenvalues

$$
\begin{bmatrix}
  z_1(t) - z_1^* \\
  z_2(t) - z_2^*
\end{bmatrix} = \begin{bmatrix}
  X_{31} \\
  X_{41}
\end{bmatrix} h_1 e^{\lambda_1 t} + \begin{bmatrix}
  X_{32} \\
  X_{42}
\end{bmatrix} h_2 e^{\lambda_2 t},
$$

where let $\lambda_1$ be the dominant (negative) eigenvalue so that $\lambda_2 < \lambda_1 < 0$ and $h_1$ and $h_2$ are constants.
determined by initial conditions. If the eigenvalues are real, the relative weight of the term involving
the smallest eigenvalue decreases exponentially. Figure 7 depicts the problem that arises as a result.

As time passes, the trajectories formed by \{z_1, z_2\} in the state space gravitate toward the dominant
eigenvector ray \{X_{31}, X_{41}\} which is the projection of the eigenvector associated with the largest
negative eigenvalue (i.e., the smallest in absolute value). Thus, on the \(\varepsilon\)-ring, the trajectories are
very densely packed around the two points where the ring intersects the DER. This densely packed
cluster includes a vast number of trajectories that start far away from the initial steady state.

Consequently, as the search moves along the \(\varepsilon\)-ring, minimum distance of the candidate trajectory
from the initial steady state remains large until the search reaches a critical point very close to the
true solution. This is point \(p_3\) in Figure 8.

![Figure 8: The minimum distance mapping.](image)

After the search passes \(p_3\), minimum distance plunges in a sheer, almost vertical drop to the
global minimum of zero. Thus the minimum distance mapping is extremely flat everywhere except
at a tiny “hole” that marks the location of an extraordinarily deep, narrow \(V\). At the bottom of
the hole, the mapping is nondifferentiable as minimum distance rises very rapidly on either side \(p^*\).
The \(V\)-shaped kink at this single point precludes the use of differentiable methods.

The nature of ill-conditioning of the mapping remains much the same for the nonlinear system.
Despite the ill-conditioning of mapping, the uniqueness of its minimum allows one to systematically
find improvement to zero in on the desired trajectory. The practical implication of ill-conditioning
is that often it will be necessary to slice the \( \varepsilon \)-ring very finely. Exactly how fine depends on the value of \( \varepsilon \) and the tolerance criterion that determines how close the acceptable trajectory has to get to the initial steady state. In examples we have solved, to achieve desired accuracy, it was not unusual to split the \( \varepsilon \)-ring into \( 10^8 \)-\( 10^{30} \) parts. Because the ring had to be partitioned so finely and because numerical error accumulates in the differential equation solver, it was necessary to compute the trajectories with high precision: for some runs, 24-digit precision was sufficient; but stiff systems, where the ratio of the eigenvalues was eight or more, required 32-40 digit precision. (A few cases needed 40+ digits.)

Computational complexity is also affected by the presence of endogenous variables in the differential equations for the core dynamic system. In general, a dynamic GE model will produce a system of the type

\[
\begin{align*}
\dot{z} &= f(z, x, w), \\
\dot{x} &= g(z, x, w), \\
h(z, x, w) &= 0,
\end{align*}
\]

where

\[
\begin{align*}
z &= \text{vector of state variables}, \\
x &= \text{vector of jump variables}, \\
w &= \text{vector of endogenous variables}.
\end{align*}
\]

In both examples we have discussed, it was possible to analytically/explicitly solve \( h(z, x, w) = 0 \) for the endogenous variables \( w \) that appeared in the system in terms of the variables in the system. In general, it will not be possible to do so in which case there are two ways to handle these variables. One option is to differentiate \( h(z, x, w) = 0 \) with respect to time and then solve the expanded system with \( w + z \) control variables. This method introduces additional numerical error, however, when the \( \dot{w} \) equations are integrated backward to solve the system. The other option is to employ a nonlinear equation solver to numerically compute \( w = s(z, x) \) at each step when solving the core dynamic system. The advantage of this method is that numerical error does not accumulate in the differential equation solver. Its drawback is that computation time increases when the solution to a

---

8This refers to how close comparison trajectories have to be to locate a solution with the desired accuracy, not to the number of trajectories computed. When information about the dominant eigenvector ray is not used to guide the search, the maximum number of trajectories that has to be computed is \( 2\log_4 10^8 - 2\log_4 10^{30} = 53 - 199 \).
system of nonlinear equations is required as an input at each step for the solution of the system of differential equations. This is a potentially serious problem when computation entails high precision and a small step size. We recommend expanding the system which is simple and mechanical though a tedious process; hopefully, future versions of Mathematica will eliminate the need for them by solving mixed systems of differential and algebraic equations at high precision.\textsuperscript{9}

3.2 Implementation Procedure: Reverse Shooting Made Easy

Despite the ill-conditioning of the minimum distance mapping, the algorithm for finding the global nonlinear saddlepath naturally lends itself to automation. One reason, we are able to do so, is the choice of mapping for reverse shooting. Unlike the mapping suggested in literature (see Judd, 1999), the minimum distance mapping is well defined over entire domain. Further, the choice of mapping also clarifies the underlying ill-conditioning of the reverse shooting problem which plagues not only the minimum distance mapping but also the mapping suggested in the literature.\textsuperscript{10} This implies that precision in calculations becomes an important issue that needs attention for automating the search for the global nonlinear saddlepath.

The programs \textit{Reverse Shoot Easy} and \textit{Reverse Shoot Fast} that solve two state variable system by reverse shooting may be downloaded from \url{http://mailer.fsu.edu/~matolia/}. Below we describe the general structure and implementation procedure of the programs and how to use them.

\textsuperscript{9}Mathematica 5.2 solves mixed systems only for machine precision, not at higher levels of precision. Users should also know that the program does not really solve a mixed system; it differentiates the algebraic equations and then solves an expanded system of differential equations.

\textsuperscript{10}The mapping suggested in literature is between the initial guess for \( x(T) \) (for a suitably large \( T \)) and the resulting value of \( z(0) \) i.e. \( z(0) = F(x(T)) \). The problem of solving the system then reduces to solving the equation \( F(x(T)) - z_o = 0 \). It is tacitly assumed/hoped in the literature that this mapping is well defined for all likely guesses of \( x(T) \) and it is differentiable whereas, in fact, neither is true.

For example, while comparing the algorithm with the forward-shooting method, Judd states that “This is more likely to work, since small changes in \( x(T) \) will, hopefully, generate only small changes in the implied value of \( z(0) \)” (p.361, emphasis is ours and we have changed notation). The reason this is not the case relates to the fact that in state space all trajectories tend to gravitate toward the dominant eigenvector ray. Reverse shooting only ensures that the solution path for a given value of \( x(T) \) quickly gravitates toward the unstable manifold of the original system. But its projection onto the state space (called a trajectory) will also be close to the dominant eigenvector ray. When \( x(T) \) is perturbed a little bit. Even if it causes a small perturbation of trajectory in neighborhood of the dominant eigenvector ray, the trajectory generated by this perturbation can be quite distant from the trajectory generated by the original value \( x(T) \) as trajectories spanning the entire state space all cluster around the dominant eigenvector ray.

The reason that it is not well defined relates to the fact many times for a given value of \( x(T) \) the corresponding trajectory (solution path) may leave the positive orthant before the time-reversed system can be solved back till \( t = 0 \).
3.2.1 “Slow” But Very Easy

Potential users should first try Reverse Shoot Easy. The program does not require any intelligent human input, and in the great majority of cases it computes the solution in 5-30 seconds. The user has to only take care of the following preliminary steps:

- Buy and install Mathematica 5.2 or higher.
- Type in the equations of the model and then sit, relax, and watch the program solve for the global nonlinear saddlepath. This is a straightforward business when $h(z, x, w) = 0$ gives explicit solutions $w = s(z, x)$ that link the paths of the endogenous variables $w$ to the paths of $z$ and $x$. Otherwise, the user should artificially expand the number of control/jump variables in the system as mentioned earlier.

After entering the equations of the model and checking for typos, the user can relax. Everything in the next three “steps” is done by the computer in a series of loops.

Step 1 (Get a candidate solution)

- Evaluate the Main Cell. If the first pass through the main cell does not return a solution, then increase precision by four digits and re-execute the program.

The program fails to generate a solution only when computations are not executed with sufficient precision. Consider the situation in Figure 9.

The updating algorithm has narrowed the search to the segment $ab$ on the $\varepsilon$-ring. One more bisection will locate the solution – the trajectory that intersects the ring at point $c$. But point $c$ is inaccessible if $a - c$ is smaller than working precision for internal computations: when the computer tries to assign a number to point $c$, rounding error prevents it from doing so. The updating algorithm stops updating; recognizing the problem, the program aborts the current search for the candidate solution and starts a new search with precision increased by four digits. This process continues until a candidate solution is produced.

Although the solution obtained in Step 1 meets the tolerance criterion, it is not necessarily the correct solution. The next two steps check that the candidate solution is numerically precise (not distorted by rounding error) and accurate (sufficiently close to the stable manifold).
Figure 9: Failure to find a solution due to lack of sufficient precision.

Step 2 (Check numerical precision of the candidate solution)

- Proceed to Error Checking. Compute a new solution with precision increased by four digits. If the new solution differs significantly from the current candidate solution, then add another four digits to precision and re-run the program. Continue until successive runs yield the same solution.

There is a simple way to determine whether rounding error has distorted the solution: re-run the program with a higher value for working precision; if accumulated rounding error is a problem, then the solution will change noticeably. The program stores the solutions from successive runs and samples a large number of points in the time domain. If the change in any system variable ($x_i$ or $z_i$) exceeds $10^{-5}$ at any sample point, the program increases precision. Here it is assumed that the user has appropriately normalized the variables in the system so that their steady state values are of the order of 1. In that case, tolerance of $10^{-5}$ gives at least 4 significant digits in the solution.

Step 3 (Check accuracy of the candidate solution)

- Compare the candidate solution with the solution for an $\varepsilon$-ring that is 50% smaller. If the two solutions differ significantly, then decrease the $\varepsilon$-ring another 50% and re-run the program. Continue decreasing the $\varepsilon$-ring until the solutions for successive $\varepsilon$-rings differ by less than the tolerance criterion.
Exponential tracking reduces the errors in jump variables quickly but not instantaneously. The accuracy check requires therefore that the solutions for two consecutive values of $\varepsilon$ to be close over a common time horizon $(0, \bar{t})$, where $\bar{t}$ is smaller than the time at which the solution for the larger $\varepsilon$-ring hits the initial steady state (see Figure 6). Accordingly, the criterion for accuracy is that the solutions for successive $\varepsilon$-rings be virtually identical (i.e., not differ by more than $10^{-5}$ for any variable at any point) over the time horizon $(0, .67t_{1, \text{min}})$, where $t_{1, \text{min}}$ is the time at which the solution for the larger $\varepsilon$-ring is at its minimum distance from the initial steady state. The value of $t_{\text{min}}$ for the smaller $\varepsilon$-ring, $t_{2, \text{min}}$, is larger than $t_{1, \text{min}}$. When the candidate solution passes the accuracy test, the comparison of solution paths for the two $\varepsilon$-rings will show that the paths are close up to time $t_3$, where $.67t_{1, \text{min}} < t_3 < t_{1, \text{min}}$. The program calculates $t_3$ and presents the user with a solution for the time horizon $(0, t_3)$ and announces the result to the user by printing the message: “Congratulations! .... You are done. Hope to see you in another model...”

Since the program starts with an $\varepsilon$-ring of size $10^{-2}$, the solution covers more than 99% of the distance from the initial to the new steady state. Normally, this happens over a time horizon of 20+ years. Reducing the $\varepsilon$-ring will generate a solution for a longer time horizon and allow the user to see more of what happens in the very final stages of the adjustment process. The tolerance criterion of $10^{-5}$ in the computer programs can be reduced by the user by reducing the value for the parameter $\text{tol}$. However, it must be mentioned that in either case, computational time will rise exponentially as reducing either $\varepsilon$ or $\text{tol}$ will requiring making computations with higher precision.

3.2.2 Much Faster But Less Easy

Reverse Shoot Fast is much faster but slightly less user friendly. After computing the linearized solution, the program ascertains the slope of the dominant eigenvector ray (DER) and the direction of approach as the saddle path draws close to the new steady state. Armed with this information, the program limits the search for the global saddle path to a narrow segment of the $\varepsilon$-ring. Figure 10 shows why a little information makes a big difference to the efficiency of the search process.

The true solution converges to the DER as the path approaches the steady state $F$. The solution for $\lim_{t \to \infty} \mathbf{z}(t) - \mathbf{z}^*$ tells us, in addition, whether the direction of approach is from the southwest or northeast. Suppose it is from the southwest. The search should then start on the arc $(\varphi - \theta/2, \varphi + \theta/2)$, where $\varphi \in (\pi, 3\pi/2)$ is the angle formed by the DER and the horizontal axis. The parameter $\theta$ controls the angular span of the arc. It is set in the first block of the program and
is given the name \textit{spanangle}. What’s critical to recognize is that the search can start on a very small arc because, exceptional cases aside, the saddle path is very close to the DER at the point where it intersects the $\varepsilon$-ring. A few rounds of manual search can exploit this insight to rapidly reduce the search space to an arc of microscopic dimensions. Playing it safe, start with a “large” value for $\theta$ such as $\pi/100$ and evaluate trajectories passing between two points equally spaced on either side of $\varphi$. If the true solution is the trajectory associated with point $c$, then the minimum distance from the initial steady state will first fall and then rise as the search proceeds from $\varphi - \pi/200$ to $\varphi + \pi/200$. This indicates that the search can be conducted on a smaller arc; ergo abort the current execution and reduce $\theta$ to $\pi/10,000$ or $\pi/100,000$. If this arc contains point $c$, the solutions for minimum distance will again exhibit a U-shaped pattern. Continue the manual search until the solutions for minimum distance rise monotonically, signalling that point $c$ lies outside the selected arc. The “optimal” value of $\theta$ is then the value for the immediately preceding arc, the last arc where the solutions for minimum distance were U-shaped. Searching over this arc, the program finds the solution very quickly. \textit{Reverse Shoot Fast} is primarily a backup program. Ninety percent of the time \textit{Reverse Shoot Easy} will locate the saddle path in less than a minute. In a few cases, \textit{Easy} may need up to ten minutes. If this is considered too long, the user can switch to \textit{Reverse Shoot Fast}. The \textit{Fast} program requires the user to spend a couple of minutes carrying out exploratory runs to determine the right value of spanangle ($\theta$). But once spanangle is set, the solution time drops to a fraction of that in \textit{Reverse Shoot Easy}. 

Figure 10: The idea behind \textit{Reverse Shoot Fast}. 
3.3 A Primer on Using Reverse Shoot Easy

We will now look at the program that solves the above two state example with adjustment costs by reverse shooting. As alluded to earlier, we have implemented the algorithm in Mathematica. You need Version 5.2 or higher to run the programs. The choice of Mathematica deserves some discussion. We used Mathematica as it has perhaps the most sophisticated algorithm to solve nonlinear differential equations. More importantly, since we want to automate the process of searching for the global nonlinear saddlepath without expecting much expertise from the user, it was necessary the software we use provides a nonlinear differential equation solver that can solve any nonlinear differential equation by automatically shifting or switching the algorithms as it encounters difficulties such as stiffness. In fact, Mathematica book says that the code for nonlinear differential equation solver (NDSolve) and the related functions is 1400 pages long. Besides this, due to the ill-conditioning of the problem, it is important to be able to easily work with increasing level of precision as the progresses. This can be done very easily in Mathematica.

```
(... The precision of preliminary calculations. ...) 
(precision = 100, $MinPrecision = digits = precision);
```

![Initialization](Initialization.png)

![Calibration and Steady State](Calibration.png)

![Global Nonlinear Saddlepath](Saddlepath.png)

Figure 11: Overview of the Reverse Shoot Easy.

The Reverse Shoot Easy has the structure shown in Figure 11. The first command sets the precision of calculations prior to actually solving for the global nonlinear saddlepath at 100 decimal digits. This is much higher than the precision that is needed to solve for the saddle path in a typical dynamic economic problem but ensures that the system sent to NDSolve always has more precision than what it needs to solve for the saddlepath even if increases precision many times during the search process. NDSolve automatically reduces the precision of the system from 100 to what it
The remaining file is organized in three sections with *Initialization section* containing the values of parameters and the data for calibration as well as some useful normalization for the system and the specification of the shock to the system, if any. The *Calibration and Steady State section* contains the parametric specifications of utility, production function and the like and calibrates them based on parameter values and data in the Initialization section. The final, *Global Nonlinear Saddlepath section* solves for the global nonlinear saddlepath and begins with the cells where the user enters the model. First two sections are example-specific and user can change them in whatever manner he likes except for following some guidelines that are discussed below. In the Global Nonlinear Saddlepath section, all the entries that users can potentially change without compromising the integrity of the algorithm are marked red.

The Initialization section for our example is shown in Figure 12. First cell sets the directory in which the program store intermediate results. The second cell contains the values of parameters of the model. Note the comments in the cell. It is imperative that all parameter values are entered as fractions so that they have infinite precision from the point of view of the program. The overall precision of the system is determined by the number that has the lowest precision and entering any number as a decimal number is sure recipe for program to fail to work. In fact, besides having the typos in the specifying the model, this is perhaps the only reason why the user may find that program fails to work as claimed. Same caveat applies when entering the data for calibration, normalizing value of certain variables, and specifying the shocks in subsequent cells.

The Calibration and Steady State section for our example is as shown in Figure 12. The first cell spells out the parametrization of the utility function, production function and adjustment cost function. The next cell computes the initial steady state and calibrates the distribution paratemeters of these functions ($a_i$’s and $g$) using the data provided in the Initialization section. The only thing user needs to be cautious about here is to make sure that whenever, a numerical root finding algorithm such as *NSolve* or *FindRoot* is used, *AccuracyGoal* and *WorkingPrecision* options are set to digits to ensure that there is no loss in precision and results are accurate. In addition, the values of variables in initial and final steady states be subscripted by $o$ and $f$ for sake of consistency with the next section. This should definitely be ensured that at least the value of the variables in the system follow this convention.

You may recall that any thing in next section titled Global Nonlinear Saddlepath that is in red
Initialization

\begin{verbatim}
(* Set the directory where Mathematica will store intermediate results. *)
SetDirectory["C:";]

(* ALWAYS ENTER NUMERICAL VALUES AS FRACTIONS AND NOT AS DECIMALS TO
ENSURE REQUISITE PRECISION IN CALCULATIONS *)
\{\tau = \frac{3}{4}, \rho = \frac{1}{10}, \beta = \frac{1}{2}, \delta = \frac{1}{20}, \Omega = 2\} // N (* Parameters of the model *)

\{(0.75, 0.1, 0.5, 0.05, 2.)\}

\{\{bby_0 = \frac{25}{100}, mby_0 = \frac{15}{200}, m_0 = \frac{1}{2}, \theta_0 = \frac{1}{2}\} // N (* Data for calibration *)

\{(0.25, 0.075, 0.5, 0.5)\}

\{(z_0 = 1, k_0 = 1) // N (* Normalization *)

\{(1., 1.)\}

\{z_r = \frac{6}{5} z_o, \omega = \frac{1}{3}\} // N (* Shock - Exogenous rise in productivity *)

\{(1.2, 0.2)\}
\end{verbatim}

Figure 12: Initialization section of *Reverse Shoot Easy.*

color may be changed without affecting the algorithm. For example, see the first two cells of this section in Figure 13. These are the cells where user specifies the dynamic system that needs to be solved and the specified system appears in red color as it changes with the problem. Coming back to our example, it is clear from the first cell that I have used Mathematica to solve for the \(r(m, c, k, z)\) and \(\pi(m, c, k, z)\). It is usually possible to have *Mathematica* do such dirty work for you. This is also relevant for differential equation for some jump/ control variables. The ordering of variables in the system as defined in first cell is important: the state variables should always in the end. Further the same order of variables should be maintained subsequently.

If you are adapting this file to your own example, at this point you are almost done. The only thing that remains to be done in the rest of this section is to change the names of the variables to match that of your model. The names or part of the names of the variables that may be so changed again appear in red color as first cell in Figure 14 shows: \(\text{mdot}\) has \(m\) in red color, \(\text{cdot}\) has \(c\) in
Calibration and Steady State

Define utility function and production function etc.

\[ U(c_m) = \frac{1}{1 - \gamma} \left( \frac{\gamma - 1}{\gamma} \right)^{c_1 \cdot \gamma} \cdot Y(k, z) \cdot \psi(x) = \frac{n}{2} (x - \delta)^2; \]

\[ n = \frac{1}{\alpha \delta}; \quad Y_x(k, z) = D(Y(k, z), k); \]

Calibrate and compute the original steady state

\[ Y_x = Y(k_0, z_0); \quad c_x = \alpha \delta; \quad m_x = \beta b Y_x, \quad b = \beta b Y_x, \quad L = \pi_x m_x = \rho b b; \]

\[ g = (\alpha_0 + \rho) \left( \frac{m_x}{c_x} \right)^{1/r}; \quad (a_0, a_1) = ; \]

\[ (a_0, a_1) / ; \quad \text{FindRoot}\left[ \left( \frac{Y_x(k_0, z_0) = \rho + \delta}{\theta_0 = (\rho + \delta) k_0, (a_0, 1), (a_1, 1), \text{MaxIterations} \rightarrow 1000, \text{AccuracyGoal} \rightarrow \text{digits}, \text{WorkingPrecision} \rightarrow \text{digits} \right) ; \right] \]

Compute the new steady state

\[ k_x = ; \quad k_x = k_x / ; \quad \text{FindRoot}\left[ \left( Y_x(k_x, z_x) = \rho + \delta, (k_x, k_x), \text{MaxIterations} \rightarrow 1000, \quad \text{AccuracyGoal} \rightarrow \text{digits}, \text{WorkingPrecision} \rightarrow \text{digits} \right) ; Y_x = Y(k_x, z_x); \right] \]

\[ c_x = Y_x - \delta k_x; \quad (m_x, \pi_x) = ; \]

\[ (m_x, \pi_x) / ; \quad \text{FindRoot}\left[ \left( g \left( \frac{m_x}{c_x} \right)^{1/r} = \pi_x + \rho, L = \pi_x m_x = \rho b b, (m_x, \pi_x), (\pi_x, \pi_x) \right), \right] \quad \text{AccuracyGoal} \rightarrow \text{digits}, \text{WorkingPrecision} \rightarrow \text{digits} \right) ; \]

\[ \left( (m_x, c_x, k_x, Y_x, b, L, z_x, \pi_x, m_x, \pi_x, \pi_x) / \right) / \text{MatrixForm} \]

\[
\begin{array}{cccccccc}
0.0225 & 0.25 & 1.0 & 0.3 & 0.075 & 0.00375 & 1.0 & 0.5 & 11.1111 & 0.075 \\
0.0444438 & 0.331822 & 1.19089 & 0.391366 & 0.075 & 0.00375 & 1.2 & 0.253128 & 7.4661 & 0.11356
\end{array}
\]

Figure 13: Calibration and Steady State section of Reverse Shoot Easy.

red color and so on.

The user may note that the Initialization cell (Figure 15) contains the parameters settings for the search for the global nonlinear saddlepath. The program starts with an \( \varepsilon \)-ring of size \( 10^{-2} \) (of the change in state variables across steady state) so that the corresponding solution covers 99% of the distance from the initial to the new steady state. Normally, this happens over a time horizon of 20+ years. Reducing the \( \varepsilon \)-ring will generate a solution for a longer time horizon. The value of \( tol = 10^{-5} \) implies that the solutions for the variables in the system are accurate to 5 decimal digits. Since we have normalized the value of the capital stock in the economy to 1 this means we would have at least 4 to 5 digit accuracy in each variable in the system which is more than acceptable for
Global Nonlinear Saddle Path

(*) Specify the nonlinear system with state variables at end and maintain same order subsequently *)

\[
\begin{align*}
\dot{m} & = L \cdot r_b - p_i m; \\
\dot{c} & = \tau (r - \rho); \\
\dot{k} & = \hat{i}(c, k, z) - \delta k; \\
\dot{z} & = \omega (z_t - z);
\end{align*}
\]

\[\text{sysvars}_{\text{subs}} = \{ m \mapsto m[t], c \mapsto c[t], k \mapsto k[t], z \mapsto z[t] \};\]

\[\text{system} = \{ \dot{m}, \dot{c}, \dot{k}, \dot{z} \};\]

(*) Solve for the endogenous variables *)

\[
\begin{align*}
i[c, k, z] & = \\
\text{i}. & \text{Solve}\{ Y[k, z] - c - i \cdot \psi(i/k) \cdot k = 0, (i), \text{WorkingPrecision} \rightarrow \text{digits}[[1, 1]] \}; \\
\text{idot} & = \text{Dt}[i[c, k, z]] / \text{Dt}[c] \rightarrow \text{cdot}, \text{Dt}[k] \rightarrow \text{kdot}, \text{Dt}[z] \rightarrow \text{zdot}; \\
eq \text{eq1} := & \psi'\prime[i[c, k, z] / k] \cdot i[c, k, z] \cdot \left( \frac{\text{idot}}{i[c, k, z]} - \frac{\text{kdot}}{k} \right) = \\
& r + \delta - (Y_k[k, z] + \psi'\prime[i[c, k, z] / k] \cdot i[c, k, z] / k - \psi[i[c, k, z] / k]) / \\
& (1 + \psi'\prime[i[c, k, z] / k]); \\
eq \text{eq2} := & \tau + \pi = \varphi \left( \frac{m}{c} \right)^{1/t};
\end{align*}
\]

\[
\begin{align*}
\{ r[m, c, k, z], p_i[m, c, k, z] \} = \\
(r, p_i) / \text{Solve}[\{ \text{eq1}, \text{eq2}, (r, p_i), \text{WorkingPrecision} \rightarrow \text{digits}[[1]] \}; \\
\text{endovars}_{\text{subs}} = \{ r \mapsto r[m, c, k, z], p_i \mapsto p_i[m, c, k, z] \};
\end{align*}
\]

Figure 14: Entering the dynamic system in Reverse Shoot Easy.

an economist. All that needs to be done now it to evaluate the file up to and including the Main Cell and relax and watch the program solve for the global nonlinear saddlepath. The tolerance criterion of $10^{-5}$ can be reduced by the user if necessary. However, it must be mentioned that in either case, computational time will rise exponentially as reducing either $\varepsilon$ or $tol$ will requiring making computations with higher precision.

3.3.1 Interpreting Output of Reverse Shoot Easy

Let us now look at the output from the program as it searches for the nonlinear saddlepath. Step 1 begins with the initial search on $n = 20$ points $p_i$, $i = 1, 2, \ldots, n$. Figure 16 shows the output from Reverse Shoot Easy: for each point on the $\varepsilon$-ring, the program prints out the duration for the solution, the minimum distance of the trajectory in the state space from the initial steady state, and the time at which the trajectory comes closest to the initial steady state. Note the following general pattern: the trajectories the come closer to the initial steady state correspond to paths that last longer and the time of closest approach, $t_{\text{min}}$, is also larger. This is a very general pattern
that we will observe across all shooting algorithms and is closely related to the exponential tracking property. The output ends by showing which trajectory the program has found to be closest to the initial steady state during the initial search.

<table>
<thead>
<tr>
<th>Trajectory</th>
<th>Duration of Solution</th>
<th>Minimum Distance</th>
<th>Time of Closest Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>104.451</td>
<td>0.277767893681396</td>
<td>7.885920582191326</td>
</tr>
<tr>
<td>1</td>
<td>36.1609</td>
<td>0.2783607716965848</td>
<td>4.55517798546887</td>
</tr>
<tr>
<td>2</td>
<td>32.712</td>
<td>0.2786748629680602</td>
<td>3.706239988685183</td>
</tr>
<tr>
<td>3</td>
<td>31.2336</td>
<td>0.278535304635160</td>
<td>3.19457612998760</td>
</tr>
<tr>
<td>4</td>
<td>30.4185</td>
<td>0.27859751782329</td>
<td>2.794793220821373</td>
</tr>
<tr>
<td>5</td>
<td>30.1321</td>
<td>0.2782220285959219</td>
<td>2.41913843251530</td>
</tr>
<tr>
<td>6</td>
<td>30.4232</td>
<td>0.27859346788476</td>
<td>1.978144290436892</td>
</tr>
<tr>
<td>7</td>
<td>31.2237</td>
<td>0.276948128320578</td>
<td>1.241868581745512</td>
</tr>
<tr>
<td>8</td>
<td>32.8128</td>
<td>0.2762120150565422</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>9</td>
<td>36.109</td>
<td>0.2752900226180188</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>110.046</td>
<td>0.2000000000000136</td>
<td>84.0068148957447</td>
</tr>
<tr>
<td>11</td>
<td>32.835</td>
<td>0.2673406619934571</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>12</td>
<td>29.44</td>
<td>0.2686772026838359</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>13</td>
<td>28.0453</td>
<td>0.2696259862409052</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>14</td>
<td>27.1125</td>
<td>0.2703958586774011</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>15</td>
<td>26.2977</td>
<td>0.2710456316343211</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>16</td>
<td>27.1742</td>
<td>0.2715667772094816</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>17</td>
<td>28.0452</td>
<td>0.2719435768170521</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>18</td>
<td>29.6354</td>
<td>0.272904777764260</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>19</td>
<td>32.856</td>
<td>0.2738748850055374</td>
<td>Time of Closest Approach</td>
</tr>
<tr>
<td>20</td>
<td>32.1044</td>
<td>0.277767893681396</td>
<td>Time of Closest Approach</td>
</tr>
</tbody>
</table>

Extreme Left Trajectory: 0 Closest Trajectory: 10 Extreme Right Trajectory: 20 No. of Subdivisions of the ε-ring: 20

Current Minimum Distance: 0.2000000000000136 Time of Closest Approach: 84.0068148957447

Figure 15: The Initialization cell in Reverse Shoot Easy.

The output from a typical bisection gives the value of \( i \) corresponding the two points on the arc on which the search was conducted, number of current subdivisions, \( n \), of the \( \epsilon \)-ring, the trajectory that is closest to initial steady state at the end of current bisection along with its minimum distance.
and corresponding $t_{\text{min}}$ (Figure 17). It also prints out the closest trajectory in the state space. Remaining part of output shown in Figure 17 is produced if current search produces a trajectory that meets the tolerance criterion, $tol/3$, at which point the program starts step 2 to check the precision of the candidate solution just obtained by increasing precision by 4 digits.

**Figure 17:** Output as *Reverse Shoot Easy* enters Step 2.

Figure 18 shows the output as *Reverse Shoot Easy* finishes Step 2 and enters Step 3. It shows the maximum difference in values of the variables in the system. For example, the solution for $m$ with increased precision differed from earlier solution at most by $1.42098 \times 10^{-7}$ along entire computed transition path. Since the differences for all variables are less than the prescribed tolerance, the program concludes ‘THE PRECISION IS SATISFACTORY’ and starts Step 3 to check for accuracy of the current candidate solution.

For this, it contract the $\varepsilon$-ring by 50% (see Figure 18), precision is left unchanged at 20 decimal digits. The program now finds the path that comes closest to the initial steady state begining at this $\varepsilon$-ring. This path is compared with the current candidate solution. If the difference for various variables in the system is less than the tolerance $tol$, we would be done. In this case, this does not happen. The program, therefore, makes the path computed for this reduced $\varepsilon$ as the new candidate solution. To vet the precision of the new candidate solution, it increases precision to 24 digits (see Figure 19). Incidentally, note that as $\varepsilon$-ring is now smaller, durations of the solution and the $t_{\text{min}}$s increase. Having successfully checked for the precision of the new candidate solution, the program produces the output shown in Figure 20 indicating precision is satisfactory but epsilon is being
Extreme Left Trajectory = 42949683720 Closest Trajectory = 42949683721
Extreme Right Trajectory = 42949683724 No. of Subdivisions of the ε-ring = 85899345920
Current Minimum Distance = 3.7385067319804336012 × 10⁻⁶ Time of Closest Approach = 91.681083766374333414

THE PRECISION IS SATISFACTORY. The variables changed by less than ‘tol’ due to latest increase in ‘precision’ for current ε-ring.

Starting Step 3: Checking for the accuracy of the candidate solution.
Decreased ‘epsilon’ by a half to 0.00183823809163662360893.

Figure 18: Output as Reverse Shoot Easy enters Step 3.

3.4 Using Reverse Shoot Fast

The procedure for setting up the system is exactly the same for doing the reverse shooting for two state variables using Reverse Shoot Fast. It shares same overall structure as shown in Figure 11. The Initialization section and Calibration and Steady State section are exactly same. As in Reverse Shoot Easy, the first two cells of the Global Nonlinear Saddlepath section are used to specify the system. As the user will see, other programs also share the same structure. In fact, from the perspective of the user, many of them are essentially identical: they differ in details that do not affect the user when solving for the global nonlinear saddlepath.

The Initialization cell for Reverse Shoot Fast, however, has variable spanangle which user sets (Figure 22). Start with a large value of π/100 for spanangle and execute the Initialization and the Main cells. The first few lines of output will be as in Figure 23. Abort evaluation once these initial trajectories are evaluated. As long as closest trajectory from the search is the middle trajectory labeled 1, the intersection of the nonlinear saddlepath with the ε-ring lies within the arc the user is currently searching over. Reduce spanangle in large steps by diving by 100 and re-evaluate the

31
Target Tolerance = $3.33333 \times 10^{-6}$ Tolerance Achieved = $2.4485635122837641121 \times 10^{-6}$

Increased 'precision' by 4 digits to 24 digits.

Trajectory 0 Duration of Solution = 114.696 Minimum Distance = 0.277120533406910358100168 Time of Closest Approach = 7.895389714957864374117

Trajectory 1 Duration of Solution = 39.695 Minimum Distance = 0.277417147016838635521682 Time of Closest Approach = 4.5801905296420689796

Figure 19: Output as Reverse Shoot Easy as current candidate solution fails the accuracy test.

Initialization and the Main cells and repeat the process until the minimum is no longer on the middle trajectory. At that point, change spanangle back to its previous value. Now with spanangle set just evaluate the Initialization and Main cells and relax. The output produced by Reverse Shoot Fast is identical to that of Reverse Shoot Easy except for the fact that initial search is now done only for 3 trajectories.

3.5 Complex Eigenvalues and Oscillatory Dynamics

We have concentrated on the case where the negative eigenvalues are real. This is the most challenging case because the associated distance mapping is badly ill-conditioned. It is also the case we have encountered most often in our own work. The present paper is no exception. When the eigenvalues are complex conjugates, the approach to the new steady state is oscillatory and the global saddle path may intersect the $\varepsilon$-ring at several points. If it does, the distance mapping has multiple global minima but still has no other local minima. More importantly, since there is no DER, trajectories do not gravitate toward a single point on the $\varepsilon$-ring.16 This makes life easy as the distance mapping is differentiable and nicely conditioned with wide, smooth valleys. Reverse Shoot Easy finds the solution in the blink of an eye, but programs based on differentiable method should also work well.
4 Reverse Shooting with 3 State Variables

In systems with three state variables, the global saddle path intersects a δ-diamond centered around the new steady state. Searching for this intersection point is a daunting task. Because of the extra dimension, primitive bracketing no longer works. A smart polytope method, however, can find the solution. As before, information from the minimum distance mapping plays a crucial role in guiding the search.

4.1 The Minimum Distance Mapping

Turn back to Figure 8 for a moment. The distance mapping for models with two state variables has a “hole” around the intersection of the global saddle path with the ε-ring. In the case of three state variables, the hole expands to a crack. To see this, examine the solution for the state variables in a linear system:

\[
\begin{bmatrix}
  z_1(t) - z_1^*
  \\
  z_2(t) - z_2^*
  \\
  z_3(t) - z_3^*
\end{bmatrix}
= \begin{bmatrix}
  X_{41}
  \\
  X_{51}
  \\
  X_{61}
\end{bmatrix} h_1 e^{\lambda_1 t} + \begin{bmatrix}
  X_{42}
  \\
  X_{52}
  \\
  X_{62}
\end{bmatrix} h_2 e^{\lambda_2 t} + \begin{bmatrix}
  X_{42}
  \\
  X_{52}
  \\
  X_{62}
\end{bmatrix} h_3 e^{\lambda_3 t},
\]

where again let \( \lambda_1 \) be the dominant (negative) eigenvalue so that \( \lambda_3 < \lambda_2 < \lambda_1 < 0 \) and \( h_1 - h_3 \) are constants determined by initial conditions. As \( t \to \infty \), the trajectories formed by \( \{z_1, z_2, z_3\} \) in the state space gravitate toward the dominant eigenvector ray \( \{X_{41}, X_{51}, X_{61}\} \) and the minimum
Extreme Left Trajectory = 687194777948 Closest Trajectory = 687194777949
Extreme Right Trajectory = 687194777952 No. of Subdivisions of the ε-ring = 1374389534720
Current Minimum Distance = 3.88907227605729246151337 × 10⁻⁶ Time of Closest Approach = 112.556198931753121163036

Target Tolerance = 3.33333 × 10⁻⁵ Tolerance Achieved = 5.20533760977295857926391 × 10⁻⁷
Difference in values of variables due to latest decrease in epsilon = (3.33631 × 10⁻⁸, 1.19801 × 10⁻⁷, 9.45938 × 10⁻⁸, 7.16228 × 10⁻⁷)

CONGRATULATIONS! The variables changed by less than 'tol' due to latest decrease in 'epsilon'.
You are done! Hope to see you in another model. Continue with the next cell.
The solution is accurate within tolerance 'tol' upto 75 years. Continue with next cell.

Figure 21: Successful completion of search after meeting accuracy check for the new candidate solution.

(NEW cell - Initialization for finding the required trajectory. *)
tol = 10⁻⁵;  (* minimum distance of acceptable trajectory from initial steady state *)
spanangle = π / 40000;  (* the angular span of the arc of ε-ring on which search is confined *)
thetabegin = π + Re(SlopeDER) - spanangle / 2;  (* starting point of the arc of ε-ring on which search is confined *)
epsilon = 10⁻² Norm([kr - ko, zk - zo]);  (* radius of the ε-ring *)

Figure 22: Initialization cell for Reverse Shoot Fast

The spatial intuition is the same when the system is nonlinear. Connecting the points where
Trajectory 0  Duration of Solution: 80.884  Minimum Distance: 0.25785477785208  Time of Closest Approach: 58.94986474977051
Trajectory 1  Duration of Solution: 110.046  Minimum Distance: 0.2000000000000136  Time of Closest Approach: 84.00681484957447
Trajectory 2  Duration of Solution: 77.5253  Minimum Distance: 0.2054018764501246  Time of Closest Approach: 69.54036563261533
Extreme Left Trajectory: 0  Closest Trajectory: 1  Extreme Right Trajectory: 2  No. of Subdivisions of the $\varepsilon$-ring: 2
Current Minimum Distance: 0.2000000000000136  Time of Closest Approach: 84.00681484957447

Figure 23: Initial output from Reverse Shoot Fast.

Figure 24: The Rosenbrock Curve.

Trajectories intersect the $\delta$-diamond now traces out a curved line in Figure 24. We label this line the Rosenbrock Curve (RC) because it delineates the boundary of the Rosenbrock valley where the solution for minimum distance from the initial steady state decreases very rapidly. In fact, we are battling against an exceptionally nasty version of the classic Rosenbrock problem. In the Rosenbrock problem, the mapping is U-shaped at the bottom. By contrast, as explained earlier (see Figure 8), our distance mapping is kinked and hence nondifferentiable at the minimum. Moreover, the RC in Figure 24 is many, many orders of magnitude narrower than in the classic Rosenbrock problem. To examine the nature of the distance mapping in three state variable case, let us draw lines perpendicular to each point on the RC. In Figure 25, $RV_B$, $RV_C$, and $RV_D$ show how minimum distance varies along perpendiculars paired with points $B$, $C$, and $D$.

Trajectories that hit the perpendiculars at points closer to the RC start closer to the initial steady state and have smaller values of minimum distance in the time-reversed system. Approaching

35
the $RC$ from either direction, the reduction in minimum distance is relatively small until a trajectory intersects the $\delta$-diamond at a critical point very close to the curve. The landscape then changes suddenly. As the mapping passes the top of the narrow $V$, minimum distances plunges in a free fall to its minimum, the associated point on the $RC$. Moving along the $RC$ from $B$ to $C$ to $D$ sees minimum distance decrease at a tremendous rate: $MD_1 >> MD_2 >>> MD_3$, etc. The global distance mapping is nondifferentiable not just at $M$, which is global minimum we are looking for, but at every point on the $RC$. While the distance mapping is nondifferentiable all along the $RC$, point $M$ stands out as the one point where every directional derivative is undefined.

The Problem with Business-as-Usual

Lack of differentiability is not a barrier to the use of polytope methods, which aim only for a local improvement at each stage of the search. The most natural choice, however, the simplex method, runs into trouble on the $RC$. In Figure 26, the search has just finished computing minimum distance for the trajectories associated with vertices $A$, $B$, and $C$. Point $A$ is best and $B$ is worst. In the next iteration, point $B$ is rotated through line $AC$ to generate the new trial point $D$. But point $D$ is certain to be worse than point $A$ because the $RC$ is extremely narrow. Failing to find a local improvement, the algorithm concludes erroneously that the solution must lie inside the simplex.

Figure 25: The nature of distance mapping in neighborhood of $RC$ curve.
The next iteration searches therefore over a smaller simplex.\footnote{\textsuperscript{11} Some algorithms generate an additional trial point by rotating point $C$ through $AB$ before shrinking the simplex.} But since the the $RC$ has almost no width, the simplex has to contract to a very small triangle to find a very small adjustment that reduces the value of minimum distance: To find an improvement, the simplex has to be on the same order of magnitude as the width of the $RC$. The standard simplex method is simply too dumb. It works in theory but not in practice.

\section*{4.2 The Circle-Search Algorithm}

An efficient search does two things. First, it quickly finds the $RC$. Second, after finding the $RC$, it follows the curve to the global minimum. In this section we describe a refined simplex algorithm that satisfies the two conditions for efficiency. The algorithm starts with a good initial guess by solving for the point where the saddle path of the linearized model intersects the $\delta$-diamond. This narrows the search to one of the eight faces of the diamond.\footnote{\textsuperscript{12} In theory, point $M$ could lie on a different face of the diamond. This is very unlikely because far out on the transition path the linearized solution is close to the true solution.} Typically, the initial guess lies in the general vicinity of the $RC$. This is the case portrayed in Figure 27.

Starting from point $A$, the algorithm searches over a sequence of circles for local improvements. In the first round, the search is conducted over a large circle that intersects the $RC$.\footnote{\textsuperscript{13} The position of the $RC$ is unknown at the start of the search. But even if the first circle misses the curve, the} The trajectory...
that starts from point $B$ (in the time-reversed system) emerges as the trajectory with smallest minimum distance from the initial steady state.\textsuperscript{14} If this trajectory does not meet the tolerance criterion, then the program updates the search process, making point $B$ the center of a new, smaller circle. This yields $C$ as the best point in the second round of the search. And so it goes. The updating mechanism moves the search along the $RC$ until a circle search close to $M$ does not find a local improvement. The algorithm then starts contracting the circle. On one of the smaller circles, it finds a point close to $M$ where minimum distance for the associated trajectory satisfies the tolerance criterion. The circle-search algorithm is fast, efficient, and reliable. The key innovation is that, at each stage, updating is based on sampling many points on a \textit{closed} curve. This ensures that the algorithm finds the $RC$ in a few iterations (occasionally, the first big circle misses the curve) instead of wandering all over the face of the diamond. Furthermore, after the search locates the $RC$, it moves quickly along the curve in “large” steps. In contrast to the standard simplex algorithm, the circle-search algorithm does not contract the simplex prematurely. Contraction occurs at the right point, when the global minimum lies in the interior of the circle.

\textsuperscript{14}There are two local minima where the circle intersects the $RC$. The program engages in bisection to determine the value of minimum distance at each minimum.
4.3 Implementation Procedure: Reverse Shoot 3D

The implementation routine for Reverse Shoot 3D is essentially the same as for Reverse Shoot Easy. There is only one slight difference. At the start of the program, the user is asked to specify $T$, the time horizon of interest. Specifying $T$ allows the $\delta$-diamond to be set at the size that minimizes computation time. This refinement is superfluous in the programs for models with two state variables – the solution invariably comes back in less than a minute without it. In Reverse Shoot 3D, however, the saving in computation time is often significant.

For expository purposes, we use the two state variable case to explain the connection between $T$ and computation time. Paths $AK$ and $BJR$ in Figure 28 are the solutions for the rings with $\varepsilon = 10^{-3}$ and $\varepsilon = 10^{-4}$. Suppose the paths hit points $K$ and $J$ around year twenty. Over this time horizon, the two paths are close to the stable manifold and to each other. The solution for $\varepsilon = 10^{-4}$ is required to track the stable manifold for another (say) twenty years. (In most economic models, the speed of adjustment slows down a great deal as the path gets close to the new steady state.) Since it takes longer to find path BJR, incurring the extra computation time is a pure waste if the user does not need the solution to be accurate for more than twenty years. The $\varepsilon$-ring should be fixed at $10^{-3}$.

All of this carries over to the case of three state variables. In Reverse Shoot 3D, the size of the
δ-diamond determines how long the solution stays close to the stable manifold. Let \( t_{\text{min}} \) be the time at which the solution trajectory for the linearized time-reversed system starting at its intersection with the \( \delta \)-diamond hits the initial steady state for a particular value of \( \delta \). Then program chooses \( \delta \) such that \( T + 10 < t_{\text{min}} < T + 20 \). The cushion of ten to twenty years is added to the time horizon specified by the user to allow exponential tracking time to work and to account for difference in durations of the solutions for the linearized system used to choose \( \delta \) and that of the nonlinear system.\(^{15}\)

The other steps parallel those in case of Reverse Shoot Easy. In keeping with the spirit of plug and play, the program computes the linearized solution and the coordinates of the initial guess in state space. Following this, it checks for internal rounding error by increasing precision and tests for accuracy by comparing the candidate solution with the solution produced by a smaller \( \delta \)-diamond. Everything is automated, so, as in Reverse Shoot Easy, the user’s main responsibility is to enter the model without typos.

### 4.4 Further Remarks

**‘Nonmonotonicity’ of the distance mapping.** It is conceivable that with 3 state variables, the distance mapping may have local minima besides the global minimum of zero. If the distance mapping is nonmonotonic, the search could get stuck at a local minimum. We have never encountered this in practice. The minimum appears to be unique in most economic models. In any case, if the problem should ever arise, it is easy to fix. Since the value of minimum distance is known to be zero at the global minimum, the program cannot be fooled. When it settles at a local minimum, it recognizes the problem and restarts the search at a new point. (So far, this command has never been activated.)

**Solving models with exogenous or endogenous growth.** In models with growth, whether exogenous or endogenous, although the variables in the dynamic system do not converge to a steady state in level form, it is always possible to rewrite the model so that ratios of variables converge to a constant on the balanced growth path if such a path exists. Once the system has been written in “ratio form,” our programs can solve for the global saddle path. All paths in level form can then be retrieved from the solutions in ratio form.

\(^{15}\)The program also conducts a preliminary check for the sufficiency of precision in calculations and raises precision if needed.
The current work in area of growth and effect of government policies on growth has restricted attention either to the balanced growth path which completely eschews the issues of transition. Or, the transition to the balanced growth path is based on linearized system which casts particular doubts on issues of welfare.

**Rosenbrock Curve - A Loch Ness Monster?** The $RC$ is not a figment of our imagination. In all of the simulations undertaken for this and other papers, the curve shows up and the distance mapping has the shape drawn in Figure 12. These sightings are numerous enough to constitute what Judd calls a “virtual proof.” While a virtual proof does not command the same respect as a real proof, it does allow us to say that the $RC$ is more real than the Loch Ness monster. We have many clear photos of the curve; there is no reason to think it does not exist.

### 4.5 An Example with 3 State Variables

In this example, the economy is small and open, the exchange rate floats, and foreign currency $F$ competes with domestic currency in the provision of liquidity services. World prices are constant and equal to unity, so the price level is one and the same as the exchange rate. The private agent solves the problem

$$\max_{c,m,l,b,F} \int_0^\infty \left[ \frac{c^{1-1/\tau}}{1-1/\tau} + \phi(m, F) \right] e^{-\rho t} dt,$$

subject to

$$A = m + b + F;$$
$$\dot{A} = Y(k, z) + L + rb - C - I - \psi \left( \frac{I}{k} \right) k - \pi m,$$
$$\dot{k} = I - \delta k,$$
$$A(0) = A_o \equiv m_o + b_o + F_o, \quad \text{and} \quad k(0) = k_o$$

Liquidity services enter the utility function in the form

$$\phi(m, F) = g_o \frac{[g_1 m^{1-1/\sigma} + (1-g_1) F^{1-1/\sigma}]^{1-\frac{1}{\sigma}}}{1 - \frac{1}{\sigma}},$$

where $g_o$ and $g_1$ are constants and $\sigma$ is the elasticity of substitution between domestic and foreign currency.
As before, the path of productivity is
\[
\dot{z} = \omega (z_f - z), \quad \omega > 0,
\]
\[
z(0) = z_o, \quad z_f > z_o.
\]
and government’s budget constraint is given by
\[
\dot{m} + \dot{b} = rb + L - \pi m.
\]

4.6 Using Reverse Shoot 3D

The procedure for setting up the system is exactly the same for doing the reverse shooting for two state variables using Reverse Shoot Easy and Reverse Shoot Fast. Reverse Shoot 3D shares overall structure shown in Figure 11. The Initialization section and Calibration and Steady State section are exactly the same. Once again the first two cells of the Global Nonlinear Saddlepath section are used to specify the system. These cells are shown in Figure 29.

**Global Nonlinear Saddlepath**

(* Specify the nonlinear system with state variables at the end and maintain same order subsequently *)

```plaintext
mdot = L*rL - p.im; cdot = r c (r - p); Fdot = Y[k, z] - c - i - \psi[i/k] k;
kdot = i - \delta k; zdot = \omega (z_e - z);
eq0 := \psi''[i/k] i i dot - kdot = r + \delta - Y_k[k, z] + \psi'[i/k] i/k - \psi[i/k];

idot=.; idot = First[idot/. Solve[eq0, {idot}]];
system = {mdot, cdot, idot, Fdot, kdot, zdot};
```

(* Solve for the endogenous variables *)

```plaintext
r[m_, c_, F_] = \phi_2[m, F] / \phi_1[m, F] ; pi[m_, c_, F_] = \phi_2[m, F] - \phi_2[m, F] / \phi_1[m, F] ;
endovarsubs = (r -> r[m, c, F], pi -> pi[m, c, F]);
```

Figure 29: Entering the dynamic system into the program.

The only difference is that the user does have to specify the solution time \( T \). This is done in the Initialization cell which is just before the Main cell. If you are adapting this file to your own example, at this point you are almost done. The only thing that remains to be done in the rest of
this section is to change the names of the variables to match that of your model. The names or part of the names of the variables that may be so changed again appear in red color as first cell in Figure 29 shows: \( m \text{dot} \) has \( m \) in red color, \( c \text{dot} \) has \( c \) in red color and so on. The value of \( T \) is set in the Initialization cell that is shown in Figure 30. In our example we have set \( T = 40 \) years.

```
(* INITIALIZATION CELL - Initialization for finding the required trajectory. *)
T = 40;
(* number of years for which you want the solution. BEWARE!!! Setting Tss larger than necessary can substantially increase computational time.*)
delta = 10^{-1} \left( (F_f - F_i)^2 + (k_i - k_o)^2 + (z_f - z_o)^2 \right)^{\frac{1}{2}};
(* size of \( \delta \)-simplex or distance of simplex from the final steady state *)
tol = 10^{-4};
(* desired accuracy of the solutions of the variables in the system *)
```

Figure 30: The Initialization Cell.

You do not need to change value of delta (\( \delta \)) as that is automatically adjusted by the program based on you choice of \( T \). The value of \( tol \) in this cell is \( 10^{-4} \) which implies that the solutions for the variables in the system are accurate to 4 decimal digits. Since we have normalized the value of the capital stock in the economy to 1 this means we would have at least 3 to 4 digit accuracy in each variable in the system which is quite acceptable for an economist. All that needs to be done now it to evaluate the file upto and including the Main Cell and relax and watch the program solve for the global nonlinear saddlepath.

### 4.6.1 Interpreting the Output of Reverse Shoot 3D

Let us now look at the output from the program as it searches for the nonlinear saddlepath (Figure 31). The vector \( \{-1, -1, -1\} \) indicates that all state variables approach their new steady state from below. Accordingly, the program selects the appropriate face of the \( \delta \)-diamond. The output shows that the linearized system hits the \( \delta \)-diamond in 55.45 years. This duration is in the permissible range for \( T \) of 40 years. Hence, no adjustment is made to the value of \( \delta \)-diamond that was set in the Initialization cell. The preliminary check also indicates that the initial precision of 16 is enough.

The next line shows the trajectory that starts from the intersection of the linearized solution with \( \delta \)-diamond (point in A in Figure 27) has a distance of .237981. The search around first circle centered at A results in a minimum distance of .198328. We have found the RC. In practice, point
$M$ (see Figure 27) the point we are seeking is very close to $A$. (We showed it otherwise in Figure 27 to make the exposition of circle search easier.) In view of closeness of $M$ to $A$, the program now decrease the radius $\varepsilon$ of the circle and continues this process until minimum distance rises. At some point, the minimum distance rises as $\varepsilon$ is reduced further (.0695882 to .235928 in Figure 31), and the circle search shift to a circle centered on a point on the $RC$ such as point $B$ in Figure 27.\footnote{In fact, in this case, this last circle completely missed the $RC$ which clearly indicates that $M$ is indeed very close to $A$.}

Let us now look at the output from the program as it searches for the nonlinear saddlepath or point $M$ along the $RC$. The typical output from a circle search is shown in Figure 32. Initial search around the circle loosely identifies the two intersections of the circle with the $RC$ as two local minima. Further bisection around the two local minima yields the minimum of .143564 which is worse than the value of .0695882 at the current best point which is the center of the circle. As a result, in next iteration the circle will not flip but will contract.

This process continues until minimum distance falls below prescribed tolerance $tol/3$.\footnote{It may so happen that before reaching this point, the program may run out of precision in which case it will increase precision by 4 and restart the search.}

\begin{verbatim}
BEGINNING ITERATION 1 WITH CENTER OF $\varepsilon$-RING AT 
(0.032402614498896658679289, 1.16304344347395864, 1.199991613054427416626)
Local minima from initial search over the ring.
1: (mindis, tdis, jmin, ntraj) = (0.2370448671934229, 0.0696882, 4, 20)
2: (mindis, tdis, jmin, ntraj) = (0.23650021931978433, 0.032402614498896658679289, 13, 20)
This Iteration: Minimum Distance = 0.143564 Time of Closest Approach = 37.7794 No. of Subdivisions of $\varepsilon$-RING = 3572114195268
AT THE END OF ITERATION 1: CURRENT MINIMUM DISTANCE = 0.0696882 TIME OF CLOSEST APPROACH = 49.61304344347395864
\end{verbatim}

Figure 32: Output from search around a circle centered on a point on Rosenbrock Curve.

Figure 31: Initial output from Reverse Shoot 3D.
point Step 1 is over. In our example, the output from program is shown in Figure 33. The program increases precision by 4 digits to 20 digits to check the precision of the candidate solution. Step 2 otherwise is like Step 1.

BEGINNING ITERATION 14 WITH CENTER OF ε-ring at (0.0324018406931471000, 1.163422782893766797, 1.19999790308750996410)
Local minima from initial search over the ring.
1: (mindis, tdis, jmin, ntraj) = (0.2266001196873122, 19.3641, 4, 20)
2: (mindis, tdis, jmin, ntraj) = (0.224882328022661, 20.6665, 13, 20)
This Iteration: Minimum Distance = 3.60457×10^{-6}  Time of Closest Approach = 57.3424  No. of Subdivisions of ε-ring > 7820751455710405381808
AT THE END OF ITERATION = 14:  CURRENT MINIMUM DISTANCE = 3.60457×10^{-6}  TIME OF CLOSEST APPROACH = 57.3424006873490
Target Tolerance = 0.000333333  Tolerance Achieved = 3.604571478033548×10^{-6}
Starting Step 2: Checking for the precision of the candidate solution.
Increased ‘precision’ by 4 digits to 20 digits.
NONLINEAR SYSTEM: Minimum Distance from Initial Steady State = 0.221474  Time of Closest Approach = 23.2779  Solution Time = 23.2779
Initial Ring = 1:  MINIMUM DISTANCE = 0.00849947  TIME OF CLOSEST APPROACH = 57.310404030429932647
Initial Ring = 2:  MINIMUM DISTANCE = 0.000956285  TIME OF CLOSEST APPROACH = 57.25029045902738053
Initial Ring = 3:  MINIMUM DISTANCE = 0.00192792  TIME OF CLOSEST APPROACH = 57.156841182879194747

Figure 33: Output as Step 1 ends and Step 2 Starts.

The solution obtained from Step 2 vets the candidate solution as shown in Figure 34 by indicating ‘the precision is satisfactory’. Notice that Step 2 took only 5 iterations compared to 14 iterations for Step 1. That is not surprising. We have already have a good estimate of M from Step 1 and Step 2 just refines it by using higher precision and check if that refinement leads to changes in variables that exceed specified tolerance. Step 3 which checks the accuracy of the candidate solution (i.e. check if deviations away from the stable manifold are a cause of concern over time T or have been taken care of by exponential tracking) starts with contracting δ-diamond by a 50% and finding the intersection of the linearized solution with the relevant face of the new δ-diamond which becomes the center for initial circle search in Step 1.

Recall, in the two state example discussed earlier, first candidate solution was not vetted by accuracy check. In this example, we are lucky and the first candidate solution is passes the accuracy check. The final output from the program is shown in Figure 35. It took 17 iterations to find the solution in Step 3. Both change in precision (from Step 2) and change in δ result in differences in values of variables in the system that are less than the tolerance of 10^{-4} prescribed in Initialization cell. In fact, actual achieved tolerance is close to 10^{-5} over time T. Thus, over 40 year horizon, the values of variables in system are accurate to 5 decimal digits. The output in Figure 35 also says that if user is comfortable with prescribed tolerance of 10^{-4}, then he can use solution paths till 57
Figure 34: Output as Step 2 ends and Step 3 starts.

The actual time path of some important variables of the model is shown in Figure 36. Note that despite agents’ having access large stockpile of foreign currency, due to strong intertemporal consumption smoothing (τ has a low value of .25), 20% productivity shock leads to prolong fall in investment. The fall in inflation on impact is very temporary as rising interest rate raises the debt-service burden of government.

4.7 A Picture of the Minimum Distance Mapping

In the previous subsection, we showed how initial search around a circle led to two local minima. We have also claimed the existence of RC which contains M where minimum distance mapping has global minimum of zero. However, for those who are still sceptic of the existence of RC like that of Loch Ness monster, Figure 37 provides two perspectives. Panel A shows that as one moves along a circle, indeed there are two local minima and the mapping the quite flat over a large region. But more importantly, it is non-differentiable at these two local minima. Panel B will convince the reader that even if we knew RC, it would not be possible to use differentiable methods to locate M within it. Essentially, we are stuck with using refined simplex based technique of which circle-search
Beginning iteration 17 with center of \( \epsilon \)-ring at \((0.0325116290387952525487, 1.177486854217484666804, 1.19999900757636786848398)\).

Local minima from initial search over the ring.

1: \((\text{mindis}, \text{tdis}, jmin, ntraj) = (0.23876620460021237707, 23.4377, 4.02)\)
2: \((\text{mindis}, \text{tdis}, jmin, ntraj) = (0.23749161778566510140, 24.7309, 13.02)\)

This iteration: Minimum Distance = 0.0000312083 Time of Closest Approach = 72.5694 No. of Subdivisions of \( \epsilon \)-ring = 410638661974841900562862504000

At the end of iteration 17: Current Minimum Distance = 0.0000312083 Time of Closest Approach = 72.569352873569529446

Target Tolerance = 0.0000333333 Tolerance Achieved = 0.0000312083

Difference in values of variables due to latest increase in precision = \(1.1283 \times 10^{-6}, 4.68493 \times 10^{-6}, 3.51503 \times 10^{-6}, 7.47941 \times 10^{-6}, 9.03884 \times 10^{-6}, 3.66891 \times 10^{-6}\)

Difference in values of variables due to latest decrease in \( \epsilon \) = \(1.1283 \times 10^{-6}, 4.68493 \times 10^{-6}, 3.51503 \times 10^{-6}, 7.47941 \times 10^{-6}, 9.03884 \times 10^{-6}, 3.66891 \times 10^{-6}\)

Congratulations! The variables changed by less than 'tol' due to latest decrease in 'epsilon'.

You are done! Thank you for using Reverse Shoot 3D. The solution is accurate up to 57 years. Continue with next cell.

Figure 35: Final output from Reverse Shoot 3D.

Figure 36: The solutions for the system from Reverse Shoot 3D.
is one example.

5 Reverse Shooting with 4 State Variables

It is possible to generalize our search strategy to models with 4 state variables. Following the same general strategy as in the three state variable case, the geometry of the state space again allows the search to be conducted in a lower dimension. In the three state variable case, the intersection of the linearized solution with the $\delta$-diamond picks out the 2-dimensional simplex (i.e., one face of the diamond) that the global saddle path penetrates on its way to the new steady state. Similarly, in models with four state variables, the linearized solution picks out the relevant 3-dimensional $\delta$-
simplex. With three state variables, saddle path trajectories converge to an eigenvector plane in an intermediate phase of adjustment; with four state variables, the trajectories first gravitate toward the 3-dimensional eigenspace formed by the eigenvectors paired with the three largest negative eigenvalues. The intersection of the eigenspace with the 3-dimensional simplex selected by the linearized solution defines a Rosenbrock Plane ($RP$). This plane contains the $RC$ which in turn contains the global minimum of the minimum distance mapping. The position of the $RP$ and $RC$ on this plane in the 3-dimensional simplex can be found by searching over a hollow sphere. In short, the algorithm for the four state variable case consists of one layer of search at a higher level plus the algorithm for the three state variable case. This layering approach, in fact, can be generalized to solve for systems with 5 state variable but current level of computing speed makes that extension not so useful. Once a point on the $RC$ has been identified, one now needs to search on a close surface (a sphere) around this point. This is similar to the search around a closed curve on the face of the $\delta$-diamond in case of a three state variables.

5.1 Searching with Spheres

The search process can be described by analogy with the search process for the three state variable case and with reference to Figure 27. Let $A$ be the intersection of the linearized solution with the relevant 3-dimensional simplex which plays same role here as the face of the $\delta$-diamond in three-state variable case. Start with a large sphere centered on $A$ so that it intersects the $RP$ and the $RC$. This initial search will find a point on the $RC$ which the center of the sphere for the next round of search. The similarity of the approach to that with the three state variable case is now obvious. If the search on a sphere does not lead to a reduction in minimum distance, it must be the case that the global minimum $M$ lies inside the sphere and hence we contract the sphere. This process continues alternating between relocating and shrinking the sphere until point $M$ is located will sufficient accuracy so the resulting saddlepath comes close to the initial steady state within the specified tolerance.

The search on the sphere can be broken down in search along the ‘meridians’ on the sphere. Suppose, the search is conducted on $n$ equispaced meridians on the sphere. Since the $RP$ intersects

---

18 This is not to demanding a requirement. Recall from 3-state variable case that not just $RC$ but even point $M$ was very close to the intersection of the linearized system with the $\delta$-diamond labeled $A$. The problem that we face in not that $M$ is far from $A$ but that it is hidden in a crack, the $RC$.

19 Meridians are half-rings like meridians on Earth’s surface.
the sphere, many (a contiguous set of) meridians will intersect the \( RP \). The search along these meridians will locate this point of intersection of the meridian with the \( RP \). As the sphere on which search is conducted also meets the \( RC \), out of these meridians that intersect the \( RP \), there will be typically two (sometimes may be only one) that will show local minima for the minimum distance mapping as we move around the sphere from meridian 1 back to meridian 1. This is similar to finding two local minima as we move along the circle in the case with three state variables. One can now search (on neighboring meridians) around these meridians with local minima to find the intersection of the \( RC \) with the sphere.

Once again from the point of view of the user, there is no additional effort required. He/She has to just make entries in Initialialization section and the Calibration and Steady State Section and enter the model equations in the first two cells of the next section. Thereafter, all that is needed is to make mechanical changes in the names of the variables and specify the time period \( T \) over which solution is desired and then relax.\(^{20}\)

5.2 An Example with 4 State Variables

Recall in our 3-state example, the economy was small and open, the exchange rate floated, and foreign currency \( F \) competed with domestic currency in the provision of liquidity services. World prices were constant and equal to unity. Let us augment this model by introducing a non-traded good in the model.

The production functions for the two sectors now are

\[
Y_N = z_0 F (k_N, l_N), \\
Y_T = zG (k_T, l_T),
\]

both of which are CES with elasticity of substitution \( \beta \). The adjustment cost function \( \psi \) is identical for both sectors. The total labor supply in the economy is fixed at \( l \). The capital is produced from traded and nontraded good using Cobb-Douglas specification with \( \alpha \) being the cost share of

\(^{20}\)At present, we have not reached this level of automation in four-state-variable case. The problem is that as the program keeps iterating and solving the system over and over again, Mathematica keeps hogging more and more memory. Perhaps it keeps storing results form previous solutions of the system which are no longer required. As a result, at some point the memory requirement goes beyond the RAM available. Thereafter, operating system switches to paging the contents of the memory to the hard drive which considerably slows down the program. Of course, there is nothing wrong with the algorithm.
Choosing the traded good as numeraire, zero profit in capital production sector implies

\[ P^k = C^k (P_N) \equiv \alpha^\alpha (1-\alpha)^{1-\alpha} P_N^\alpha, \]

where \( P^k \) and \( C^k(.) \) are the price and the unit cost function for capital, and \( P_N \) is the price of nontraded good. Since, traded good is numeraire, it is also the relative price of the nontraded good (in terms of the traded good).

The agent’s momentary indirect utility function is

\[ V (E, P_N) = \frac{1}{1-\frac{1}{\tau}} \left( \frac{E}{\gamma (1-\gamma)^{1-\gamma} P_N^\gamma} \right)^{1-\frac{1}{\tau}}, \]

so that elasticity of intertemporal substitution is \( \tau \), and the preference over goods at a point in time is Cobb-Douglas implying intratemporal elasticity of substitution of 1. The denominator in square brackets is the exact consumption based price index, \( P \). Thus, real exchange rate

\[ \nu = \frac{e}{P} = \gamma^{-\gamma} (1-\gamma)^{-(1-\gamma)} P_N^{-\gamma}, \]

where \( e \) is the nominal exchange rate.

Liquidity services enter the utility function in the form

\[ \phi(\nu_m, \nu_F) = g_o \left[ g_1 (\nu_m)^{1-1/\sigma} + (1-g_1) (\nu_F)^{1-1/\sigma} \right]^{\frac{1-\frac{1}{\tau}}{1-\frac{1}{\tau}}}, \]

where \( m \) is the domestic currency balances measured in terms of traded goods, \( g_o \) and \( g_1 \) are constants and \( \sigma \) is the elasticity of substitution between domestic and foreign currency. The agent also hold nominal bonds (denominated in domestic currency) issued by government and receives real lump transfers \( L \) from the government.

The private agent solves the problem

\[ \max_{c,m,l_N,l_T,b,F} \int_0^{\infty} [V (E, P_N) + \phi(\nu_m, \nu_F)] e^{-\rho t} dt, \]

subject to

\[ A = m + b + F; \]
\[ \dot{A} = P_N z_o (k_N, l_N) + z G (k_T, l - l_N) + \frac{L}{\nu} + (i - \chi) b - E - I_N - \psi \left( \frac{I_N}{k_N} \right) k_N - I_T - \psi \left( \frac{I_T}{k_T} \right) k_T - \chi m, \]
\[ \dot{k}_N = I_N - \delta k_N, \]
\[ \dot{k}_T = I_T - \delta k_T, \]
\[ A(0) = A_o \equiv m_o + b_o + F_o, \quad k_N(0) = k_{N,o} \quad \text{and} \quad k_T(0) = k_{T,o}. \]
The traded sector experiences a rise in productivity given by

\[ \dot{z} = \omega (z_f - z), \quad \omega > 0, \]

\[ z(0) = z_o, \quad z_f > z_o, \]

and government’s budget constraint is given by

\[ \dot{m} + \dot{b} = (i - \chi) b + \frac{L}{\nu} - \chi m. \]

The first-order conditions for agent’s problem are

\[ V_E(E, P_N) = \lambda_1, \]

\[ v \phi_m (vm, vF) = \lambda_1 i, \]

\[ v \phi_F (vm, vF) = \lambda_1 (i - \chi), \]

\[ \lambda_1 P^k \left( 1 + \psi' \left( \frac{I_N}{k_N} \right) \right) = \lambda_2, \]

\[ \lambda_1 P^k \left( 1 + \psi' \left( \frac{I_T}{k_T} \right) \right) = \lambda_3, \]

\[ P_N z_o F_i (k_N, l_N) = zG_i (k_T, l - l_N). \]

The co-state equations are

\[ \dot{\lambda}_1 = [\rho - (i - \chi)] \lambda_1, \]

\[ \dot{\lambda}_2 = (\rho + \delta) \lambda_2 - \left[ P_N z_o F_i (k_N, l_N) - P^k \left\{ \psi \left( \frac{I_N}{k_N} \right) - \psi' \left( \frac{I_N}{k_N} \right) \frac{I_N}{k_N} \right\} \right] \lambda_1, \]

\[ \dot{\lambda}_3 = (\rho + \delta) \lambda_3 - \left[ zG_i (k_T, l_T) - P^k \left\{ \psi \left( \frac{I_T}{k_T} \right) - \psi' \left( \frac{I_T}{k_T} \right) \frac{I_T}{k_T} \right\} \right] \lambda_1, \]

The good market clearing condition for nontraded goods is

\[ Y_N = \frac{\gamma E}{P_N} + \frac{\alpha P^k}{P_N} \left[ I_N + \psi \left( \frac{I_N}{k_N} \right) k_N + I_T + \psi \left( \frac{I_T}{k_T} \right) k_T \right], \]

and the current account equation gets modified to

\[ \dot{F} = P_N z_o F_i (k_N, l_N) + zG_i (k_T, l - l_N) - E - I_N - \psi \left( \frac{I_N}{k_N} \right) k_N - I_T - \psi \left( \frac{I_T}{k_T} \right) k_T. \]

Once again assume that stock of government bonds is fixed. If we work with the Lagrange multipliers then the core dynamic system of the model, then, consists of \((\lambda_1, \lambda_2, \lambda_3, m, F, k_N, k_T, z)\). However,
this system contains two endogenous variables \( P_N \) and \( l_N \) which cannot be solved for explicitly. Therefore, we expand the system by including them in the system.

How the differential equations for \( P_N \) and \( l_N \) needs some discussion. First note that, first-order condition for \( I_N \) and \( I_T \) can be solved for

\[
I_N = I_N(P_N, \lambda_1, \lambda_2, k_N),
\]
\[
I_T = I_T(P_N, \lambda_1, \lambda_3, k_T),
\]

and first-order conditions for the currency balances give

\[
i = i(P_N, \lambda_1, m, F),
\]
\[
\chi = \chi(P_N, \lambda_1, m, F),
\]

and first-order condition for consumption expenditure yields

\[
E = E(P_N, \lambda_1).
\]

To obtain the differential equations for \( P_N \) and \( l_N \), time differentiate the first-order condition for labor and market clearing condition for the nontraded good after substituting the functional relationships for \( I_N, I_T, \) and \( E \). Simplify the resulting equations and substitute for \( \dot{\lambda}_1, \dot{\lambda}_2, \dot{\lambda}_3, \dot{k}_N, \) and \( \dot{k}_T \). It is best to have Mathematica do these manipulations especially to avoid days of horriﬁying algebraic manipulations and untraceable typos. Appending these equations to the core dynamic system

\[
\dot{m} = (i(P_N, \lambda_1, m, F) - \chi(P_N, \lambda_1, m, F))b + \frac{L}{\nu(P_N)} - \chi(P_N, \lambda_1, m, F)m
\]
\[
\dot{\lambda}_1 = [\rho - (i - \chi)]\lambda_1,
\]
\[
\dot{\lambda}_2 = (\rho + \delta)\lambda_2 - \left[ P_N z_a F_1(k_N, l_N) - P^k \left\{ \psi \left( \frac{I_N}{k_N} \right) - \psi' \left( \frac{I_N}{k_N} \right) \frac{I_N}{k_N} \right\} \right] \lambda_1,
\]
\[
\dot{\lambda}_3 = (\rho + \delta)\lambda_3 - \left[ zG_l(k_T, l_T) - P^k \left\{ \psi \left( \frac{I_T}{k_T} \right) - \psi' \left( \frac{I_T}{k_T} \right) \frac{I_T}{k_T} \right\} \right] \lambda_1,
\]
\[
\dot{F} = P_N z_a F(k_N, l_N) + zG(k_T, l - l_N) - E - I_N - \psi \left( \frac{I_N}{k_N} \right) k_N - I_T - \psi \left( \frac{I_T}{k_T} \right) k_T,
\]
\[
\dot{k}_N = I_N - \delta k_N,
\]
\[
\dot{k}_T = I_T - \delta k_T,
\]
\[
\dot{z} = \omega (zf - z),
\]

53
gives the system $\left(P_N, l_N, \lambda_1, \lambda_2, \lambda_3, m, F, k_N, k_T, z\right)$ which has 6 jump variables and 4 state variables.

The boundary conditions for the problem are

$$F(0) = F_o, \quad \text{and} \quad \lim_{t \to \infty} F(t) = F_f,$$

$$k_N(0) = k_{N,o} < k_{N,f}, \quad \text{and} \quad \lim_{t \to \infty} k_N(t) = k_{N,f},$$

$$k_T(0) = k_{T,o} < k_{T,f}, \quad \text{and} \quad \lim_{t \to \infty} k_T(t) = k_{T,f},$$

$$z(0) = z_o < z_f, \quad \text{and} \quad \lim_{t \to \infty} z(t) = z_f.$$

5.3 Implementation Procedure: Reverse Shoot 4D

The implementation routine for Reverse Shoot 4D is essentially the same as for Reverse Shoot 3D except that now instead of a circle in each step the search is done over a sphere as discussed before.

Once again at the start of the program, the user is asked to specify $T$, the time horizon of interest which allows $\delta$-hyperdiamond to be set at the size that minimizes computation time. As with 3 state variable, the size of the $\delta$-hyperdiamond determines how long the solution stays close to the stable manifold. Let $t_{\min}$ be the time at which the solution trajectory for the linearized time-reversed system starting at its intersection with the $\delta$-diamond hits the initial steady state for a particular value of $\delta$. Then program chooses $\delta$ such that $T + 10 < t_{\min} < T + 20$. Again, a cushion of ten to twenty years is added to the time horizon specified by the user to allow exponential tracking time to work and to account for difference in durations of the solutions for the linearized system used to choose $\delta$ and that of the nonlinear system.\(^\text{21}\)

The other steps parallel those in case of Reverse Shoot 3D. In keeping with the spirit of plug and play, the program computes the linearized solution and the coordinates of the initial guess in state space. Following this, it checks for internal rounding error by increasing precision and tests for accuracy by comparing the candidate solution with the solution produced by a smaller $\delta$-hyperdiamond.

5.4 Using Reverse Shoot 4D

The procedure for setting up the system is exactly the same for doing the reverse shooting for two state variables using Reverse Shoot Easy and Reverse Shoot Fast. Reverse Shoot 3D shares overall structure shown in Figure 11. The Initialization section and Calibration and Steady State section

\(^{\text{21}}\)The program also conducts a preliminary check for the sufficiency of precision in calculations and raises precision if needed.
are exactly the same. Once again the first two cells of the Global Nonlinear Saddlepath section are used to specify the system. These cells are shown in Figure 38.

Global Nonlinear Saddlepath

![Image of dynamic system code]

Figure 38: Entering the dynamic system into Reverse Shoot 4D.

The value of $T$ is set in the Initialization cell which is shown in Figure 39. There is no difference with the Initialization cell for Reverse Shoot 3D. Here, have set $T = 30$ years. As before, the user do not need to change value of delta ($\delta$) as that is automatically adjusted by the program based on you choice of $T$.

5.4.1 Interpreting Output of Reverse Shoot 4D

To be included.
The idea of finding the minimum of a distance mapping can be easily extended to forward shooting. Although there are some differences at conceptual level, from the point of view of implementation
no new issues arise. To see the merit of the idea of computing minimum distance of a transition path from the final steady state as a method of finding the saddle path, let us go back to the neoclassical growth model.

As Figure 2 shows, transition paths corresponding to values of guesses $c^g$ closer to $c(0)$ come closer to the final steady than the once for which difference between $c^g$ and $c(0)$ is larger. This is exponential tracking in action but time flowing in different direction. This gives a very simple updating rule for finding $c(0)$: (1) Given current value of guess $c^g$, solve the system with guesses $c^g + \varepsilon$ and $c^g - \varepsilon$. (2) If either choice yields a lower value of minimum distance than $c^g$, that becomes the new value of $c^g$ and repeat the previous step. If not, reduce $\varepsilon$ and repeat step 1. Notice that unlike the standard forward shooting algorithm, we have not relied on model-specific information which variable turns negative (first) for decreasing or increasing $c^g$. Although this may seem as a trivial distinction in the context of the neoclassical model, it becomes the determining factor for writing reliable programs that can solve any problem with a given number of jump variables.

The reason is that once we are out of one jump variable case, one cannot draw phase diagrams that would tell us which variable will turn negative etc. In fact, there is no general pattern that will be observed for all models. It appears that similar problem besets the algorithm in the previous paragraph. We have implicitely relied on trapping argument there as transition paths in Figure 2 cannot cross. But this is deceptive. The idea that transition paths corresponding to values of guesses jump variables that are closer to their true values come closer to the final steady than the guess that lie farther is fairly general and holds in higher order systems.

To see this consider the solution of a two-state variable, two-jump variable system

$$
\begin{bmatrix}
    x_1(t) - x_1^* \\
    x_2(t) - x_2^* \\
    z_1(t) - z_1^* \\
    z_2(t) - z_2^*
\end{bmatrix}
= \begin{bmatrix}
    X_{11} & X_{12} & X_{13} & X_{14} \\
    X_{21} & X_{22} & X_{23} & X_{24} \\
    X_{31} & X_{32} & X_{33} & X_{34} \\
    X_{41} & X_{42} & X_{43} & X_{44}
\end{bmatrix}
\begin{bmatrix}
    h_1 e^{\lambda_1 t} \\
    h_2 e^{\lambda_2 t} \\
    h_3 e^{\lambda_3 t} \\
    h_4 e^{\lambda_4 t}
\end{bmatrix},
$$

where as before $x_i$s are jump variables and $z_i$s are state variables and $\lambda_1, \lambda_2 < 0 < \lambda_3, \lambda_4$. At $t = 0$ it gives

$$
\begin{bmatrix}
    x_1(0) - x_1^* \\
    x_2(0) - x_2^* \\
    z_{1,0} - z_1^* \\
    z_{2,0} - z_2^*
\end{bmatrix}
= \begin{bmatrix}
    X_{11} & X_{12} & X_{13} & X_{14} \\
    X_{21} & X_{22} & X_{23} & X_{24} \\
    X_{31} & X_{32} & X_{33} & X_{34} \\
    X_{41} & X_{42} & X_{43} & X_{44}
\end{bmatrix}
\begin{bmatrix}
    h_1 \\
    h_2 \\
    h_3 \\
    h_4
\end{bmatrix}.
$$
On the stable manifold, \( h_3 = 0 \) and \( h_4 = 0 \), and, therefore
\[
\begin{bmatrix}
  z_{1,o} - z_{1}^* \\
  z_{2,o} - z_{2}^*
\end{bmatrix}
= \begin{bmatrix}
  X_{31} & X_{32} \\
  X_{41} & X_{42}
\end{bmatrix}
\begin{bmatrix}
  h_1 \\
  h_2
\end{bmatrix},
\]
which gives
\[
\begin{bmatrix}
  h_1 \\
  h_2
\end{bmatrix}
= \begin{bmatrix}
  X_{31} & X_{32} \\
  X_{41} & X_{42}
\end{bmatrix}^{-1}
\begin{bmatrix}
  z_{1,o} - z_{1}^* \\
  z_{2,o} - z_{2}^*
\end{bmatrix},
\]
and
\[
\begin{bmatrix}
  x_{1}(0) - x_{1}^* \\
  x_{2}(0) - x_{2}^*
\end{bmatrix}
= \begin{bmatrix}
  X_{11} & X_{12} \\
  X_{21} & X_{22}
\end{bmatrix}
\begin{bmatrix}
  h_1 \\
  h_2
\end{bmatrix}.
\]
Now if we choose incorrect values, \( x_1^g(0) \) and \( x_2^g(0) \) of jump variables, then we have
\[
\begin{bmatrix}
  x_{1}^g(0) - x_{1}^* \\
  x_{2}^g(0) - x_{2}^*
\end{bmatrix}
= \begin{bmatrix}
  X_{11} & X_{12} & X_{13} & X_{14} \\
  X_{21} & X_{22} & X_{23} & X_{24}
\end{bmatrix}
\begin{bmatrix}
  h_1 \\
  h_2 \\
  h_3 \\
  h_4
\end{bmatrix},
\]
\[
\begin{bmatrix}
  x_{1}(0) - x_{1}^* \\
  x_{2}(0) - x_{2}^*
\end{bmatrix}
+ \begin{bmatrix}
  x_{1}^g(0) - x_{1}(0) \\
  x_{2}^g(0) - x_{2}(0)
\end{bmatrix}
= \begin{bmatrix}
  X_{11} & X_{12} \\
  X_{21} & X_{22}
\end{bmatrix}
\begin{bmatrix}
  h_1 \\
  h_2
\end{bmatrix}
+ \begin{bmatrix}
  X_{13} & X_{14} \\
  X_{23} & X_{24}
\end{bmatrix}
\begin{bmatrix}
  h_3 \\
  h_4
\end{bmatrix},
\]
\[
\begin{bmatrix}
  h_3 \\
  h_4
\end{bmatrix}
= \begin{bmatrix}
  X_{13} & X_{14} \\
  X_{23} & X_{24}
\end{bmatrix}^{-1}
\begin{bmatrix}
  x_{1}^g(0) - x_{1}(0) \\
  x_{2}^g(0) - x_{2}(0)
\end{bmatrix}.
\]
So errors in guesses of jump variables translate into terms that explode over time and lead the corresponding path away from the steady state. The above equation can be rewritten as
\[
\begin{bmatrix}
  h_3 \\
  h_4
\end{bmatrix}
= \begin{bmatrix}
  \hat{X}_{13} & \hat{X}_{14} \\
  \hat{X}_{23} & \hat{X}_{24}
\end{bmatrix}
\begin{bmatrix}
  x_{1}^g(0) - x_{1}(0) \\
  x_{2}^g(0) - x_{2}(0)
\end{bmatrix}.
\]
Suppose, we knew \( \{ x_{1}(0), x_{2}(0) \} \). Then as we move along a circle of radius \( r \) in jump space then
\[
\begin{bmatrix}
  h_3 \\
  h_4
\end{bmatrix}
= \begin{bmatrix}
  \hat{X}_{13} & \hat{X}_{14} \\
  \hat{X}_{23} & \hat{X}_{24}
\end{bmatrix}
\begin{bmatrix}
  rCos(\theta) \\
  rSin(\theta)
\end{bmatrix},
\]
This shows that, contributions of exploding terms will get bigger as \( r \) increases for a given \( \theta \). As result, paths starting at points with a higher value of \( r \) showing greater deviation from true values jump variables will not be able to come as close to the final steady state as those starting at points with lower value of \( r \). Once again, the results carry over from the linear to the nonlinear model.

Thus, like reverse shooting, the minimum distance mapping provides a very powerful updating rule to search for the true values of the jump variables at \( t = 0 \) that is independent of the specifics of the problem. However, one should note the difference in the definition of distance mapping in
reverse and forward shooting. In reverse shooting, the mapping refers to the minimum distance of the trajectories (which are the projections of the corresponding transition paths) from the final steady values of the state variables. That is, the minimum distance is calculated in the state space. In contrast, in forward shooting, the mapping refers to the distance of the actual transition path from the final steady state. Thus, in this case, deviation of the transition path from the stable manifold is taken into account as instead of exponential tracking taking the paths close to the stable manifold (of the original system), it takes the path away from it.

From an operational point of view, however, there is not much of a difference. For example, from (*) it is clear that if vector \( \{x_1^g(0) - x_1(0), x_2^g(0) - x_2(0)\} = \{r \cos(\theta), r \sin(\theta)\} \) is orthogonal to vector \( \{\bar{X}_{13}, \bar{X}_{14}\} \), \( h_3 = 0 \) and similarly if vector \( \{x_1^g(0) - x_1(0), x_2^g(0) - x_2(0)\} \) is orthogonal to vector \( \{\bar{X}_{23}, \bar{X}_{24}\} \), \( h_4 = 0 \). Assuming that \( 0 < \lambda_3 < \lambda_4 \), the solution will explode faster if the error in guessing jump variables is in direction \( \{\bar{X}_{13}, \bar{X}_{14}\} \). Whereas it will explode relatively slowly if error is in direction \( \{\bar{X}_{23}, \bar{X}_{24}\} \). Locus of all the point with \( h_4 = 0 \) as \( r \) varies defines the Rosenbrock Curve. Thus, Rosenbrock Curve intersects the circle at two points and as we move along a circle of radius \( r \) in jump space, we will observe two local minima.\(^{22}\) Note the similarity of the results from search around the circle here when we have 2 jump variables to the case with 3 state variables. This suggests that the approach used to reverse shoot for 3-state variable case can *mutatis mutandis* be applied to forward shoot for problems with 2-jump variables.

### 6.1 Forward Shooting versus Reverse Shooting

For even systems with equal number of jump and state variables, *Ex ante*, reverse shooting clearly dominates forward shooting. For example, in a system with two jump and two state variables, reverse shooting conducts the search in one dimension along the \( \varepsilon \)-ring. On the other hand, forward shooting would conduct search in two dimension (by conducting search over many circles or ring and not just one.) Hence reverse shooting should solve the problem earlier. While this will in general be true, exceptional cases may arise where forward shooting may be preferable over reverse shooting.

\(^{22}\)In reality, when we use circle search the circle is not centered on \( \{x_1(0), x_2(0)\} \). Suppose, however, that \( \{x_1(0), x_2(0)\} \) is inside the circle. Then as we move along the circle, both \( r \) [the distance from \( \{x_1(0), x_2(0)\} \)] and \( \theta \) [angle with respect to \( \{x_1(0), x_2(0)\} \)] will change. The change in value of \( r \) will also exercise some influence on transition path but that will still be dominated by the change in \( \theta \) as we move along the circle and we will still see two minima. However, when \( \{x_1(0), x_2(0)\} \) lies out the circle around which we are searching, movement along circle will not cause \( \theta \) to take all values from 0 to \( 2\pi \) and in this case we will see only one minima.
For this we need to understand the factors that control the nature of ill-conditioning of the distance mapping. Let us, for example, consider the solution of state variables in a 2-state variable linear system. It can be put in following form after some manipulations

\[
\begin{align*}
    z_1(t) - z_1^* &= A e^{\lambda_1 t} \left( 1 + B e^{\left(\frac{\lambda_2}{\lambda_1} - 1\right) \lambda_1 t} \right), \\
    z_2(t) - z_2^* &= C e^{\lambda_1 t} \left( 1 + D e^{\left(\frac{\lambda_2}{\lambda_1} - 1\right) \lambda_1 t} \right),
\end{align*}
\]

where \( \lambda_2 < \lambda_1 < 0 \) and \( A, B, C, \) and \( D \) are some constants. If we consider a number of such two state variable systems all of which we want to solve by reverse shooting for say 99% of the transition path. Then irrespective of the eigenvalues of the systems, \( \lambda_1 t \) will be approximately the same across the models: differences in constants will exercise some influence by will be dominated by the exponential term. But then, the second term in parenthesis will smaller for models with higher value of the ratio of eigenvalues \( \lambda_2/\lambda_1 \). What that means for reverse shooting algorithm is that the trajectories will be more densely packed around the DER for models with larger \( \lambda_2/\lambda_1 \) which will require use of higher precision and hence more computational time. On the other hand, the difficulty in solving the same model by forward shooting depends on the ratio \( \lambda_4/\lambda_3 \) where \( 0 < \lambda_3 < \lambda_4 \) are the positive eigenvalues of the system. In a rare situation where \( \lambda_2/\lambda_1 >> \lambda_4/\lambda_3 \), it may be possible for forward shooting to solve the problem faster than reverse shooting despite the forward shooting conduct the search in an extra dimension.

Generally speaking, a typical system from a dynamice GE problem has equal number of jump and state variables. However, in situations where the system has smaller number of jump variables than state variables, once again forward shooting may be a viable alternative. For example, consider the model without adjustment costs in Section 2 where system consisted of \((c, k, z)\). In this case both forward shooting and reverse shooting involve search along one dimension. Similar situation may arise in higher order systems.

### 6.2 Implementation Procedure

The preliminary steps for the user are exactly same as that for Reverse Shoot 3D. The user enters the model and specifies \( T \), the duration for which solution is desired. All subsequent steps are automated. However, there are slight difference in the these steps compared to Reverse Shoot 3D.

**Step 1 (Get a candidate solution)**
• Evaluate the **Main Cell**. If the first pass through the main cell does not return a solution, then increase **precision** by four digits and re-execute the program.

As before, the program fails to generate a solution only when computations are not executed with sufficient precision. This step ends when the program has found a solution with \( t_{\text{min}} > T + 20 \) years. The solution so found is not necessarily an accurate solution.

The strategy, in next two steps, for checking the precision and accuracy of the candidate solution is slightly different from that for reverse shooting.

**Step 2 (Check accuracy of the candidate solution)**

• **Proceed to Error Checking.** Compare the candidate solution to a solution with \( t_{\text{min}} \) that is 10 years more than that for the candidate solution. If the two solutions differ significantly, keep finding solutions with higher \( t_{\text{min}} \) until the solutions for successive solutions with \( t_{\text{min}} \) differing by at least 10 years are same (i.e. differ by less than the tolerance criterion upto time \( T \)).

The idea is again based on exponential tracking.\(^\text{23}\) Exponential tracking also implies that a solution that tracks the true nonlinear saddlepath more closely will also have a larger \( t_{\text{min}} \). Comparing solutions with differing values of \( t_{\text{min}} \) is similar to comparing two solutions shown in Figure 6. Two solutions with \( t_{\text{min}} \) differing by at least 10 years will be close to each other upto time \( T \) only when they are also close to the true nonlinear saddlepath.

**Step 3 (Check numerical precision of the candidate solution)**

• **Compute a new solution with precision** increased by four digits. If the new solution differs significantly from the current candidate solution (over time \( T \)), then add another four digits to **precision** and re-run the program. Continue until successive runs yield the same solution.

\(^{23}\)The result has been proved for a class of nonlinear models by Castaneda and Rosa [?]. They call this property ‘exponential tracking.’ Consider the system 
\[
\dot{x} = Cx + G(x),
\]
where \( x \in \mathbb{R}^n \), \( C \) is a bounded linear operator and \( G : \mathbb{R}^n \to \mathbb{R}^n \) a Lipschitz continuous function with constant \( \delta \). Let the difference between the smallest positive eigenvalue and the largest negative eigenvalue (spectral gap) be greater than \( 2\delta \) and for any \( x_o \in \mathbb{R}^n \), \( \Phi(t; x_o) \) be the solution of the system passing through \( x_o \). Then there exists a unique \( x_0 \) on the unstable manifold such that
\[
|\Phi(t; x_o) - \Phi(t; x'_o)| \leq e^{(\alpha+\delta)t} |x_o - x'_o| \quad \forall t \geq 0,
\]
where \( \alpha \) is an appropriately defined constant, and \( \alpha + \delta < 0 \).
The idea behind increasing \textit{precision} to check if rounding error has distorted the solution is same as before. Further, it may be clarified that changing the order of checking for numerical precision and accuracy does not matter.

6.3 Solving Systems with 2 Jump variables

\textit{Forward Shoot 2D} solves systems with 2 jump variables. As earlier discussion shows, one conducts search around a circle with center at the current best point just like \textit{Reverse Shoot 3D}. However, there is one difference. In \textit{Reverse Shoot 3D}, the initial circle was centered at the intersection of the linear solution with the \(\delta\)-simplex. When doing forward shooting, there is no such obvious choice independent to the specifics of the model except to center the circle at the initial steady state values of the jump variables. Getting the jumps from the linearized system?

There may be an advantage in adjusting the center of the circle if problem-specific information is available but that is not necessary.

6.4 Solving Systems with 3 Jump variables

\textit{Forward Shoot 3D} solves systems with 3 jump variables. The core algorithm is like \textit{Reverse Shoot 4D}. At each step, the program conducts search on a sphere along meridians. The initial circle is centered on the values of jump variables obtained from solving the linearized system.

7 Forward Shooting with Anticipated Shock

Forward shooting algorithm of previous section works if the time \(\bar{t}\) at which the anticipated shock hits the economy is small say 3-5 years. A typical analysis of anticipated reform would have \(\bar{t}\) in this ball park. However, one may at times wrestle with problems where \(\bar{t}\) is larger. In these case, search strategy for forward shooting may encounter road block. In any case, given the spirit of approach adopted here, it is desirable to modify the approach to make it as general as possible. It is not hard to see why the simple forward shooting approach may fail.

Let us once again consider the neoclassical growth model. Assume that, the government has announced undertaking reforms at time \(\bar{t}\) that will increase productivity from \(z_o\) to \(z^*\) at \(t = \bar{t}\). As agents learn about this future shock at \(t = 0\), consumption jumps to \(c(0) > c_o\), and, up to time \(\bar{t}\), the economy follows an exploding path reaching the saddlepath \(SS\) of the economy with \(z = z^*\) (Figure 28). As exponential tracking is not at work over \((0, \bar{t})\), nothing prevents other paths to get
closer to the final steady state during period \((0, \bar{t})\). This becomes a source of problem during the initial part of the search process. This happens because sample evaluated paths (either on circle or meridians on sphere) last for very short duration during initial search. And unless \(\bar{t}\) is small this duration is less than \(\bar{t}\). Thus, the initial part of search may pick paths that are not close to the true transition path. In a sense, the minimum distance mapping loses its nice ‘monotonicity’ property and as a result has multiple local minima. It is still the case that there is unique global minimum and with value of zero. But, that is not enough to allow us to build reliable programs.

Fortunately, there is a very easy way to fix this problem: do a homotopy over the time at which the anticipated shock hits the economy. For example, in our example, the path of the productivity is

\[
z(t) = z_o + (z_* - z_o) \text{UnitStep}[t - \bar{t}],
\]

where

\[
\text{UnitStep}[x] = \begin{cases} 
1, & x \geq 0, \\
0, & x < 0.
\end{cases}
\]
To do homotopy over \( \tilde{t} \), we solve a sequence of \( n \) problems indexed by \( i \)

\[
z(t) = z_0 + (z_* - z_0) \text{Unit Step}[t - \frac{i}{n}], \quad i \in \{1, 2, \ldots, n\}.
\]

One can set \( i = 1 \) to begin with and if the program fails, \( i \) is increased until the search process produces a candidate solution in Step 1. There will always exist some large enough \( i \) so that the program will find a candidate solution. In practice, for \( \tilde{t} < 10 \), \( i \) would rarely go over 5.

The homotopy needs to be done only for finding the candidate solution in step 1. Vetting the candidate solution for precision and accuracy only requires local perturbation of the point in the jump space. Even if the current candidate solution is not pass the tests for precision and accuracy, the search for new candidate solution is also confined to the neighborhood of the current solution. Hence, except in step 1, one need not be concerned about the multiplicity of minima and use of homotopy.

8 Forward Shooting with Temporary Shock

The bare bones forward shooting algorithm worked for anticipated shock when \( \tilde{t} \) was small. However, with temporary shock, it stumbles at the block. Once again the nature of problem can be understood with reference to the neoclassical growth model. Consider a temporary increase in productivity from \( z_0 \) to \( z_* \) for time \( \tilde{t} \). The transition path of the economy is shown in Figure 29. The temporary shock generates short-run consumption boom and capital accumulation. The economy follows an unstable/explosive path of the system with \( z = z_* \) upto time \( \tilde{t} \). Thereafter, it follows the saddlepath \( SS \) back to the initial steady state.

The source of problem for the forward shooting is that the initial and the final steady states are the same—the economy is guided by a system with different steady state during transition only up to time \( \tilde{t} \). For any typical guess value \( c^g \) of \( c(0) \), the transition path starts close to the final steady state and veer away from it for a while before coming back to it. Thus, the distance of the path from the final steady state first rises and then falls and there are to value of \( t \) for which distance is minimum. The minima relevant for the minimum distance mapping is the second one with higher value of \( t \) when the path has started veering back to the final steady state. As of now it appears that this is not a problem. We can always pick the minimum with higher value of \( t \). The problem is again that the sample evaluated paths (either on circle or meridians on sphere) last for very short duration during initial search. Thus, during the initial part of search we only see the first minimum
and hence, in a sense, the minimum distance mapping loses it well behaved character: it takes the search process to the path that goes closest to the steady state at $t = 0$.

How does one salvage the situation? The basic element of the strategy has to be to avoid the minimum of a path at $t = 0$. One way to do so is to do a homotopy on the level of shock. The actual shock process is

$$z(t) = z_\ast + (z_o - z_\ast) \text{UnitStep}[t - \tilde{t}]$$

Now, considered following sequence of problems indexed by $i$

$$z(t) = z_\ast + \frac{i}{n} (z_o - z_\ast) \text{UnitStep}[t - \tilde{t}], \quad i \in \{0, 1, \ldots, n\}.$$ 

We start with $i = 0$, which is the case with permanent shock which does not suffer from the problem described above. We then withdraw the shock at $t = \tilde{t}$ in steps. With this homotopy, the program always solves a path for long enough time to see the second minimum and selects it as the value of the minimum distance mapping. It is guaranteed that for sufficiently large $n$ we can always solve a problem with temporary shock to get a candidate solution in step 1. However, compared to anticipated shock, it takes more steps to get the candidate solution. As in case of anticipated shock and for the same reasons, the homotopy needs to be done only for finding the candidate solution in step 1.
9 Forward Shooting for Unit Root Problems

The unit root problems are characterized by the fact that it is not possible to solve for the final steady state where the system ends without solving for the transition. Consider the following canonical unit root problem from international economics. The economy is small and open, the exchange rate floats, and agents hold domestic currency and foreign/traded bonds. World prices are constant and equal to unity, so the price level is one and the same as the exchange rate. The private agent solves the problem

$$\max_{c,m,i,b,F} \int_0^\infty \left[ \frac{c^{1-1/\tau}}{1-1/\tau} + \frac{m^{1-1/\tau}}{1-1/\tau} \right] e^{-\rho t} dt,$$

subject to

$$A = m + b,$$

$$\dot{A} = Y(k, z) + L + rb - c - I - \frac{n}{2} \left( \frac{I}{k} - \delta \right)^2 k - \pi m,$$

$$\dot{k} = I - \delta k,$$

$$A(0) = A_o \equiv m_o + b_o, \quad \text{and} \quad k(0) = k_o$$

As before, the path of productivity is

$$z(t) = z_o + (z^* - z_o) \left[ 1 - e^{-\omega t} \right], \quad \omega > 0.$$ 

and government’s budget constraint is given by

$$\dot{m} = L - \pi m.$$

From agent’s problem, we obtain following first order conditions

$$e^{-\frac{1}{\tau}} = \lambda_1,$$

$$\lambda_2 = \lambda_1 \left[ 1 + \psi' \left( \frac{I}{k} - \delta \right) \right],$$

$$gm^{-\frac{1}{\tau}} = \lambda_1 (r + \pi),$$

where $\lambda_1$ and $\lambda_2$ are the Lagrange multipliers on the agent’s budget constraint and capital accumulation equation. The co-state equations are

$$\dot{\lambda}_1 = (\rho - r) \lambda_1,$$

$$\dot{\lambda}_2 = (\rho + \delta) \lambda_2 - \left[ Y_k(k, z) + \psi' \left( \frac{I}{k} - \delta \right) \frac{I}{k} - \psi \left( \frac{I}{k} - \delta \right) \right] \lambda_1.$$
On substituting government’s budget constraint into the agent’s budget constraint, we obtain the economy’s current account equation: saving is associated with a current account surplus and accumulation of foreign bonds:

\[ \dot{b} = Y(k, z) + \rho b - c - I - \frac{n}{2} \left( \frac{I}{k} - \delta \right)^2 k. \]

In the \( \dot{\lambda}_1 \) equation both \( r \) and \( \rho \) are exogenous constants and need to be set equal for sensible equilibrium. Next, the first order conditions can be used to solve for

\[
\begin{align*}
  c &\equiv c(\lambda_1), \\
  I &\equiv I(\lambda_1, \lambda_2, k).
\end{align*}
\]

The core dynamic system of the model, then, apparently consists of \((\lambda_1, \lambda_2, k)\)

\[
\begin{align*}
  \dot{\lambda}_1 &= 0, \\
  \dot{\lambda}_2 &= (\rho + \delta) \lambda_2 - \left[ Y_k(k, z) + \psi' \left( \frac{I}{k} - \delta \right) \frac{I}{k} - \psi \left( \frac{I}{k} - \delta \right) \right] \lambda_1, \\
  \dot{k} &= I - \delta k,
\end{align*}
\]

and the boundary conditions for the system are

\[
\begin{align*}
  k(0) &= k_o < k^*, & \text{and} & \lim_{t \to \infty} k(t) &= k^*.
\end{align*}
\]

Though \( \lambda_1 \) does not change during transition, it can jump at \( t = 0 \). Thus, we have to guess both \( \lambda_1 \) and \( \lambda_2 \). The choice of \( \lambda_1 \) has two implications. It fixes the consumption

\[
c = c^* = c(\lambda_1),
\]

which in turn fixes the stock of foreign bonds in the final steady state as

\[
Y_k(k^*, z^*) = (\rho + \delta),
\]

gives \( k^* \) and in steady state, therefore

\[
b^* = b(\lambda_1) = \frac{c(\lambda_1) + \delta k^* - Y(k^*, z^*)}{\rho}.
\]

On the other hand, given \((\lambda_1, \lambda_2)\), the bond accumulation equation implies

\[
b^* = b(\lambda_1, \lambda_2) = b_o + \int_0^\infty \left[ Y(k, z) + \rho b - c - I - \frac{n}{2} \left( \frac{I}{k} - \delta \right)^2 k \right] dt.
\]
If we chose true value of \((\lambda_1, \lambda_2)\) at \(t = 0\), both value of \(b^*\) would be same. Clearly this cannot happen for any other guess values of \((\lambda_1, \lambda_2)\). We can recast this fact in terms of the minimum distance mapping for the augmented system

\[
\begin{align*}
\dot{\lambda}_1 &= 0, \\
\dot{\lambda}_2 &= (\rho + \delta) \lambda_2 - \left[ Y_k (k, z) + \psi' \left( \frac{I}{k} - \delta \right) \frac{I}{k} - \psi \left( \frac{I}{k} - \delta \right) \right] \lambda_1, \\
k &= I - \delta k, \\
\dot{b} &= Y(k, z) + \rho b - c - I - \frac{n}{2} \left( \frac{I}{k} - \delta \right)^2 k,
\end{align*}
\]

with boundary conditions

\[
\begin{align*}
k(0) &= k_0 < k^*, \quad \text{and} \quad \lim_{t \to \infty} k(t) = k^*, \\
b(0) &= b_0, \quad \text{and} \quad \lim_{t \to \infty} b(t) = b^*,
\end{align*}
\]

where \(b^*\) is unknown. But, if we happen to stumble upon the true values \((\lambda_1(0), \lambda_2(0))\), we will be immediately able to recognize them. \(\lambda_1(0)\) will allow us to find the \(b^*\). Then, as we solve the system we will we will converge to \((k^*, b^*)\). Whereas this will not happen for other guess values \((\lambda_{g1}, \lambda_{g2})\). Thus, if we modified the minimum distance mapping so that for a given set of guess values \((\lambda_{g1}, \lambda_{g2})\) it is the minimum distance of the resulting path from the corresponding implied steady state \((k^*, b(\lambda_{g1}))\). Two thing are clear about this modified mapping: it is well defined for all plausible guess values and it has unique global minimum of zero at \((\lambda_1(0), \lambda_2(0))\). It is not clear, however, the mapping has nice monotonicity property like the previous minimum distance mapping that would preclude presence of local minima. Thus, as the mapping is continuous, the circle-search algorithm will definitely find a local minima. Of course, if we get stuck at a local minima, we can start at a different point. However, in our work we never encountered a local minima. If one ever encounters a local minima that would be only in step 1 of the process and not in the subsequent steps as they typically only involve local perturbations as discussed before.

## 10 Unit Root and Anticipated or Temporary Shock

If a model with unit root is subject to an anticipated or temporary shock, one can apply the method outlined previously for such shock with the modification that the bond equation is included
in the system and the minimum distance mapping is redefined to the minimum distance from the corresponding steady state.