15. On the numerical solution of optimal control problems

AGEC 637 - Summer 2002

We will not cover this lecture this year. But I make these notes available for those that might be interested in how a program such as GAMS could be used to solve a dynamic optimization problem.

I. The direct optimization approach

When you face a complicated OC problem, it is rather unlikely that you will be able to find an exact solution analytically. In practice, therefore, such problems are typically numerically. Numerical optimization programs like GAMS are designed to solve large constrained optimization problems of the form:

$$\max_z u(z) \text{ s.t.}$$

$$f_i(z_i) \le 0 \quad i = 1, \ldots n$$

If the objective function and constraints are linear, then this is a linear programming problem. If the objective function is quadratic and constraints are linear, then it is a quadratic programming problem. If the objective function and constraints are nonlinear, then it is a nonlinear programming problem. GAMS (and a number of other programs) can solve any of these. However, it is worth noting that the numerical solution to an NLP problem may not be the global optimum so you should make a practice of checking your solution carefully to make sure that it is reasonable and not sensitive to starting values.

One of the most important steps that you must take to use the direct optimization approach is to discretize your problem. In the optimal control problems we've seen so far we have solved for z_t for all $t \in [0,T]$. Clearly a computer cannot find z at **every** point in this interval -- you must instead convert it to a discrete-time problem. This does not mean that your interval has to be 1. The interval Δ can be of any finite length you choose. The smaller Δ is, the closer your answer will be to the continuous-time problem, but the longer it will take to find the solution.

If the continuous-time problem that you are attempting to solve is

$$\boxed{\begin{aligned} &\max_z \int_0^T u(z_t, x_t, t) \text{ s.t.} \\ &\dot{x} = g(z_t, x_t, t) \end{aligned}}$$

then the discrete-time analogue would be

$$\boxed{\begin{aligned} &\max_{[z_t]_0^T, \{x_t\}_1^T} \sum_{t=0}^{T/\Delta} \tilde{u}(z_t, x_t, t) \text{ s.t.} \\ &x_{t+1} - x_t = \tilde{g}(z_t, x_t, t) \end{aligned}}$$

where $\tilde{u}(z_t, x_t, t) = \int_0^\Delta u(z_t, x_t, t)dt$ and $\tilde{g}(z_t, x_t, t) = \int_0^\Delta g(z_t, x_t, t)dt$.

Note that although x_0 is fixed, but x_1, x_2, … are somewhat flexible, although they are chosen in a sense but are bound by the state equations that govern their evolution. Hence, the x's are typically represented as choice variables in the program and a dynamic optimization problem can be written as a general constrained optimization problem in which we choose the z's and the x's, with the exception of x_0. In order to solve the

problem, we also need to specify the terminal or transversality condition which would be imposed as a final constraint on the problem.

Here's the GAMS output from a slightly simplified version of the program you'll use in PS4.

```
GAMS 2.50A    NT/W95/DOS                        02/22/00 12:26:01  PAGE      1
Optimal Economic Growth - PS3 - AGEC642, Spring 2000
    3  *
    4  *  This formulation is based on code described in 'GAMS/MINOS:
    5  *  Three examples' by Alan S. Manne, Department of Operations
    6  *  Research,Stanford University, May 1986.  The original code can be
    7  *  downloaded from the GAMS web site, http://www.gams.com/
    8  *  I have left in a lot of the program's original comments intended
    9  *  to help you understand GAMS but not necessarily relevant to OC.
   10  *---------------------------------------------------------------------
   11  * The planning horizon covers the years from TFIRST to TLAST.
   12  * The intervening asterisk indicates that this set includes
   13  * all the integers between these two values. This first statement is
   14  * the only one that needs to be changed if one wishes to examine a
   15  * different planning horizon.
   16  *
   17  * Note that the numerical solution has a positive level of the state
   18  * variable in all periods Tfirst through Tlast.  Hence, if you want to
   19  * compare to an n-period continuous-time problem with x(n)=0, you'll
   20  * need to define the horizon as zero to n-1.
   21  *---------------------------------------------------------------------
   22   SETS     T         TIME PERIODS      /0*9/
   23            TFIRST(T) FIRST PERIOD
   24            TLAST(T)  LAST PERIOD
   25  *---------------------------------------------------------------------
   26  * Data may also be entered in the form of SCALAR(S), as illustrated
   27  * below.
   28  * NOTE: USE THE CORRECT VALUES BELOW
   29  *---------------------------------------------------------------------
   30   SCALARS  R      DISCOUNT RATE             /.04 /
   31            D      INTEREST RATE             /.04 /
   32            X0     INITIAL CAPITAL          /2.00 /
   33            PERT   A PERTURBATION FACTOR    /0.05/
   34   PARAMETERS      BETA(T)  DISCOUNT FACTOR;
   35  *---------------------------------------------------------------------
   36  * The following statements make use of ORD which means that we could,
   37  * for example, change the calendar year to starting in 1999 and end in
   38  * 2019 without altering the results.  Here the symbol "$" means "such
   39  * that"; "ORD" defines the ordinal position in a set;"CARD" defines the
   40  * cardinality of the set.  Thus, TFIRST is determined by the first member
   41  * included in the set; and TLAST by the cardinality (the last member) of
   42  * the set. This seems like a roundabout way to do things, but is useful
                                                                          if
   43  * we want to be able to change the length of the planning horizon by
   44  * altering a single entry in the input data.  The same programming style
   45  * is employed when we calculate the present-value factor BETA(T) and the
   46  * output-labor vector AL(T).
   47  *---------------------------------------------------------------------
   48   TFIRST(T) = YES$(ORD(T) EQ 1);
   49   TLAST(T)  = YES$(ORD(T) EQ CARD(T));
   50   DISPLAY TFIRST, TLAST;
   51   BETA(T)  = 1/((1+R)**ORD(T));
   52  * BETA(TLAST) = BETA(TLAST)/(1-(1/(1+R)));
   53  *---------------------------------------------------------------------
   54  * Although it's not required for the problem set, I left the BETA(TLAST)
   55  * line in, though commented out, to show you how you might deal with a
   56  * problem in which you are interested in an infinite horizon.
   57  * BETA(TLAST) is a special discount factor which treats the utility in
   58  * the last period as if it were repeated forever from the Tth period
   59  * onward.
   60  *---------------------------------------------------------------------
   61   DISPLAY BETA;
   62   VARIABLES X(T)     - CAPITAL STOCK
   63             C(T)     - CONSUMPTION
   64             UTILITY - PV OF UTILITY
   65             H(T)     - HAMILTONIAN AT T
   66             HLO(T)   - HAMILTONIAN AT T WITH SMALL PERTURBATION DOWN
   67             HHI(T)   - HAMILTONIAN AT T WITH SMALL PERTURBATION UP
   68  *---------------------------------------------------------------------*
   69  * Note that variables and equations cannot be identified by the same
   70  * name. That is why the capital stock variables are called X(T), and
   71  * the capital balance equations are XX(T).
   72  *---------------------------------------------------------------------*
   73   EQUATIONS XX(T) CAPITAL BALANCE
   74             TC(T) TERMINAL CONDITION
   75             UTIL  DISCOUNTED UTILITY: OBJECTIVE FUNCTION ;
   76  *---------------------------------------------------------------------*
   77   XX(T+1)..          X(T+1) =E=  X(T) + D*X(T) -C(T);
   78   TC(TLAST)..              0 =L=  X(TLAST) + D*X(TLAST) -C(TLAST);
   79   UTIL..           UTILITY =E=  SUM(T, BETA(T)*LOG(C(T)));
   80  *---------------------------------------------------------------------
   81  * Instead of requiring that "ALL" of these constraints are to be
   82  * included, we specify that the PS2 model consists of each of the
   83  * four individual constraint types.  If, for example, we omit TC, we can
   84  * check the sensitivity of the solution to this terminal condition.
   85  *---------------------------------------------------------------------
   86   MODEL  PS2   /XX, TC, UTIL/;
   87  *---------------------------------------------------------------------
   88  * The following statements represent lower bounds on the individual
   89  * variables X(T) and C(T); a fixed value for the initial period's
   90  * capital stock, X(TFIRST); and upper bounds (absorptive capacity
   91  * constraints) on I(T). Bounds are required for X and C because
   92  * LOG(C(T)) is defined only for positive values of C and X
   93  *---------------------------------------------------------------------
   94   X.LO(T) = 0;
   95   C.LO(T) = 0.000000001; C.UP(T) = 3;
   96   X.FX(TFIRST) = X0;
   97   C.L(T) = BETA(T)*X0/(CARD(T));
   98   DISPLAY C.L;
   99  *---------------------------------------------------------------------
  100   SOLVE PS2 MAXIMIZING UTILITY USING NLP;
  101
  102  *---------------------------------------------------------------------
  103  * We now evaluate the Hamiltonian at each point in t and then perturb
  104  * the value of C by a little bit to see if it looks like a local max.
  105  * Note that in the numerical estimation we use the shadow price on the
  106  * state equation in t+1 as the estimate of lambda.
  107  *---------------------------------------------------------------------
  108   H.L(T)   = 100*(BETA(T)*LOG(C.L(T))      + XX.M(T+1)*( D*X.L(T) -C.L(T))
                                                                          );
  109
  110   C.L(T) = C.L(T)*(1-PERT);
  111   HLO.L(T) = 100*(BETA(T)*LOG(C.L(T)) + XX.M(T+1)*( D*X.L(T) -C.L(T)));
  112   C.L(T) = C.L(T)/(1-PERT);
  113   C.L(T) = C.L(T)*(1+PERT);
  114   HHI.L(T) = 100*(BETA(T)*LOG(C.L(T)) + XX.M(T+1)*( D*X.L(T) -C.L(T)));
  115
  116   DISPLAY HLO.L, H.L, HHI.L;


COMPILATION TIME     =        0.000 SECONDS    0.2 Mb      WIN-18-095
```

```
----     50 SET       TFIRST        FIRST PERIOD

0


----     50 SET       TLAST         LAST PERIOD

9


----     61 PARAMETER BETA          DISCOUNT FACTOR

0 0.962,    1 0.925,    2 0.889,    3 0.855,    4 0.822,    5 0.790,    6 0.760
7 0.731,    8 0.703,    9 0.676


----     98 VARIABLE  C.L           - CONSUMPTION

0 0.192,    1 0.185,    2 0.178,    3 0.171,    4 0.164,    5 0.158,    6 0.152
7 0.146,    8 0.141,    9 0.135


Equation Listing    SOLVE PS2 USING NLP FROM LINE 100


---- XX          =E=  CAPITAL BALANCE

XX(1)..  - 1.04*X(0) + X(1) + C(0) =E= 0 ; (LHS = -1.8877, INFES = 1.8877 ***)

XX(2)..  - 1.04*X(1) + X(2) + C(1) =E= 0 ; (LHS = 0.1849, INFES = 0.1849 ***)

XX(3)..  - 1.04*X(2) + X(3) + C(2) =E= 0 ; (LHS = 0.1778, INFES = 0.1778 ***)

REMAINING 6 ENTRIES SKIPPED


---- TC          =L=  TERMINAL CONDITION

TC(9)..  - 1.04*X(9) + C(9) =L= 0 ; (LHS = 0.1351, INFES = 0.1351 ***)


---- UTIL        =E=  DISCOUNTED UTILITY: OBJECTIVE FUNCTION

UTIL..  - (5)*C(0) - (5)*C(1) - (5)*C(2) - (5)*C(3) - (5)*C(4) - (5)*C(5)

      - (5)*C(6) - (5)*C(7) - (5)*C(8) - (5)*C(9) + UTILITY =E= 0 ;

      (LHS = 14.7009, INFES = 14.7009 ***)
```



```
---- X           - CAPITAL STOCK


X(0)
                (.LO, .L, .UP = 2, 2, 2)
      -1.04     XX(1)

X(1)
                (.LO, .L, .UP = 0, 0, +INF)
        1       XX(1)
      -1.04     XX(2)

X(2)
                (.LO, .L, .UP = 0, 0, +INF)
        1       XX(2)
      -1.04     XX(3)

REMAINING 7 ENTRIES SKIPPED


---- C           - CONSUMPTION


C(0)
                (.LO, .L, .UP = 1.0000000E-9, 0.1923, 3)
        1       XX(1)
      (-5)      UTIL

C(1)
                (.LO, .L, .UP = 1.0000000E-9, 0.1849, 3)
        1       XX(2)
      (-5)      UTIL

C(2)
                (.LO, .L, .UP = 1.0000000E-9, 0.1778, 3)
        1       XX(3)
      (-5)      UTIL

REMAINING 7 ENTRIES SKIPPED


---- UTILITY     - PV OF UTILITY


UTILITY
                (.LO, .L, .UP = -INF, 0, +INF)
        1       UTIL
```

MODEL STATISTICS

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS	11
BLOCKS OF VARIABLES	3	SINGLE VARIABLES	21
NON ZERO ELEMENTS	40	NON LINEAR N-Z	10
DERIVATIVE POOL	13	CONSTANT POOL	18
CODE LENGTH	133		

GENERATION TIME = 0.000 SECONDS 0.9 Mb WIN-18-095

EXECUTION TIME = 0.000 SECONDS 0.9 Mb WIN-18-095

S O L V E S U M M A R Y

MODEL	PS2	OBJECTIVE	UTILITY
TYPE	NLP	DIRECTION	MAXIMIZE
SOLVER	MINOS5	FROM LINE	100

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 2 LOCALLY OPTIMAL
**** OBJECTIVE VALUE -11.3557

RESOURCE USAGE, LIMIT	0.063	1000.000
ITERATION COUNT, LIMIT	11	10000
EVALUATION ERRORS	0	0

MINOS 5 Sep 9, 1998 WIN.M5.18.0 105.034.036 GAMS/MINOS 5.4

B. A. Murtagh, University of New South Wales
and
P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright
Systems Optimization Laboratory, Stanford University.

Work space allocated -- 0.04 Mb

EXIT -- OPTIMAL SOLUTION FOUND

MAJOR ITNS, LIMIT	1	200
FUNOBJ, FUNCON CALLS	30	0
SUPERBASICS	9	
INTERPRETER USAGE	0.06	
NORM RG / NORM PI	4.902E-07	

---- EQU XX CAPITAL BALANCE

	LOWER	LEVEL	UPPER	MARGINAL
1	.	.	.	3.899
2	.	.	.	3.749
3	.	.	.	3.605
4	.	.	.	3.467
5	.	.	.	3.333
6	.	.	.	3.205
7	.	.	.	3.082
8	.	.	.	2.963
9	.	.	.	2.849

---- EQU TC TERMINAL CONDITION

	LOWER	LEVEL	UPPER	MARGINAL
9	-INF	.	.	2.740

	LOWER	LEVEL	UPPER	MARGINAL
---- EQU UTIL	.	.	.	1.000

UTIL DISCOUNTED UTILITY: OBJECTIVE FUNCTION

---- VAR X - CAPITAL STOCK

	LOWER	LEVEL	UPPER	MARGINAL
0	2.000	2.000	2.000	4.055
1	.	1.833	+INF	.
2	.	1.660	+INF	.
3	.	1.480	+INF	.
4	.	1.293	+INF	.
5	.	1.098	+INF	.
6	.	0.895	+INF	.
7	.	0.684	+INF	.
8	.	0.465	+INF	.
9	.	0.237	+INF	.

---- VAR C - CONSUMPTION

	LOWER	LEVEL	UPPER	MARGINAL
0	1.0000E-9	0.247	3.000	.
1	1.0000E-9	0.247	3.000	EPS
2	1.0000E-9	0.247	3.000	EPS
3	1.0000E-9	0.247	3.000	EPS
4	1.0000E-9	0.247	3.000	EPS
5	1.0000E-9	0.247	3.000	EPS
6	1.0000E-9	0.247	3.000	EPS
7	1.0000E-9	0.247	3.000	EPS
8	1.0000E-9	0.247	3.000	EPS
9	1.0000E-9	0.247	3.000	EPS

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR UTILITY	-INF	-11.356	+INF	.

UTILITY - PV OF UTILITY

**** REPORT SUMMARY : 0 NONOPT
0 INFEASIBLE
0 UNBOUNDED
0 ERRORS

```
GAMS 2.50A    NT/W95/DOS                        02/22/00 12:26:01  PAGE      7
Optimal Economic Growth - PS3 - AGEC642, Spring 2000
E x e c u t i o n


----    116 VARIABLE  HLO.L         - HAMILTONIAN AT T WITH SMALL PERTURBATION
                                      DOWN

0 -199.704,    1 -194.521,    2 -189.538,    3 -184.746,    4 -180.139
5 -175.709,    6 -171.449,    7 -167.354,    8 -163.415,    9  -98.048


----    116 VARIABLE  H.L           - HAMILTONIAN AT T

0 -199.579,    1 -194.402,    2 -189.423,    3 -184.636,    4 -180.033
5 -175.607,    6 -171.351,    7 -167.259,    8 -163.324,    9  -94.583


----    116 VARIABLE  HHI.L         - HAMILTONIAN AT T WITH SMALL PERTURBATION
                                      UP

0 -199.696,    1 -194.513,    2 -189.531,    3 -184.739,    4 -180.132
5 -175.703,    6 -171.443,    7 -167.348,    8 -163.409,    9  -91.287


EXECUTION TIME       =        0.000 SECONDS    0.9 Mb      WIN-18-095


USER: Agricultural Economics                          G981020:1201AP-WAT
      Texas Agricultural Exp. Sta.                                DC1237

**** FILE SUMMARY

INPUT      D:\INSTRUCTION\642DYNAMICS\PS'S\RAMSEY.GMS
OUTPUT     D:\INSTRUCTION\642DYNAMICS\PS'S\RAMSEY.LST
```

II. An application of numerical optimal control - Chatterjee, Howitt and Sexton (1998)

This paper provides a nice example of applied use of dynamic optimization. The authors do a good job of setting up and motivating the problem, making the conceptual links to dynamic optimization and then clearly present results of a numerical exercise. We will discuss the paper referring to a handout with the most important that will be provided in class.

III. The backward sweep algorithm (optional)

The backward sweep algorithm (BSA) is a relatively straightforward but not terribly efficient way to solve simple one-dimensional optimal control problems. The algorithm does, however, have three pedagogical advantages: 1) it uses the maximum conditions of optimal control and it is relatively easy to program; 2) it's a good way to get started programming in Visual Basic; and 3) it demonstrates the numerical technique of searching for the solution to the problem by gradually improving our guesses.

Suppose we're interested in solving the problem:

$\max_z \int_0^T e^{-rt} u(z_t, x_t) dt$ s.t.

$\dot{x}_t = f(z_t, x_t)$

with x_T free so that our transversality condition is $\lambda_T=0$. Furthermore, suppose that for some reason it is difficult or impossible to find a closed form solution to the problem, perhaps because f is messy or the resulting differential equations cannot be solved. So we decide to use numerical methods to solve the problem instead.

To use the backward sweep algorithm you need to discretize the problem. So instead of an integral we use the discounted sum of utility from 0 to T and we have to change $\dot{x}$ equation to $x_{t+1}-x_t$. Choosing the appropriate time step is important. Sometimes we could think of this as years, but other times it might be days.

Your solution will be three vectors with z, x and λ, all from zero to T. You would also want to present the discounted value of the present value of your sum.

The BSA makes extensive use of the maximum conditions of the Hamiltonian,

A) $\frac{\partial H}{\partial z} = 0$ and

B) $\frac{\partial H}{\partial x_t} = -(\lambda_{t+1} - \lambda_1)$

The basic steps of the BSA are as follows:

1. Initialize the arrays for z, x and λ.

Find initial values for z, x and λ.

2. Make a feasible guess as to the value for z over the time horizon. The better your guess, the faster you'll get convergence and the better will be your results.

3. Set $x(0)= x_0$.
4. Calculate the vector $x_1, x_2,\ldots,x_T$ associated with your initial guess at z using the state equation.
5. Use your transversality condition to calculate λ_T. If x_T is free then this is easily found by setting $\lambda_T=0$. If x_T is not free (as in PS#3) you'll need to find the λ_T that would push x_{T+1} to zero.
6. Using the adjoint equation, B, work backwards finding $\lambda_{T-1}, \lambda_{T-2}, \ldots,\lambda_0$.

Iteratively find the optimal values of z, x and λ.

7. In finding our next estimate of z we will use solution to A. If $z(x,\lambda)$ is the value of z that solves A, then our new guess of z_0 would be a partial step toward $z(x,\lambda)$, i.e., $z_0^2 = \theta \cdot z(x_0, \lambda_0) + (1-\theta) z_0^1$ where z_0^2 is the second estimate of z_0 and z_0^1 is the first estimate. *The choice of the parameter θ is very critical! If it is too large the algorithm will blow up.*
8. Calculate x_1 using $x_{t+1}=x_t+f(z_t,x_t)$.
9. Repeat 7 and 8 until the full vector z and x are estimated.
10. Now, we sweep backwards calculating λ by using B just as in 5 and 6.
11. Repeat 10 for T-2, T–3, etc. until you get back to t=0.

Check for convergence

12. Compare your new estimate of z with the previous estimate of z. If they are very close to each other, then we say that the algorithm has converged and we jump to 14. If they are not close enough, go through the process again starting at 7. (see note on convergence below).

Scroll up and down again and again until convergence is met or until the maximum number of iterations is reached

13. Go back to 7 and repeat until a convergence criterion is met.

You're done

14. Save your results to a file. If convergence criterion has not been met, look at your output and see what's going wrong.

A. A word on convergence of numerical algorithms

Generally, the convergence criterion is some measure of the "distance" between the k^{th} and the k+1$^{\text{th}}$ iteration. For example, you may want to use $\sum_{t=0}^{T}\left(z_t^k - z_t^{k-1}\right)^2 \le \varepsilon$ or $\max_t \left|z_t^k - z_t^{k-1}\right| \le \varepsilon$. In the program below I use the , two options are available, diff1 = the difference between the z-vectors, and diff2 = the distance in the value function from one iteration to the next.

IV. Reading for next lecture

- Study for the exam
- If you are unfamiliar with excel, it would be a good idea to spend a little time playing around with the program. Also, try recording a macro (Tools Macro) then look at the program that is created. The next lecture will focus entirely on the basics of programming in Visual Basic.

Visual Basic Program for the Backward Sweep Algorithm

```
Option Explicit
Dim z() As Double, zstore() As Double, L() As Double, x() As Double
Dim Val As Double, ValOld As Double, Theta As Double, diff As Double
Dim d As Double, x0 As Double, r As Double, eps As Double, DiffOld
Dim it As Integer, iIter As Integer, maxiter As Integer, T As Integer
' I used the watch variables for debugging.  This allows me to
' look at one value rather than the full array.
Dim zwatch, xwatch, xnextwatch, lwatch
Sub BackSweep()
' ----------------------------
' Initialize values
' ----------------------------
T = 19
x0 = 3#
r = 0.03
d = 0.07
eps = 0.00000001
Theta = 0.005
maxiter = 500
' ----------------------------
' Redimension the arrays
' ----------------------------
ReDim z(T)
ReDim x(T + 1)
ReDim zstore(T)
ReDim L(T)
ReDim zstore(T)
'--------------------------------------------------------------------
'Name the ranges where we'll be writing output
'    (note that ranges start at row 3 so that I can use row 2 for
'     the 0'th entry)
'--------------------------------------------------------------------
Range("a1") = "t"
Range("a3:a100").Name = "t"
Range("b1") = "X"
Range("b3:b100").Name = "X"
Range("c1") = "Z"
Range("c3:c100").Name = "Z"
Range("d1") = "Lambda"
Range("d3:d100").Name = "Lambda"

'--------------------------------------------------------------------
'Set the initial values of x and z by looping from iT=0 to iT=T,
'arbitrarily picking the values of z, then calculating x
'Write these values to the spreadsheet.
'--------------------------------------------------------------------
 x(0) = x0
 For it = 0 To T
        xwatch = x(it)
    z(it) = x(it) / (T - it + 1)
        zwatch = z(it)
    x(it + 1) = x(it) * (1 + d) - z(it)
        xnextwatch = x(it + 1)

    Range("T").Cells(it) = it
    Range("x").Cells(it) = x(it)
    Range("z").Cells(it) = z(it)
 Next it
'
' --------------------------------------------------------------------
' Initialize values of L.  L(T) uses the transversality condition.
' Then we calculate L(t-1) using the maximum condition,
' dH/dx=-[L(t+1)-L(t)]
' --------------------------------------------------------------------
    L(T) = Exp(-r * T) / (x(T) * (1 + d))
    Range("Lambda").Cells(T) = L(T)
    For it = 1 To T
       L(T - it) = L(T - it + 1) / (1 - d)
    Range("Lambda").Cells(T - it) = L(T - it)
    Next it
' --------------------------------------------------------------------
' Enter Main portion of program
' Repeatedly sweep forward, taking a partial step from your
' current z to z that follows from the FOC.  Then sweep backward
' as we did below.
' --------------------------------------------------------------------
  For iIter = 1 To maxiter
    Range("f1") = iIter
    Call ForwardSweep
    Call BackwardSweep
    Call Convergence
    Range("e1") = "The algorithm converged in "
    Range("e2") = iIter
    Range("e3") = "iterations."

    If diff < eps Then
        Exit For
    End If
  Next iIter
' --------------------------------------------------------------------
' Once Convergence has been reached, write your results to the
' spreadsheet
' --------------------------------------------------------------------
 For it = 0 To T
    Range("x").Cells(it) = x(it)
    Range("z").Cells(it) = z(it)
    Range("x").Cells(T + 1) = x(T + 1)
    Range("Lambda").Cells(it) = L(it)
   Next it
End Sub
Sub ForwardSweep()
' --------------------------------------------------------------------
' Forward sweep
' H = exp(-r*t)*ln(c(t)) +L(t)*(x(t) - c(t))
' FOC: exp(-r*t)/z(t) = L(t)
'      z(t) = exp(-r*t)/L(t)
' State equation
' x(t+1) = x(t)*(1+d) - c(t)
' note that x(0) is not changed
' The if statement ensures that x never goes below zero
' --------------------------------------------------------------------
    For it = 0 To T
'     Use FOC to find z*
      xwatch = x(it)
      z(it) = Theta * (Exp(-r * it) / L(it)) + _
           (1 - Theta) * z(it)
      zwatch = z(it)
'     Check for feasibility
      If z(it) > x(it) * (1 + d) Then
          z(it) = (x(it) * (1 + d)) / 2
```

```
        zwatch = z(it)
        End If
'     State equation
        x(it + 1) = x(it) * (1 + d) - z(it)
        xnextwatch = x(it + 1)
    Next it
End Sub
Sub BackwardSweep()
    L(T) = 0
        L(T) = Exp(-r * T) / (x(T) * (1 + d))
    For it = 1 To T
        L(T - it) = L(T - it + 1) / (1 - d)
    lwatch = L(T - it)
    Next it
End Sub
Sub Convergence()
    ValOld = Val
    DiffOld = diff
    diff = 0#
    Val = 0#
    For it = 0 To T
      Val = Val + Log(z(it)) / ((1 + r) ^ it)
    Next it
    diff = Abs(Val - ValOld)
' Adjust theta down if steps are too big
    If diff > DiffOld Then
        Theta = Theta / 2
    Else
' Increase theta if steps are proceeding o.k.
        Theta = Theta * 1.1
    End If
End Sub
```