Overview of Matlab DICE Code
Gib Metcalf
Spring 2021

This file provides an overview of the DICE model that we'll be using in this class. It is based on Matlab code written by Prof. Derek Lemoine at the University of Arizona and used with his permission. The original code can be found in an appendix to his NBER working paper submission posted on Github.

What follows is a summary description of the various files (or scripts) used by Matlab to run the model. Numbers in parentheses are line numbers in the code. Note that many lines of the code have comments added at the end of the line (comments prefixed by %) explaining the line in question.

main_dice2016r.m

I. User Options (17 – 56)

This section sets various user options including the time step (default - 5 years), number of years to model (default – 500 years) and other options. Some of these options have been moved to user interface boxes that open when you run the program (GUI_DefineParams.m).

II. Parameterization, Error Check, and Directory Structure (57 – 213)

This loads various parameters needed to initialize the model from a script sub_parameters.m (described below). this script also defines key functions of the model. After running that script, the code runs some simple checks on savings rates (can't exceed 100 percent) and displays options.

Finally, this section of the code organizes files to write the output to a sub-directory of your current directory called output.

III. Optimization (214 – 287)

In order to run the optimization, Matlab requires an initial guess. The script sub_loadguesses.m loads the appropriate guess based on the parameter and model choices you have made. Assuming DICE is being run, guess is a 100 by 8 matrix. Each column is 100 values (500 year model and 5 year time step assumed) of the following variables:

1. Abatement rate (μ_t)
2. Savings rate (s_t)
3. Capital stock (K_t)
4. Temperature (change from baseline) for atmosphere (T_t)
5. Temperature (change from baseline) for ocean $(Tocean_t)$
6. Atmospheric carbon stock (M_t^1)
7. Upper ocean and biosphere carbon stock (M_t^2)
8. Lower ocean carbon stock (M_t^3)

Note that once the abatement and savings rate are chosen, transition equations can solve for the remaining variables. Adding them explicitly as optimizing variables speeds up the solution time.

The guess is adjusted for the time horizon and step that was actually chosen at the outset. (see sub_loadguesses)

DICE maximizes a welfare function subject to various constraints. The maximization is done with this line of code (246):

[out_controls,fval,exitflag,outputstruct,lambdastruct,gradient,hessian] = fmincon(objective,guess(:),[],[],[],[],lb(:),ub(:),nonlcon,Params.optimize_options_list2);

Let's unpack this a bit. The bracketed expression to the left of the equal sign is a list of outputs from the optimization. I'll return to this in a moment.

The actual optimization command is fmincon in the next line with its inputs in parentheses. This minimizes the *objective* function starting at an initial *guess*. (Since we want to maximize a function, we'll multiply our objective function by negative 1.) The other key inputs are *nonlcon*, which details the constraints that must be met during the optimization process and a list of optimization options.

The optimization options are provided in line 71. One of the options is to explicitly define the gradient vector (vector of first derivatives) of the objective function. Doing so can speed up the optimization dramatically. This done by specifying the option 'SpecifyObjectiveGradient' as true. The gradient vector is provided in the script that constructs the welfare function (utilityobjective.m).

Outputs from the optimization are:

1. out_controls – optimized values of the 8 variables listed in guess above
2. fval – the value of the function being optimized
3. exitflag – a flag for whether optimization was successful or not
4. outputstruct – a bunch of information about the optimization routine including number of iterations and other information that you won't need.
5. lambdastruct – shadow prices on the constraint (here the constraints in nonlcon)
6. gradient – gradient of the function evaluated at out_controls (should equal zero)
7. hessian – hessian matrix.

Assuming the optimization is successful, the key output is the matrix of out_controls (and maybe fval).

IV. Post-Processing (288 – end)

The program feeds the abatement and savings rate to **trajectory.m** which computes all variable of interest over time. (see below). It then computes welfare, emissions, and other information. The implied tax on emissions (if optimizing) is set equal to the marginal abatement cost (line 359).

The code next computes the social cost of carbon (SCC) starting in line 396. The SCC is the marginal change in welfare (measured in dollars) from a marginal increment of exogenous emissions. To understand the calculation, consider the total derivative of welfare when consumption (C) and emissions (E) exogenously change in just such a fashion that welfare is unchanged:

$$dW = \frac{\partial W}{\partial E} dE + \frac{\partial W}{\partial C} dC = 0.$$

In other words, we are making marginal changes to emissions and consumption while requiring that we stay on the iso-welfare line. Solving this for the slope of the iso-welfare line in *(E,C)* space gives:

$$\left.\frac{dC}{dE}\right|_W = -\frac{\partial W/\partial E}{\partial W/\partial C}.$$

This derivative tells you how much additional consumption you require to compensate for the loss from additional emissions. Consumption is our numeraire good (measured in dollars) and so this is precisely the SCC.

When the abatement rate is endogenous, the SCC will equal the optimal tax rate so long as the abatement rate is not binding. In DICE the abatement rate is initially bounded at 1 (cannot abate more than 100 percent of emissions). Once direct air capture is allowed, the abatement rate is bounded above at 1.2. If the abatement rate is binding, generally the SCC will exceed the tax rate (which always equals the marginal cost of abatement). This simply tells us we'd like to do more abatement if we could since the benefit of abatement (SCC) exceeds its cost.

When the abatement rate is exogenous and not optimally chosen, the tax rate reported in DICE simply reflects the marginal cost of abatement for the exogenous level of abatement. The SCC, however, will reflect the true marginal benefit of reducing emissions.

The output is saved as the new guess matrix in workspace.mat for future use and the results are written to an Excel file in the script **OutputResults.m**.

sub_parameters.m

I. Set parameters based on user options (8 – 118)

All parameters are clearly described in the code. Refer to Nordhaus (2017) or Dietz et al. (2020) for fuller explanations.

II. Functions (119 – 238)

Lemoine includes both equations required by the model as well as analytic derivatives. Writing out the derivatives explicitly speeds up the computation. I'll ignore the derivatives. But note: *if you change an equation, you must check to see if you also need to change its corresponding derivative(s).*

Here is how to read equation functions. Consider line 121:

```
Fun.Ygross = @(tfp,L,K) tfp.*((L/1000).^(1-Params.capshare)).*(K.^Params.capshare);
```

The @ sign signifies function inputs. Gross output (Ygross) is a function of tfp, labor (L), and capital (K). Next, the function is written:

$$Ygross_t = tfp_t \left(\frac{L_t}{1000}\right)^{1-capshare} K_t^{capshare}.$$

Derivatives need only be taken for endogenous variables. Gross output (Ygross) is a function of two exogenous variables (tfp and L) and one endogenous variable (K). So line 127 provides the relevant partial derivative:

$$\frac{\partial Ygross_t}{\partial K_t} = capshare \cdot \frac{Ygross_t}{K_t}.$$

Next come functions providing the exogenous transitions (134 – 143):

Population in the next period is a function of current population:

$$L_{t+1} = L_t \left(\frac{L_\infty}{L_t}\right)^{g_L},$$

where L_∞ is the long-run (asymptotic) population level and g_L is the growth rate parameter. This approach has population initially growing rapidly with the growth rate declining as it approaches the long-run level.

Next come endogenous transitions (144 – 165) and their respective derivatives (166 – 224).

Capital follows a standard capital accumulation path where next period's capital stock equals this period's capital stock (less depreciation) plus investment (145):

$$K_{t+1} = K_t(1-\delta) + Inv_t$$

Lines 167 – 8 give the derivatives of the capital accumulation equation:[1]

$$\frac{\partial K_{t+1}}{\partial K_t} = 1 - \delta$$

and

$$\frac{\partial K_{t+1}}{\partial Inv_t} = 1.$$

Lines 147 – 157 give the transition matrix for the carbon sinks (atmosphere, upper ocean and biosphere, lower ocean in DICE).

Lines 158 – 165 give forcing equations and temperature transitions

III. Exogenous trajectories (238 – end)

[1] I'm ignoring the term involving the time step.

This section constructs the data series on population, total factor productivity (TFP), emissions to output ratio prior to abatement (σ_t), and abatement cost parameter (ψ_t). These all grow exogenously and so can be set up ahead of time.

sub_loadguesses.m

Matlab requires initial guesses of the optimizing variables for the optimization routine. This script loads them.

Lines 8 – 24 loads the initial guess; the loaded values depend on the model being run. If DICE is being run, the following are loaded:

1. Abatement rate
2. Savings rate
3. Capital stock
4. Temperature (change from baseline) for atmosphere
5. Temperature (change from baseline) for ocean
6. Atmospheric carbon stock
7. Upper ocean and biosphere carbon stock
8. Lower ocean carbon stock

The guess is adjusted for the time horizon and step that was actually chosen at the outset.

Lines 26 – 76 adjusts the guess based on various model run parameters.

Lines 81 – 116 sets bounds on various parameters. Surface temperature, for example, must lie between -1 and 12 (lines 100 – 101).

If not optimizing over the abatement rate, the lower bound is set exogenously to .03 (see above) in line 92. (The upper bound is set in line 44 of sub_parameters.)

Lines 117 – 149 adjust the guess depending on whether savings is fixed or endogenous.

trajectory.m

This script actually constructs the key variables from the optimized abatement rate and savings rate. Some of these are already constructed by the optimization routine but others are not. For consistency they are all constructed here.[2] Here is how the process works.

In period 1, initial values of *K*, ***M*** (a 3 column matrix), *T*, and *Tocean* are given. (37-43)

Given initial values of those control variables, the following are computed in lines 59 – 69:

- Gross output
- Emissions

[2] Results are consistent because the script nonlcon.m of constraints used by fmincon includes the transition equations that are included in trajectory.m.

- Net output
- Abatement cost
- Investment
- Consumption

We then transit from t = 2 to the end by computing the next values of *K*, ***M***, *T*, and *Tocean* (47 – 54) and then compute all the bulleted variables above for that period.

nonlcon_utilmax.m

When optimizing, you need to provide any constraints that must be satisfied in the optimization process. This script provides those constraints. Matlab can find a solution much faster if you also provide the gradients (first derivatives) of all the constraints. (This is in addition to explicitly specifying the gradient of the objective function.)

The constraints start in line 86 and are as follows (assuming DICE formulation):

Constraint	**Equation**
1	$K_{t+1} = K_t(1-\delta) + Inv_t$
2	$T_{t+1} = f^1(T_t, Tocean_t, \boldsymbol{M}_{t+1}, t)$
3	$Tocean_{t+1} = f^2(T_t, Tocean_t)$
4	$\boldsymbol{M}_{t+1} = f^3(\boldsymbol{M}_t, E_t)$

The gradients are computed starting in line 125. They will be collected in a column vector called geq (standing for gradients of equality constraints). Gradients are required for each constraint (4) times the number of controls (8) for each period. These are required for each time period. Assuming 100 time periods, this yields a matrix with 800 rows and 400 columns that will be stacked into a matrix with 80,000 rows by 4 columns. Most of these elements will equal zero since, for example, the derivative of the constraint for *T* in period 3 with respect to *T* in period 20 is zero (current temperature unaffected by changes in future temperature).

To see how this works, consider the capital transition constraint in line 96:

$$K_{t+1} = K_t(1-\delta) + Inv_t = f^0(K_t, I_t) \quad t = 1 \ldots end-1,$$

where $f^0(\cdot)$ is given by the Matlab function *Fun.Knext(K(1:end-1,1),inv(1:end-1,1))*. Writing the constraint as

$$K_{t+1} - f^0(K_t, I_t) = 0,$$

the partial derivative with respect to K_t is

$$\frac{\partial K_{t+1}}{\partial K_t} - \frac{\partial f^0(K_t, I_t)}{\partial K_t} - \frac{\partial f^0(K_t, I_t)}{\partial I_t}\frac{\partial I_t}{\partial K_t} = -\frac{\partial f^0(K_t, I_t)}{\partial K_t} - \frac{\partial f^0(K_t, I_t)}{\partial I_t}\frac{\partial I_t}{\partial K_t},$$

since next period's capital is not explicitly a function of this period's capital (other than through the function, f^0. This constraint is provided in line 152 for periods 2 through the end. It is written slightly differently by substituting in the equation for investment:

$$-\frac{\partial f^0(K_t, I_t)}{\partial K_t} - \frac{\partial f^0(K_t, I_t)}{\partial I_t} \underbrace{s_t \left(\frac{\partial Y^{net}(K_t, T_t)}{\partial K_t} - \frac{\partial AbateCost(K_t, \mu_t)}{\partial K_t} \right)}_{\frac{\partial I_t}{\partial K_t}}$$

Note that K_t also shows up in the constraint

$$K_t - f^0(K_{t-1}, I_{t-1}) = 0 \qquad t = 2, \ldots end.$$

The derivative of this equation with respect to K_t equals 1. This gradient is contained in line 151. The gradient variable g_thisperiod refers to this constraint while g_lastperiod refers to the constraint for capital in period *t+1*.

There are gradients of the capital transition equation taken with respect to K_t, s_t, T, μ.

Lines 163 – 177 give gradients of the second constraint above (*T*) with respect to $T_t, M_t, Tocean_t$. And so on for the remaining constraints.

Lines 252 to 271 build out gradients in the proper format.

If we are not choosing the abatement rate optimally, an additional constraint (and gradient) is invoked to fix the abatement rate to .03.

For more on using analytic gradients in nonlinear constraints. search help in Matlab on "Nonlinear Constraints with Gradients."

utilityobjective.m

Begin by collecting variables (18 – 46)

Welfare is computed in line 51. It then computes the gradient of the welfare function as well.