ISSN 0819-2642
ISBN 0 7340 2577 7

THE UNIVERSITY OF MELBOURNE

DEPARTMENT OF ECONOMICS

RESEARCH PAPER NUMBER 921

NOVEMBER 2004

REVERSE-SHOOTING VERSUS FORWARD-SHOOTING OVER A RANGE OF DIMENSIONALITIES

by

Ric D. Herbert
&
Peter J. Stemp

Department of Economics
The University of Melbourne
Melbourne Victoria 3010
Australia.

REVERSE-SHOOTING VERSUS FORWARD-SHOOTING OVER A RANGE OF DIMENSIONALITIES

by

Ric D. Herbert[*] and Peter J. Stemp[**]

ABSTRACT

This paper investigates the properties of dynamic solutions that have been derived using the well-known reverse-shooting and forward-shooting algorithms. Given an arbitrary large-scale model about which we have limited information, how successful are the algorithms likely to be in solving this model? We address this question using a range of investment models, both linear and non-linear. By extending the investment models to allow for multi-dimensional specifications of the capital stock, we are able to examine the computational efficiency of the competing algorithms as the dimensionality of the capital stock is allowed to increase. Our approach provides insights into how the complexity of the solutions to a broad range of macroeconomic models increases with the dimensionality of the models.

JEL classification: C63; E27

Keywords: Macroeconomics; Reverse-shooting; Forward-shooting; Saddlepath instability; Computational techniques; Investment models.

---

[*] School of Design, Communication and Information Technology, The University of Newcastle, University Drive, Callaghan, New South Wales, 2308, Australia. Email address: ric.herbert@newcastle.edu.au. Ric Herbert is grateful to The University of Newcastle for financial assistance through an Early Career Research Grant.

[**] Department of Economics, University of Melbourne, Melbourne, Victoria, 3010, Australia. Email address: pjstemp@unimelb.edu.au. Peter Stemp is grateful to the Department of Economics, University of Oxford, and to the Department of Economics, University of Washington at Seattle, for generous hospitality during his sabbatical leave and to the Faculty of Economics and Commerce at the University of Melbourne for financial support through a Faculty Research Grant.

1. INTRODUCTION

Given an arbitrary large-scale model about which we have limited information, how successful are the well-known reverse-shooting and forward-shooting algorithms likely to be in solving this model? We address this question using a series of investment models.

These models have specific properties that are common to a range of macroeconomic models. Firstly, the chosen models are derived from an optimising framework.

Secondly, the models have a number of stable and unstable trajectories so that it is likely to be complicated to solve each model for a stable solution. The economy is initially at a stable steady state equilibrium, and when shocked by, say, an exogenous change in interest rates, then it moves to a stable trajectory leading to a new steady state equilibrium. The movement to the new equilibrium is assumed to come about as a consequence of optimising behavior of the agents in the model. In each of the models, certain variables jump instantaneously after the shock, and force the model dynamics onto the trajectory leading to the stable equilibrium.

A third property of the models is that they are nonlinear with nonlinearities arising as a direct consequence of optimising behavior. The usual approach is to linearise each model in the neighborhood of the steady state and then to solve the linearised model. Of course, it is always possible to find closed-form solutions for the linearised models using matrix techniques. Such matrix solutions are likely to be more computationally efficient than solutions derived using a search algorithm. However, the solution properties derived by applying the reverse-shooting algorithm to the linearised models are also going to give an informative benchmark by providing

an indication of how successful the algorithm is likely to be in solving an arbitrary large-scale model that is "almost" linear.

The basic computational problem that we investigate is how well the reverse-shooting and forward-shooting approaches solve the example problem over a range of parameter spaces, dimensionalities and computational parameters. We are particularly interested in what is commonly referred to as Bellman's curse of dimensionality, in that we wish to investigate to what extent the computational effort required for solving the problem increases with dimensionality. In particular, we investigate the computational effort needed to solve the dynamics of both the linear and non-linear models as the numbers of stable and unstable eigenvalues are allowed to increase. We also investigate the success rates of the chosen algorithms as the dimensionality of the problem increases.

2. THE GENERAL PROBLEM

Consider the investment decision of a profit maximising firm with n types of capital along the lines of Hayashi (1982). The firm faces a Cobb-Douglas production technology. Also, adjustment costs are associated with the installation of new capital. The magnitude of these adjustment costs is governed by the magnitude of parameters, b_i. The decision of the firm can then be summarised as follows:

Choose the I_i to maximise:

$$V = \int_0^{\infty} e^{-rt}[F(K_1, K_2, ..., K_n) - \sum_{i=1}^{n} I_i]dt \qquad (1)$$

subject to

$$\dot{K}_i = I_i - b_i\left(\frac{I_i^2}{K_i}\right), \quad \text{for } i = 1, 2, ..., n \qquad (2a)$$

$$K_i(0) = K_{io}, \text{ for } i = 1,2,...,n \tag{2b}$$

$$F(K_1, K_2, ..., K_n) = a\prod_{i=1}^{n}(K_i)^{\alpha_i}, \text{ where } \sum_{i=1}^{n}\alpha_i < 1 \tag{2c}$$

where

K_i = real stock of capital of type i;

I_i = real level of investment of type i;

$F(K_1, K_2, ..., K_n)$ = real output;

r = real interest rate (assumed exogenous); and

a, b_i, α_i are exogenous parameters.

and where n is henceforth referred to as the dimensionality of the problem.

The dynamics of capital accumulation in the model reduce to the following set of equations:

$$\dot{q}_i = [r - b_i\left(\Lambda(b_i, q_i)\right)^2]q_i - F_i, \text{ for } i = 1,2,...,n \tag{3a}$$

$$\dot{K}_i = \Lambda(b_i, q_i)\left[1 - b_i\Lambda(b_i, q_i)\right]K_i, \text{ for } i = 1,2,...,n \tag{3b}$$

where

$$F_i = F_{K_i} = \frac{a\alpha_i}{K_i}\prod_{i=1}^{n}(K_i)^{\alpha_i} \tag{3c}$$

$$\Lambda(b_i, q_i) = \frac{q_i - 1}{2b_i q_i} \tag{3d}$$

The variables q_i are the co-state variables derived from the firm's optimisation problem. These co-state variables are frequently referred to as Tobin's q.

The steady state solutions of the model then reduce to the following (where an asterisk denotes the steady state value):

$$q_i^* = 1, \text{ for } i = 1,2,...,n \tag{4a}$$

$$K_i^* = \left[\frac{r}{\alpha_i a} \prod_{j \neq i} \left(\frac{\alpha_i}{\alpha_j} \right)^{\alpha_j} \right]^{\frac{1}{\sum_i \alpha_i - 1}}, \text{ for } i = 1, 2, ..., n \qquad (4b)$$

The model defined by equations (3a-3d) is non-linear. However, we can obtain a general idea about the dynamic properties of the model by linearising in the neighbourhood of the steady state. The linearised model is given by:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \\ \dot{K}_1 \\ \dot{K}_2 \\ \vdots \\ \dot{K}_n \end{bmatrix} = \begin{bmatrix} r & 0 & \cdots & 0 & -F_{11} & -F_{12} & \cdots & -F_{1n} \\ 0 & r & \cdots & 0 & -F_{21} & -F_{22} & \cdots & -F_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r & -F_{n1} & -F_{n2} & \cdots & -F_{nn} \\ \frac{K_1^*}{2b_1} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \frac{K_2^*}{2b_2} & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{K_n^*}{2b_n} & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} q_1 - 1 \\ q_2 - 1 \\ \vdots \\ q_n - 1 \\ K_1 - K_1^* \\ K_2 - K_2^* \\ \vdots \\ K_n - K_n^* \end{bmatrix} \qquad (5)$$

where $F_{ij} = F_{K_i K_j}$.

In addition, for the linearised model, the following second-order conditions for profit maximisation are satisfied:

$$K_{ii} < 0, \text{ for } i = 1, 2, ..., n \qquad (6a)$$

$$(-1)^n \det H_n > 0 \qquad (6b)$$

where

$$H_n = \begin{bmatrix} F_{11} & F_{12} & \cdots & F_{1n} \\ F_{21} & F_{22} & \cdots & F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n1} & F_{n2} & \cdots & F_{nn} \end{bmatrix}. \qquad (6c)$$

Whenever equations (6a-6c) are satisfied, it can be shown that the model given by equation (5) has precisely n stable and n unstable eigenvalues. In particular, any production technology of the form given by equation (2c) satisfies equations (6a-6c).

3. PROGRAMMING THE SOLUTION

In this paper, we will compare solutions to the model produced using the reverse-shooting and forward-shooting algorithms (Judd, 1998, pp. 355-361) as the dimensionality of the model is allowed to increase. These algorithms are compared because they form the components of a range of more sophisticated algorithms designed for specific problems and yet are sufficiently basic in their implementation so that they do not need any additional design features before they can be implemented for almost any problem.

Components of the algorithms

Both shooting algorithms have two essential components: a differential equation solver to solve for each candidate path and a search routine that chooses among possible candidate paths and determines when an acceptable candidate path has been found. To program the exercise, software components are needed to solve differential equations and undertake searches for a range of parameter sets. We used Matlab (Mathworks, 2003) as it is ideally suited for this type of computational problem. The programming was written so as to make use of key Matlab features. Library routines (toolboxes) were used so that start-of-the art solvers and searches are included in the code. Using the extensive matrix capabilities allowed for exactly the same code being executed for all dimensionalities greater than one. All results were generated using the same computer[1].

There are a number of important computational issues that will affect the solution to this problem, which is very sensitive to a whole range of approximations that are made in the solution process.

[1] Matlab 6.1 on a Dell Latitude Notebook with Pentium 3 running at 1.3 GHz and 256Mb of RAM.

Firstly, there is the parameter space. The model will be reasonably well-behaved computationally as it is an economic problem (and, thus, for example, cannot have negative capital stocks). But the parameter space will affect the size (though not the dimensionality) of the solution space. Secondly there is the choice of the differential equation solver and thus the truncation errors and ability to handle different speeds in the solution dynamics. Thirdly there is the method of searching over the candidate solution trajectories. Finally there are the definitions of "close enough" in both the solver and in the search. All these issues combine in producing errors and in producing the solution. All may increase over wider parameter spaces and dimensionalities.

Parameter calibration

To generate the results presented here, we repeatedly solved the model over a range of parameter sets. A total of 100 model repetitions were used for each dimensionality, *n*. Each model repetition differs only in the parameter calibration. For all models $a = 1$, $r_0 = 0.03$ and $r = 0.05$, and the models differ because of the values taken by α_i's and b_i's which are chosen from the following distributions:

$$\alpha_i = \frac{1}{3n} + \frac{\xi \eta_i}{2\sum \eta_i},\ i = 1, 2, ..., n \qquad (7a)$$

$$b_i = 3 + 4\delta_i,\ i = 1, 2, ..., n \qquad (7b)$$

where ξ, η_i, δ_i are each drawn from $U(0,1)$, the random uniform distribution between 0 and 1. Note that, for the nonlinear models, the α_i's and b_i's determine the extent of model nonlinearities. Hence, by employing a range of values as given by equations (7a-7b), we are able to investigate the average properties of a broad range of nonlinear models.

This choice in parameter sets produces a suite of model repetitions that have a sensible economic meaning and that are computationally well-behaved, yet give a wide-ranging parameter space. In particular, the interaction between the parameter values and the definition of the steady state value of the capital stock given by equation (4b) means that, for low dimensions, the size of the search space within that dimensionality is much larger than is the case for higher dimensions. This proposition is illustrated in Table 1.

(Table 1 about here)

ODE solver software

Solving this computational exercise is all about finding the final solution trajectory for a given model as determined by a parameter set. This final solution trajectory is a single solution to an ordinary differential equation. It is simply the solution to equation (3a-3b) from the correct set of initial conditions or equation (5) in the case of the linearised model. To find this solution trajectory, using the chosen algorithms, it is often necessary to solve thousands of ordinary differential equations. We refer to each solution of a differential equation as a candidate solution.

Basically the higher-dimensional shooting problems come down to solving many differential equations. The choice of the software component to solve the ordinary differential equations in this exercise will have considerable implications for the results. Small changes to the initial conditions of the ordinary differential equation will lead to huge differences in the final solution. For the differential equation solver we use a variable time step size Runge-Kutta method solver. This is a well-known and standard ODE solver for this type of problem. It has the key features

of robustness and accuracy, and it can cope with the problem "blowing-up". It is well suited to the type of dynamics generated by the examples chosen in this paper.

We implement the solver by calling the Matlab function *ode45*. The time step is chosen so that the local truncation error is less than 0.0001. We use a long time horizon (ranging from 0 to 1500) but use the "events" property of the Matlab ODE solver suite to stop the integration of a candidate solution so that only a small fraction of the time horizon is normally used. This, of course, significantly reduces the computational effort needed to solve the exercise. The resulting time horizon will be variable with each candidate solution. As an example of a solver stopping condition, a candidate reverse time trajectory is stopped as soon as any capital stock is greater than its corresponding initial steady state. This candidate can then be abandoned.[2]

Searcher software

Solving each computational exercise involves searching over many candidate solution trajectories to find the "correct" trajectory. From this "correct" solution comes the initial conditions required to solve the model and thus the jumps in the $q_i's$. The searcher software generates candidate solutions and stops when it finds the "correct" candidate.

For the forward shoot, a candidate ordinary differential equation is solved in forward time from a set of initial conditions at the initial steady-state values for the $K_i(0)'s$, where the $q_i(0)'s$ are allowed to vary. For the reverse shoot, a candidate ordinary differential equation is solved in reverse time from a set of terminal conditions close to the final steady state. Effectively the searcher software generates the $q_i(0)'s$ (in the case of the forward shoot) and the terminal conditions (in the case

of the reverse shoot). The searcher's software sits over the top of the ODE solver software, and generates solutions until it finds the "correct" solution. Thus the choice of the searcher software component is also important for the solution of the exercise.

For the search method we use a Nelder-Meade direct simplex search. This search has the advantage that it has memory and can go back to previous search candidates (simplex vertices) and thus is less likely to get "stuck" in a search. Unlike many other search procedures it does not require the generation (by analytic or numeric means) of derivatives. Like most searches, it works best at low dimensionalities (Lararias et al., 1998). We have found it to be a good robust searcher for this type of problem compared to other searchers we have used.

We implement the Nelder-Meade search by the Matlab function "*fminsearch*" from the Optimization Toolbox. The software is implemented by defining an objective function that is to be minimised. Like all searcher software, this function has a number of stopping conditions. These include that the objective function reaches a minimum as defined by a tolerance and within a maximum number of iterations. Alternatively, successive iterates may differ by less than a specified tolerance. Note that successful searches do not mean that the global minimum has been found.

Processing the results

For each dimensionality there are essentially four shooting experiments comprising forward and reverse-shooting for both linear and nonlinear models. For each shooting experiment and each dimensionality, the processing was completed in two steps. Step 1 was the actual shooting experiment. This comprised a long and

[2] The "greater than" comes from the fact that, the experiment considered involves an increase in r from 0.03 to 0.05 with the economy initially at the steady state associated with $r = 0.03$, so that the initial value of $\boldsymbol{K}$ is greater than its final steady-state value.

involved timing experiment involving the evaluation of a range of candidate paths interacting with the chosen search algorithm until an appropriate stopping condition was reached. In the case of reverse-shooting this also included a final forward shoot to assess the adequacy of the preferred path. In the course of running this experiment a substantial amount of data was collected. Step 2 involved the collation of these data and the assessment of "success" or "failure" for the relevant shooting experiment.

Stopping rules

In Step 1, there is a search under both reverse-shooting and forward-shooting approaches. The search has a set of stopping rules. These rules, which we will refer to as R1, for reverse-shooting, and F1, for forward-shooting, differ because the shooting algorithms are different.

Under reverse-shooting, there is a sequence of candidate paths each derived from a reverse shoot starting close to the final steady-state values of $\boldsymbol{K}$ and $\boldsymbol{q}$. The purpose of the algorithm is to find appropriate initial values for $\boldsymbol{K}$ and $\boldsymbol{q}$ on a stable path. The algorithm stops if the process has clearly failed and also if a candidate solution reaches close enough to these initial values. Closeness is measured by an appropriate norm and hence the rule R1 essentially defines the radius of a ball around the fixed initial values for $\boldsymbol{K}$ while also ensuring that the chosen candidate path is close enough to the final steady-state.

Under forward-shooting there is a sequence of candidate paths each derived from a forward shoot that starts at fixed initial values of $\boldsymbol{K}$. The purpose of the algorithm is to find a path that gets close enough to the final steady-state values of $\boldsymbol{K}$ and $\boldsymbol{q}$. Under this approach, the algorithm stops if the process has clearly failed and also if a candidate solution reaches close enough to the final steady-state. Closeness

is measured by an appropriate norm and hence the rule F1 essentially defines the radius of a ball around the final steady-state values of $\boldsymbol{K}$ and $\boldsymbol{q}$.

Measures of success

The stopping rules are very different for the two shooting algorithms. Accordingly, the choice of a candidate path under one shooting approach, does not mean that same candidate path would be accepted under the other shooting approach. It is necessary to find some measure of success that is comparable for both shooting algorithms.

This is achieved by having one final forward shoot for both reverse-shooting and forward-shooting that runs from the fixed initial value of $\boldsymbol{K}$ and the chosen initial value of $\boldsymbol{q}$ to the final steady-state values. The measure of success then assesses how close the preferred path is to the final steady-state. Closeness is measured by an appropriate norm and hence the measures of success, which we call R2 and F2, essentially define the radius of balls around the final steady-state values of $\boldsymbol{K}$ and $\boldsymbol{q}$.

The values of components of the norms are stored in Step 1 and hence different measures of success can be evaluated in Step 2 by changing the rules R2 and F2. In order that the two measures of success are comparable, it is necessary that the radius of each ball is the same or, in other words, that the measures of success, R2 and F2, are identical.

In this paper, we report results where the balls associated with the rules R1 and F1 have radius 0.0001 and where 0.1 is the radius of the balls associated with R2 and F2.

4. RESULTS

Simulations were implemented for dimensionalities of 2 to 20, where *n*, the number of capital stocks, is the dimensionality of the model. For our investment model, *n* is also the number of stable and unstable eigenvalues for the linearised model.

CPU time to solve models

The first issue we examine is the effect of dimensionality on the time it takes to solve a model. Figures 1A and 1B show the average CPU time in seconds for successful solutions to the linear and non-linear model. In all cases, the time and variability of the model solutions generally increases monotonically with dimensionality.

(Figures 1A and 1B about here)

From Figure 1A, it can be seen that reverse-shooting and forward-shooting, when successful, take about the same time to solve the linear model. Figure 1B shows that, for the nonlinear model, the solution time when using the forward-shooting approach takes substantially longer than using reverse-shooting. On the basis of timing experiments alone, it would seem that reverse-shooting is the preferred approach, at least for higher-dimensional problems.

Success rate

Figures 2A and 2B present the success rates (expressed as a percentage) for solving the model. For both linear and non-linear model, forward-shooting works best at lower dimensions, while reverse-shooting works as well or better than forward-shooting at higher dimensions.

(Figures 2A and 2B about here)

The best results are for the linear model using forward-shooting with dimensionality less than or equal to five showing close to a 100% success rate. For the nonlinear model, forward-shooting does better than reverse-shooting for dimension equal to two and three and almost as well for dimension equal to four and five.[3] Taking into account both success rates and timing experiments suggests that reverse-shooting should be the preferred approach when dimension is greater than five.

Overall assessment

The overall assessment is that forward-shooting works best at lower dimensions (dimension less than or equal to five), while reverse-shooting works better (though with clear imperfections) at higher dimensions (dimension greater than five). However, both shooting approaches struggle to successfully solve the nonlinear model for higher dimensions starting as low as four or five. Hence our results provide mixed support for both shooting algorithms.

5. CONCLUDING REMARKS

For a given n-dimensional problem, the reverse-shooting algorithm must search over an n-dimensional manifold, whereas the forward-shooting algorithm must search over a 2n-dimensional manifold. As a consequence, our initial presumption was that the reverse-shooting algorithm would always be more efficient than forward-shooting. In particular, the forward-shooting approach would be expected to require more computational effort than reverse-shooting. While generally supporting this

[3] We suspect that the reverse-shooting approach has difficulties solving the linear model for two different reasons. At low dimensionalities (1 to 5) we suspect that this is because of the large search space generated by the parameters (and described in Table 1) interacting with the search algorithm where successful searches terminated with close iterates rather than close to the global minimum. At high dimensionalities (12 and greater) we suspect this is because of the higher dimensionality.

presumption, our results demonstrate that it is possible, in some cases, particularly at lower dimensions, for the forward-shooting algorithm to have a higher success rate.

The likely reason for this outcome is the problem of compounding errors, which is likely to be greater for reverse-shooting than it is for the forward-shooting approach. One of the big problems with the reverse-shooting approach is that possible computational errors are introduced at a variety of different stages, for example in the neighbourhood of the final steady-state, through the ODE solver, and in the neighbourhood of the initial conditions. Under forward-shooting, possible computational errors are introduced at fewer stages, such as through the ODE solver and in the neighbourhood of the final steady-state, but not in the neighbourhood of the initial conditions. These computational errors have the potential to compound causing the solution trajectory to "blow-up". Our results indicate that even the introduction of "well-behaved" linearities like those introduced in this paper, can substantially reduce the effectiveness of both shooting algorithms.

It is important to note that our analysis has investigated the implications, for choice of shooting algorithms, of a particular type of complexity in macro models. The type of complexity we have investigated involves the number of stable and unstable eigenvalues. However, this is only one type of complexity that arises in macroeconomic models. Another particular type of complexity that is quite common involves the need to solve a large number of contemporaneous equations, often involving the inversion of sparse matrices. This study provides no insight into the latter problem, which is solved by completely different approaches than through the shooting algorithms analysed here. However, our approach has demonstrated how the complexity of the solutions to a broad range of macroeconomic models increases with

the number of unstable eigenvalues and provided useful insights into how techniques based on the two shooting algorithms are likely to cope as this complexity increases.

REFERENCES

Hayashi, F. (1982), "Tobin's Marginal q and Average q: a Neoclassical Interpretation," *Econometrica*, 50, pp. 213-224.

Judd, K. L. (1998), *Numerical Methods in Economics*, MIT Press, Cambridge, Massachusetts, USA.

Lararias, J. C., J. A. Reeds, H. M. Wright, and P. E. Wright, 1998, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions", *SIAM Journal of Optimization*, 9(1), pp.112-147.

MathWorks, 2003, website: www.mathworks.com,

Search Space				
	Maximum Initial Capital Stock		Maximum Final Capital Stock	
Dim	Mean	S. Deviation	Mean	S. Deviation
5	2062.9237	553.6218	160.7431	36.3833
10	73.0469	15.9201	6.4759	0.9307
15	7.4386	1.1524	1.2953	0.0555
20	3.1230	0.3415	1.0000	0.0000

Table 1: Effects of Dimensionality on Search Space

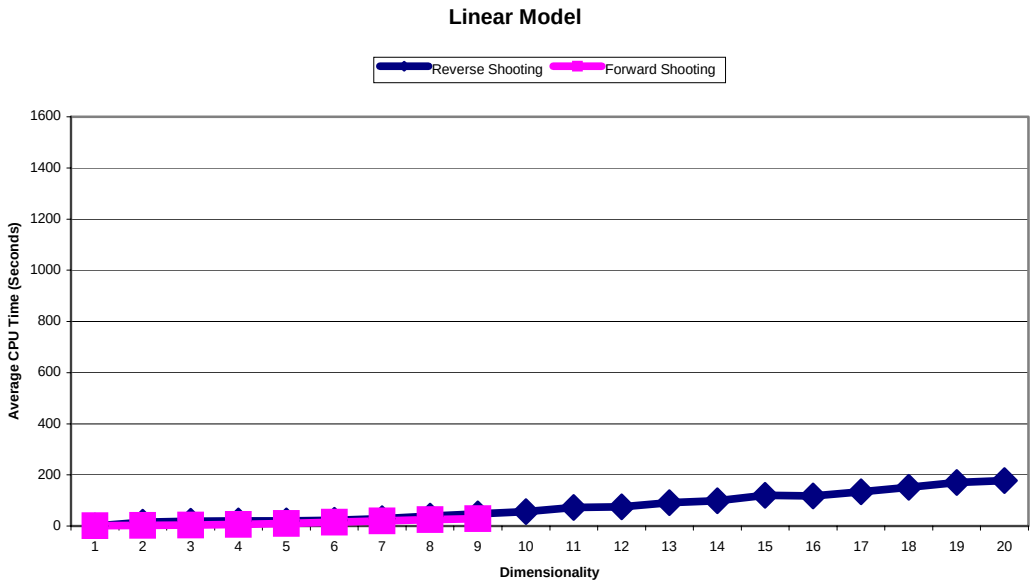

Figure 1A: Average CPU Time for Successfully Solving Linear Model

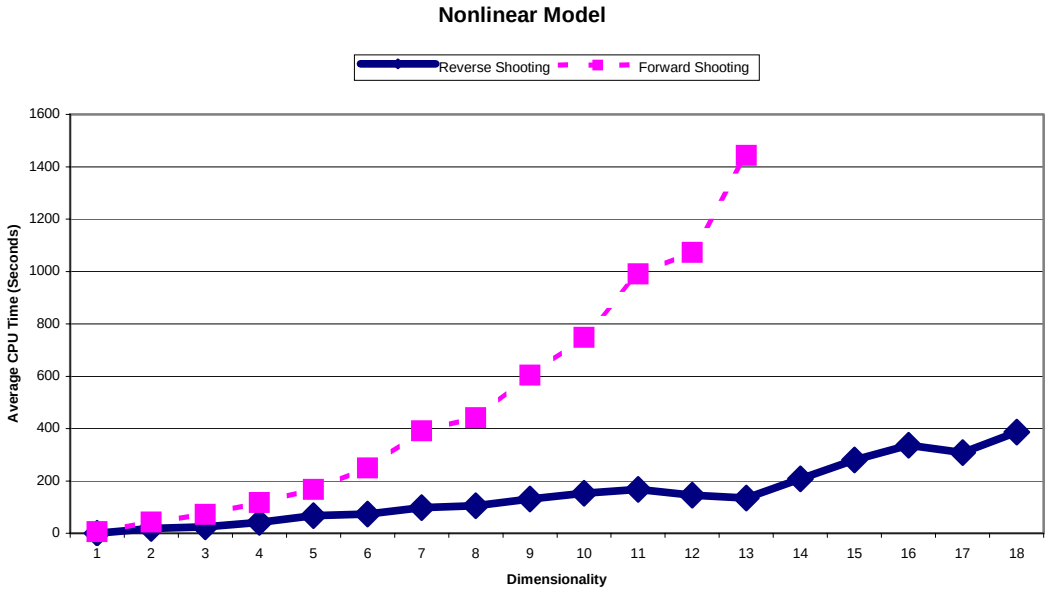

Figure 1B: Average CPU Time for Successfully Solving Nonlinear Model

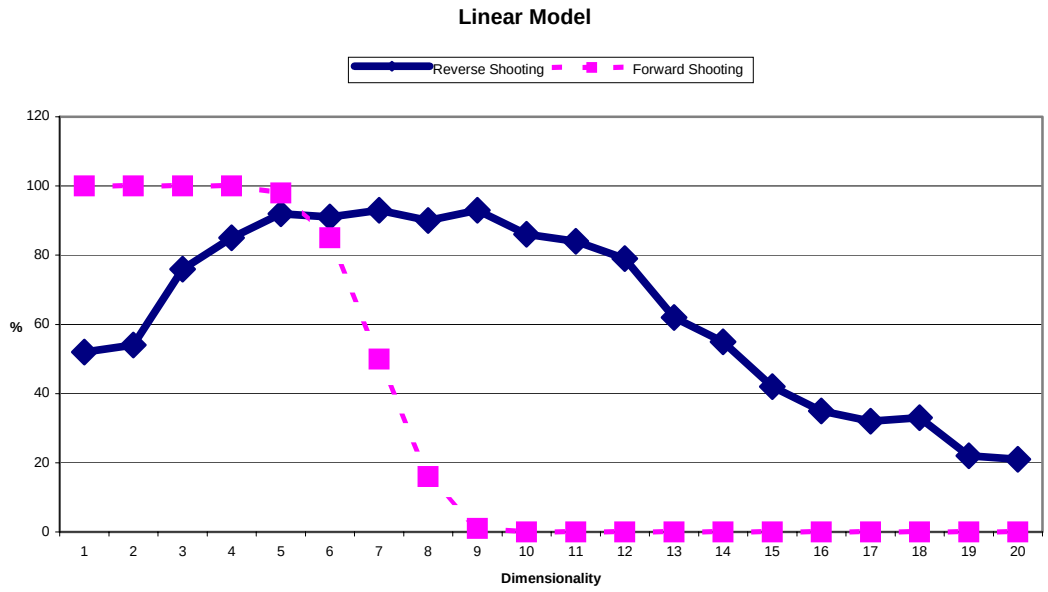

Figure 2A: Success Rates for Linear Model

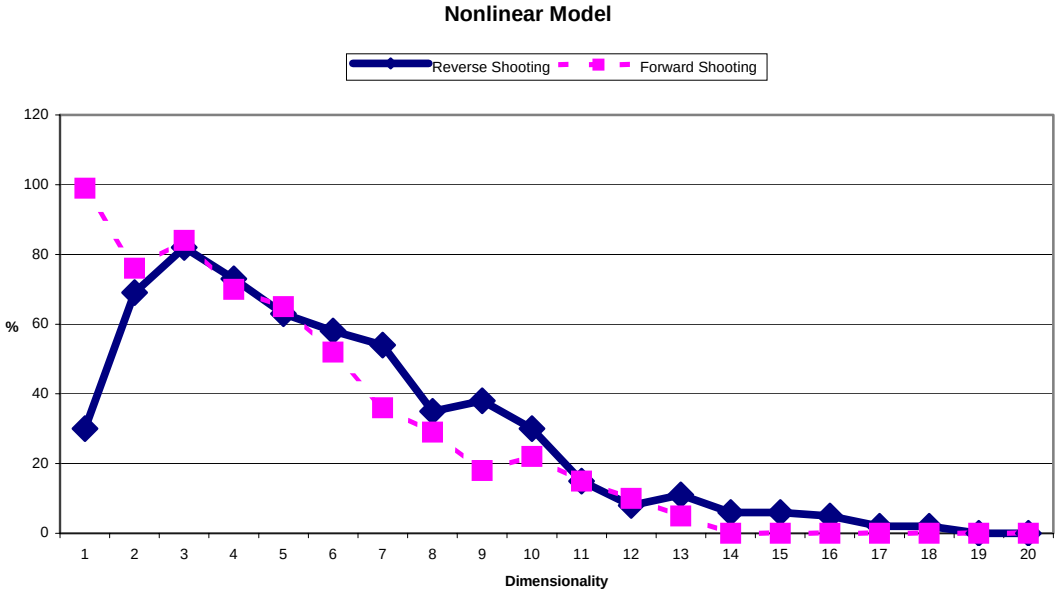

Figure 2B: Success Rates for Nonlinear Model