Computational Macroeconomics Exercises

Week 4

Willi Mutschler
willi@mutschler.eu

Version: August 16, 2022

Contents

1. Numerical optimization in MATLAB **3**

2. RBC model: preprocessing in MATLAB **4**

3. RBC model: steady-state in MATLAB **8**

A. Solutions **1**

1. Numerical optimization in MATLAB

1. In a nutshell, what is the goal of numerical optimization?
2. What is an objective function, what are decision or design variables, what is a gradient, and what is a constraint?
3. What is the general idea behind gradient based optimization algorithms?
4. What is the general idea behind gradient free optimization algorithms? Name a few examples.
5. In mathematical optimization, the Rosenbrock function is a non-convex function used as a performance test problem for optimization algorithms. It is also known as Rosenbrock's valley or Rosenbrock's banana function. It is defined by:

   $$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

   The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult.

   a) Make a 3D plot of the function to see that the global minimum is at $(x, y) = (1, 1)$ where $f(x, y) = 0$.

   ```
   fsurf(@(x,y) ((1-x).^2)+(100*((y-(x.^2)).^2)),[-2 2 -2 5],'ShowContours','on'
       )
   title('Rosenbrock function')
   xlabel('x')
   ylabel('y')
   ```

   b) Draw the initial values randomly (e.g. with `randi`).

   c) Try out different optimizers in order to find the global minimum of the function.

6. In mathematical optimization, the Rastrigin function is a non-convex function used as a performance test problem for optimization algorithms. It is a typical example of non-linear multimodal function. Finding the minimum of this function is a fairly difficult problem due to its large search space and its large number of local minima. Repeat the previous exercise for the two-dimensional Rastrigin function given by:

   $$f(x, y) = 20 + x^2 + y^2 - 10(\cos(2\pi x) + \cos(2\pi y))$$

   Note that the global minimum is at $(x, y) = (0, 0)$ where $f(x, y) = 0$.

Readings

- `https://www.youtube.com/watch?v=Q2dewZweAtU&list=PLLK3oSbvdxFdF67yVxF_1FQO9SbBY3yTL`
- `https://mathworks.com/help/gads/example-comparing-several-solvers.html`

2. RBC model: preprocessing in MATLAB

Consider the basic Real Business Cycle (RBC) model with leisure and log utility function. The dynamic model equations are given Table 1, the variable description in Table 2, and the description and calibration of parameters in Table 3.

1. Create a Dynare mod file for this model and use an `initval` block to compute the steady-state.
2. Notice that Dynare's preprocessor creates a folder with a + and the name of your mod file. Inside the folder you can find different files. Briefly explain what the script files `driver.m`, `dynamic_resid.m`, `dynamic_g1.m`, `static_resid.m` and `static_g1.m` do.
3. Briefly explain Dynare's `M_.lead_lag_incidence` matrix.
4. Use MATLAB's Symbolic Math Toolbox to process the same Model without Dynare. To this end:
   a) Declare the names of the endogenous and exogenous variables as well as the parameters as string arrays called `endo_names`, `exo_names`, and `param_names`. Store the length of these vectors to variables called `endo_names_nbr`, `exo_names_nbr`, and `param_names_nbr`.
   b) Create symbolic variables in the workspace with a `_back`, `_curr`, `_fwrd`, and `_stst` suffix, using the names in `endo_names`. That is, `c_back` should be a symbolic variable denoting c_{t-1} `c_curr` should be a symbolic variable denoting c_t, `c_fwrd` should be a symbolic variable denoting c_{t+1}, and `c_stst` should be a symbolic variable denoting the steady-state c.
   c) Create symbolic variables in the workspace using the names in `exo_names`.
   d) Create symbolic variables in the workspace using the names in `param_names`.
   e) Create a symbolic vector `dynamic_model_eqs` with the dynamic model equations given in Table 1 and using the just declared convention for variables and parameters.
   f) Create the `lead_lag_incidence` matrix. Use it to distinguish the following types of variables:
      - static variables, which appear only at t, but neither at $t-1$ nor at $t+1$.
      - predetermined variables, which appear only at $t-1$, possibly at t, but not at $t+1$.
      - forward variables, which appear only at $t+1$, possibly at t, but not at $t-1$.
      - mixed variables, which appear at $t-1$ and $t+1$, and possibly at t.
      - dynamic variables, which actually appear in the dynamic model equations.
   g) Compute the static model equations using the `subs` command to substitute the dynamic variables with their corresponding name without the `_back`, `_curr`, or `_fwrd` suffix. Store the static model equations in the symbolic variable `static_model_eqs`.
   h) Compute the Jacobian of `static_model_eqs` with respect to symbolic endogenous variables.
   i) Compute the Jacobian of `dynamic_model_eqs` with respect to symbolic dynamic variables and the symbolic exogenous variables.
   j) Write out the static and dynamic model equations and Jacobians to script files:

```
write_out(static_model_eqs, 'rbc_static_resid', 'residual', true,
    dynamic_names, endo_names, exo_names, param_names);
write_out(static_g1, 'rbc_static_g1', 'g1', true, dynamic_names, endo_names,
    exo_names, param_names);
write_out(dynamic_model_eqs, 'rbc_dynamic_resid', 'residual', false,
    dynamic_names, endo_names, exo_names, param_names);
write_out(dynamic_g1, 'rbc_dynamic_g1', 'g1', false, dynamic_names,
    endo_names, exo_names, param_names);
```

The function `write_out.m` looks like this:

../progs/matlab/write_out.m

```
function occbin_write_out(Output, nameOfFunction, nameOfOutput, is_static,
    dynamic_names, endo_names, exo_names, param_names)
filename = strcat(nameOfFunction,'.m');
% Delete old version of file (if it exists)
if exist(filename,'file') > 0
    delete(filename);
end
fileID = fopen(filename,'w');
if is_static
    fprintf(fileID,'function %s = %s(steady_state,exo_vars,params)\n',
    nameOfOutput,nameOfFunction);
    fprintf(fileID,'\n%% Evaluate numerical values for static variables\n');
    for j = 1:size(endo_names,1)
        fprintf(fileID,'%s = steady_state(%d);\n',endo_names(j),j);
    end
else
    fprintf(fileID,'function %s = %s(dynamic_vars,exo_vars,params,
    steady_state)\n',nameOfOutput,nameOfFunction);
    fprintf(fileID,'\n%% Evaluate numerical values for dynamic variables\n');
    for j = 1:size(dynamic_names,1)
        fprintf(fileID,'%s = dynamic_vars(%d);\n',dynamic_names(j),j);
    end
end

fprintf(fileID,'\n%% Evaluate numerical values for exogenous variables\n');
for j = 1:size(exo_names,1)
    fprintf(fileID,'%s = exo_vars(%d);\n',exo_names(j),j);
end

fprintf(fileID,'\n%% Evaluate numerical values for parameters from params\n')
    ;
for j = 1:size(param_names,1)
    fprintf(fileID,'%s = params(%d);\n',param_names(j),j);
end

fprintf(fileID,'\n%% Evaluate numerical values for steady-state variables\n')
    ;
for j = 1:size(endo_names,1)
    fprintf(fileID,'%s_stst = steady_state(%d);\n',endo_names(j),j);
end

fprintf(fileID,'\n%% Initialize %s\n',nameOfOutput);
fprintf(fileID,'%s = zeros(%d, %d);',nameOfOutput,size(Output,1),size(Output
    ,2));

fprintf(fileID,'\n%% Evaluate non-zero entries in %s\n',nameOfOutput);
[nonzero_row,nonzero_col,nonzero_vals] = find(Output);
for j = 1:size(nonzero_vals,1)
    fprintf(fileID,'%s(%d,%d) = %s;\n',nameOfOutput,nonzero_row(j),
    nonzero_col(j),char(nonzero_vals(j)));
end

fprintf(fileID,'\nend %% function end \n');
fclose(fileID);
```

k) Make the whole script a function called `matlab_rbc_nonlinear_preprocessing.m` with an output variable MODEL, which is a structure containing information on the names and numbers of the endogenous and exogenous variables, the names and numbers of the parameters, and also the `lead_lag_incidence` matrix.

5. Compare the script files you just created with the one's from Dynare.

a) First, run dynare on the mod file to create the preprocessing files and to compute the steady-state. Store the steady-state for the endogenous, exogenous and dynamic variables:

```
[I,~]       = find(M_.lead_lag_incidence');
ys          = oo_.steady_state;
yy0         = oo_.steady_state(I);
ex0         = oo_.exo_steady_state';
```

b) Evaluate Dynare's script files at the steady-state:

```
[dynare_resid, dynare_g1] = feval([M_.fname,'.dynamic'], yy0, ex0, M_.params,
    ys, 1);
[dynare_resid_static, dynare_g1_static] = feval([M_.fname,'.static'], ys, ex0
    , M_.params);
```

c) Evaluate our just created script files at the steady-state:

```
matlab_rbc_nonlinear_preprocessing;
our_resid          = rbc_dynamic_resid(yy0,ex0,M_.params,ys);
our_resid_static   = rbc_static_resid(ys,ex0,M_.params);
our_g1             = rbc_dynamic_g1(yy0,ex0,M_.params,ys);
our_g1_static      = rbc_static_g1(ys,ex0,M_.params);
```

d) Compare the outputs.

Table 1: Dynamic Model Equations

Equation	**Description**
$\gamma\, C_t^{-1} = \gamma\, \beta\, C_{t+1}^{-1}\, (1 - \delta + R_{t+1})$	intertemporal optimality (Euler)
$W_t = -\frac{-\psi\,(1-L_t)^{-1}}{\gamma\, C_t^{-1}}$	labor supply
$K_t = (1-\delta)\, K_{t-1} + I_t$	capital accumulation
$Y_t = C_t + I_t$	market clearing
$Y_t = A_t\, K_{t-1}^{\alpha}\, L_t^{1-\alpha}$	production function
$W_t = (1-\alpha)\, \frac{Y_t}{L_t}$	labor demand
$R_t = \alpha \frac{Y_t}{K_{t-1}}$	capital demand
$\log(A_t) = \rho^A\, \log(A_{t-1}) + \varepsilon^A{}_t$	total factor productivity

Table 2: Variables

Variable	Description
Y	output
C	consumption
K	capital
L	labor
A	productivity
R	interest Rate
W	wage
I	investment
ε^A	productivity Shock

Table 3: Parameter Values

Parameter	Value	Description
β	0.990	discount factor
δ	0.025	depreciation rate
γ	1.000	consumption utility weight
ψ	1.600	labor disutility weight
α	0.350	output elasticity of capital
ρ^A	0.900	persistence technology

3. RBC model: steady-state in MATLAB

Consider the basic Real Business Cycle (RBC) model with leisure and log utility function. The dynamic model equations are given Table 1, the variable description in Table 2, and the description and calibration of parameters in Table 3.

1. Create a Dynare mod file for this model and use an `initval` block to compute the steady-state. Store the computed steady-state into a variable `stst_dyn`. Evaluate the static file at the steady-state and store your result into a variable `resid_dyn`. Compute the sum of squared residuals of `resid_dyn` and store your result into a variable `ssq_dyn`.

2. Preprocess the model using the function `matlab_rbc_nonlinear_preprocessing.m` created in the previous exercise.

3. Put the values for the parameters given in Table 3 into a vector `MODEL.params`.

4. Create two functions handles:

```
exo_vars = 0;
fun = @(xparam) rbc_static_resid(xparam,exo_vars,MODEL.params);
fun_ssq = @(xparam) sum(rbc_static_resid(xparam,exo_vars,MODEL.params).^2);
```

5. Provide initial guess values for the steady-state of all endogenous variables in a vector called `xparam0`. Also create two vectors `LB` and `UB` with a lower and upper bound for the variables.

6. Minimize either `fun` or `fun_ssq` with a numerical optimization algorithm. You might want to have a look into `fsolve`, `lsqnonlin`, `fminunc`, `fminsearch`, `fmincon`, `simulannealbnd` and `patternsearch`.

7. Compare the computed steady-state values, the residuals of the static model equations and the sum of squared residuals.

8. Compare your results with the one obtained from Dynare.

9. Compute the steady-state in closed-form (analytically) and compare with the previous exercises.

10. Redo the exercise, but now randomize your starting values by drawing them with uniformly given the lower and upper bounds (use `rand`).

A. Solutions

1 Solution to Numerical optimization in MATLAB

1. The goal of numerical optimization is to choose inputs to a function in a way that provides the best possible output. Typically, we want to maximize a function, e.g. utility, or minimize a function, e.g. costs, by adjusting the inputs to get the best output. Of course, often we are faced with limits, constraints, or boundaries on the variables. From high-school this corresponds to choose x to get the best $y = f(x)$. From calculus we know that we should take the derivative and set it to 0. For more complicated functions, however, doing this is not feasible or even impossible, so we rely on numerical optimization techniques.

2. The objective function is the value you are trying to optimize, e.g. a utility or cost function. The goal of optimization is to improve the value of the objective function, either maximize or minimize it, or bring it to a certain value. Typically, the objective function is a scalar or a vector-valued function.

   The decision or design variables are the inputs to the objective function; these are the values the optimizer is allowed to change in order to improve the objective function. You can have either one or more decision variables. The more variables, the harder it is to find the best values for these variables.

   Gradients and derivatives describe the slope of a function, i.e. whether it decreases or increases in a certain direction. Computing gradients can be achieved with analytic, numerical or automatic differentiation techniques.

   Constraints form an area in the parameter space, where the optimizer is not allowed to go to. There might be equality constraints or inequality constraints.

3. Gradient based optimizers use derivatives to find the optimal value of an objective function. There are three steps: first the gradient defines the search direction of the next iteration, second a step size is chosen, and third the convergence is checked. Gradient based optimizers tend to be widely used and scale quite well. However, they require smooth functions and the computation of derivatives might become difficult and time-consuming. Gradient based optimizers typically find local optima and heavily depend on the initial value.

4. Gradient free algorithms belong to a broad class of numerical optimization methods that do not require the computation of derivatives or gradients to optimize objective functions. These are often required as many functions do not allow to compute the gradients correctly or it is very time-consuming.

   Some examples:

   - Exhaustive search: try out every possible solution and pick the best answer.
   - Genetic algorithms: don't use just one candidate draw, but generate a population of possible solutions. Provide a score to the candidate solutions and based on this generate a new population and repeat.
   - Particle swarm: don't use just one candidate draw, but create a swarm of particles. Each particle gets a direction based on the directions of the current swarm and the overall performance. The swarm then moves towards the optimum.
   - Simulated annealing: an initial guess is taken and then we randomly draw new candidates and see whether they improve the function value. Initially we accept many bad solutions in order to visit the parameter space. As time goes on, we reduce the temperature and only accept better solutions. This works like a ball bouncing on an uneven surface.
   - Nelder Mead simplex: a simplex has a triangular shape and contains values. At each iteration the simplex flips and flops, grows and shrinks, towards its goal to focus on the optimum.

Gradient free algorithms are generally much slower than gradient based ones. However, they are usually easy to implement as no derivatives are required. Gradient free algorithms often contain a stochastic part and often there is no guarantee that we actually arrive at the optimal solution.

../progs/matlab/numerical_optimization_examples.m

5./6.
```
%% Plot Rosenbrock function
fsurf(@(x,y) ((1-x).^2)+(100*((y-(x.^2)).^2)),[-3 3 -2 5],'ShowContours','on')
title('Rosenbrock function')
xlabel('x')
ylabel('y')

%% Plot Rastrigin function
fsurf(@(x,y) 20 + x.^2 + y.^2 - 10*(cos(2*pi*x)+cos(2*pi*y)),'ShowContours','on')
title('Rastrigin function')
xlabel('x')
ylabel('y')

%% Pick function
fun = @(x) ((1-x(1)).^2)+(100*((x(2)-(x(1).^2)).^2)); % Rosenbrock
fun = @(x) 20 + x(1).^2 + x(2).^2 - 10*(cos(2*pi*x(1))+cos(2*pi*x(2))); % Rastrigin
    function

%% Initial values
x0 = randi([-5,5],2,1);
fun(x0)

%% Options for optimizer
optim_options = optimset('Display','iter','TolX',1e-7,'TolFun',1e-7,'MaxFunEvals'
    ,200000,'MaxIter',200000);
optim_options_swarm = optimoptions('particleswarm','Display','iter','TolFun',1e-7);

%% Run optimizers and store results
x = nan(2,7);
fval = nan(7,1);

[x(:,1),fval(1)] = fsolve(fun,x0,optim_options );
[x(:,2),fval(2)] = fminunc(fun,x0,optim_options );
[x(:,3),fval(3)] = fminsearch(fun,x0,optim_options);
[x(:,4),fval(4)] = fmincon(fun,x0,[],[],[],[],[],[],[],optim_options);
[x(:,5),fval(5)] = simulannealbnd(fun,x0,[],[],optim_options);
[x(:,6),fval(6)] = patternsearch(fun,x0,[],[],[],[],[],[],[],optim_options);
[x(:,7),fval(7)] = particleswarm(fun,2,[],[],optim_options_swarm);

%% Sort and display results
optim_names = ["fsolve","fminunc","fminsearch","fmincon","simulannealbnd","
    patternsearch","particleswarm"];
[~,idx_best] = sort(fval);
array2table([x(:,idx_best);fval(idx_best)'],'RowNames',{'x','y','f'},'VariableNames',
    optim_names(idx_best))
```

2 Solution to RBC model: preprocessing in MATLAB

1. `rbc_nonlinear.mod`:

../progs/dynare/rbc_nonlinear.mod

```
var
  y      ${Y}$          (long_name='output')
  c      ${C}$          (long_name='consumption')
  k      ${K}$          (long_name='capital')
  l      ${L}$          (long_name='labor')
  a      ${A}$          (long_name='productivity')
  r      ${R}$          (long_name='interest Rate')
  w      ${W}$          (long_name='wage')
  iv     ${I}$          (long_name='investment')
;

varexo
  ea  ${\varepsilon^A}$    (long_name='Productivity Shock')
;

parameters
  BETTA  ${\beta}$  (long_name='Discount Factor')
  DELT ${\delta}$ (long_name='Depreciation Rate')
  GAMA ${\gamma}$ (long_name='Consumption Utility Weight')
  PSSI   ${\psi}$   (long_name='Labor Disutility Weight')
  ALPH ${\alpha}$ (long_name='Output Elasticity of Capital')
  RHOA  ${\rho^A}$ (long_name='Persistence technology')
;

% Parameter calibration
ALPH = 0.35;
BETTA  = 0.99;
DELT = 0.025;
GAMA = 1;
PSSI   = 1.6;
RHOA  = 0.9;

model;
[name='intertemporal optimality (Euler)']
GAMA*c^(-1) = BETTA*GAMA*c(+1)^(-1)*(1-DELT+r(+1));
[name='labor supply']
w = - (-PSSI*(1-l)^(-1)) / (GAMA*c^(-1));
[name='capital accumulation']
k = (1-DELT)*k(-1) + iv;
[name='market clearing']
y = c + iv;
[name='production function']
y = a*k(-1)^ALPH*l^(1-ALPH);
[name='labor demand']
w = (1-ALPH)*y/l;
[name='capital demand']
r = ALPH*y/k(-1);
[name='total factor productivity']
log(a) = RHOA*log(a(-1)) + ea;
end;

initval;
 a = 1;
 r = 0.03;
 l = 1/3;
 y = 1.2;
 c = 0.9;
 iv = 0.35;
 k = 12;
 w = 2.25;
```

```
end;



steady;

%-----------------------------------------------------------------
% optionally: compile latex
%-----------------------------------------------------------------
write_latex_definitions;
write_latex_parameter_table;
write_latex_original_model;
collect_latex_files;

path_to_pdflatex = '';
%path_to_pdflatex = '/usr/local/bin/'; % sometimes you need to adjust the path
    where pdflatex is, depending on your operating system
if system([ path_to_pdflatex 'pdflatex -halt-on-error -interaction=batchmode ' M_.
    fname '_TeX_binder.tex'])
    warning('TeX-File did not compile; you need to compile it manually')
end
```

2. Dynare's preprocessor creates the following script files:
   - `driver.m`: the preprocessor reads the text and information you provide in your mod file and creates MATLAB code from this. The `driver.m` contains these transformations and it is MATLAB code, meaning that you can simply run it in MATLAB (e.g. hit the green run button). In this file the global structures `M_` (which contains model information), `options_` (which contains default values for options), and `oo_` (which contains results) are initialized (and some other structures as well depending on the commands you run in your mod file).
   - `dynamic_resid.m`: this is a script file with MATLAB code that evaluates the residuals of the dynamic model equations, i.e. what we get when we put everything on the right hand side and evaluate the equation for some arbitrary values of the dynamic variables and the parameters. That is dynamic files take into account that variables can appear at $t-1$, t or $t+1$.
   - `static_resid.m`: this is a script file with MATLAB code that evaluates the residuals of the static model equations. The static model equations are given when we get rid of the $t-1$, t or $t+1$ subindex of the variables. The file the computes the residuals, i.e. when we put everything on the right hand side and evaluate the static equations for some arbitrary value of the static variables and the parameters.
   - `dynamic_g1.m`: this is a script file with MATLAB code that evaluates the Jacobian of the dynamic model equations with respect to the dynamic model variables and the exogenous shock. That is, let $f(z)$ denote the model equations and z the vector of the dynamic variables and the exogenous variables, then $g1$ computes $\partial f(z)\ \partial z$.
   - `static_g1.m`: this is a script file with MATLAB code that evaluates the Jacobian of the static model equations with respect to the model variables. That is, let $\bar{f}(y)$ denote the model equations and y the vector of the model variables, then this file computes $\partial \bar{f}(x)\ \partial y$.
3. When computing the Jacobian of the dynamic model, the order of the endogenous variables in the columns is stored in `M_.lead_lag_incidence`. The rows of this matrix represent time periods: the first row denotes a lagged (time $t-1$) variable, the second row a contemporaneous (time t) variable, and the third row a leaded (time $t+1$) variable. The columns of the matrix represent the endogenous variables in their order of declaration. A zero in the matrix means that this endogenous does not appear in the model in this time period. The value in the `M_.lead_lag_incidence` matrix corresponds to the column of that variable in the Jacobian of the dynamic model.

4. `matlab_rbc_nonlinear_preprocessing.m`:

../progs/matlab/matlab_rbc_nonlinear_preprocessing.m

```
%% Basic Real Business Cycle Model
function MODEL = matlab_rbc_nonlinear_preprocessing

%% Declarations
endo_names = ["y"; "c"; "k"; "l"; "a"; "r"; "w"; "iv"];
exo_names = ["eps_a"];
param_names = ["BETTA"; "DELT"; "GAMA"; "PSSI"; "ALPH"; "RHOA"];
endo_nbr = length(endo_names);
exo_nbr = length(exo_names);
param_nbr = length(param_names);

% declare endogenous variables as symbolic
for j = 1:length(endo_names)
    syms(sym(strcat(endo_names(j), '_back')))
    syms(sym(strcat(endo_names(j), '_curr')))
    syms(sym(strcat(endo_names(j), '_fwrd')))
    syms(sym(strcat(endo_names(j), '_stst')))
end

% declare exogenous variables as symbolic
for j = 1:length(exo_names)
    syms(sym(strcat(exo_names(j))))
end

% declare parameters as symbolic
for j = 1:length(param_names)
    syms(sym(strcat(param_names(j))))
end

%% Model Equations
% intertemporal optimality (Euler)
dynamic_model_eqs(1,1) = GAMA*c_curr^(-1) - BETTA*GAMA*c_fwrd^(-1)*(1-DELT+r_fwrd)
    ;
% labor supply
dynamic_model_eqs(2,1) = w_curr + (-PSSI*(1-l_curr)^(-1)) / (GAMA*c_curr^(-1));
% capital accumulation
dynamic_model_eqs(3,1) = k_curr - (1-DELT)*k_back - iv_curr;
% market clearing
dynamic_model_eqs(4,1) = y_curr - c_curr - iv_curr;
% production function
dynamic_model_eqs(5,1) = y_curr - a_curr*k_back^ALPH*l_curr^(1-ALPH);
% labor demand
dynamic_model_eqs(6,1) = w_curr - (1-ALPH)*y_curr/l_curr;
% capital demand
dynamic_model_eqs(7,1) = r_curr - ALPH*y_curr/k_back;
% total factor productivity
dynamic_model_eqs(8,1) = log(a_curr) - RHOA*log(a_back) - eps_a;

if size(dynamic_model_eqs,1) ~= endo_nbr
    error('You need to have as many endogenous variables as model equations.')
end

%% Get dynamic variables, i.e. which variables do we actually use at t-1, t, and t
    +1

% create lead_lag_incidence matrix for dynamic variables
lead_lag_incidence = zeros(endo_nbr,3);
used_vars = symvar(dynamic_model_eqs);
for j=1:endo_nbr
    if any(has(used_vars, sym(strcat(endo_names(j),'_back'))))
        lead_lag_incidence(j,1) = 1;
    end
```

```
    if any(has(used_vars, sym(strcat(endo_names(j),'_curr'))))
        lead_lag_incidence(j,2) = 1;
    end
    if any(has(used_vars, sym(strcat(endo_names(j),'_fwrd'))))
        lead_lag_incidence(j,3) = 1;
    end
end
% this might help to understand lead_lag_incidence:
%disp(array2table(lead_lag_incidence','VariableNames',endo_names,'RowNames',{'t
    -1','t','t+1'}));

% static variables: appear only at curr, but not at back and not at fwrd
endo_static_names = endo_names(ismember(lead_lag_incidence, [0 1 0],'rows'));
% (purely) predetermined variables: appear at back but not at fwrd, possibly at
    curr
endo_pred_names = endo_names(ismember(lead_lag_incidence(:,[1 3]), [1 0],'rows'));
% (purely) forward looking variables: appear at fwrd but not at back, possibly at
    curr
endo_fwrd_names = endo_names(ismember(lead_lag_incidence(:,[1 3]), [0 1],'rows'));
% mixed variables: appear at back and fwrd, and possibly at curr
endo_mixed_names = endo_names(ismember(lead_lag_incidence(:,[1 3]), [1 1],'rows'))
    ;

dynamic_names = [strcat(endo_names(lead_lag_incidence(:,1)==1),'_back');...
                 strcat(endo_names(lead_lag_incidence(:,2)==1),'_curr');...
                 strcat(endo_names(lead_lag_incidence(:,3)==1),'_fwrd');...
                ];
dynamic_names_no_suffix = [endo_names(lead_lag_incidence(:,1)==1);...
                           endo_names(lead_lag_incidence(:,2)==1);...
                           endo_names(lead_lag_incidence(:,3)==1);...
                          ];
dynamic_vars = sym(dynamic_names);
dynamic_vars_no_suffix = sym(dynamic_names_no_suffix);
exo_vars = sym(exo_names);

% replace ones in lead_lag_incidence by number in dynamic_vars
lead_lag_incidence(lead_lag_incidence==1) = 1:nnz(lead_lag_incidence);
lead_lag_incidence = transpose(lead_lag_incidence); % transpose to make according
    to Dynare's M_.lead_lag_incidence

% this might help to understand lead_lag_incidence:
%disp(array2table(lead_lag_incidence,'VariableNames',endo_names,'RowNames',{'t
    -1','t','t+1'}));
%disp(transpose(dynamic_vars))

%% Compute static model equations
static_model_eqs = subs(dynamic_model_eqs,dynamic_vars,dynamic_vars_no_suffix);

%% Compute static Jacobians
static_g1 = jacobian(static_model_eqs,sym(endo_names));

%% Compute dynamic Jacobians
dynamic_g1 = jacobian(dynamic_model_eqs,[dynamic_vars;exo_vars]);


%% Write out to script files
write_out(static_model_eqs, 'rbc_static_resid', 'residual', true, dynamic_names,
    endo_names, exo_names, param_names);
write_out(static_g1, 'rbc_static_g1', 'g1', true, dynamic_names, endo_names,
    exo_names, param_names);

write_out(dynamic_model_eqs, 'rbc_dynamic_resid', 'residual', false, dynamic_names
    , endo_names, exo_names, param_names);
write_out(dynamic_g1, 'rbc_dynamic_g1', 'g1', false, dynamic_names, endo_names,
    exo_names, param_names);
```

```
%% Store to structure
MODEL.endo_names = endo_names;
MODEL.endo_nbr = endo_nbr;
MODEL.exo_names = exo_names;
MODEL.exo_nbr = exo_nbr;
MODEL.lead_lag_incidence = lead_lag_incidence;
MODEL.param_names = param_names;
MODEL.param_nbr = param_nbr;
```

3 Solution to RBC model: preprocessing in MATLAB

../progs/matlab/matlab_rbc_nonlinear_steadystate.m

```
%% Compute steady-state numerically
clear all;

%% from dynare
oldfolder = cd('../dynare'); % go to folder of rbc_nonlinear.mod
dynare rbc_nonlinear
stst_dyn = oo_.steady_state;
[resid_dyn] = feval([M_.fname,'.static'], stst_dyn, 0, M_.params);
ssq_dyn = sum(resid_dyn.^2);
cd(oldfolder)

%% preprocess model
MODEL = matlab_rbc_nonlinear_preprocessing;

%% calibration
ALPH  = 0.35;
BETTA = 0.99;
DELT  = 0.025;
GAMA  = 1;
PSSI  = 1.6;
RHOA  = 0.9;

% create numerical params, make sure to have same ordering as in param_names
for j=1:MODEL.param_nbr
    MODEL.params(j,1) = eval(MODEL.param_names(j));
end

%% Option : Compute the steady-state completely numerically
exo_vars(1,1) = 0; % shocks are always zero
% create function handles
fun = @(xparam) rbc_static_resid(xparam,exo_vars,MODEL.params);
fun_ssq = @(xparam) sum(rbc_static_resid(xparam,exo_vars,MODEL.params).^2);


% provide initial values in a vector, ordering as in MODEL.param_names
xparam0(1,1) = 1.2;  LB(1,1) = 0; UB(1,1) = 10;% y
xparam0(2,1) = 0.9;  LB(2,1) = 0; UB(2,1) = 10;% c
xparam0(3,1) = 12;   LB(3,1) = 0; UB(3,1) = 20;% k
xparam0(4,1) = 1/3;  LB(4,1) = 0; UB(4,1) = 10;% l
xparam0(5,1) = 1;    LB(5,1) = 0; UB(5,1) = 10;% a
xparam0(6,1) = 0.03; LB(6,1) = 0; UB(6,1) = 10;% r
xparam0(7,1) = 2.25; LB(7,1) = 0; UB(7,1) = 10;% w
xparam0(8,1) = 0.35; LB(8,1) = 0; UB(8,1) = 10;%iv
randomize_initial_values = 1;
if randomize_initial_values
    no_good = 1;
    while no_good
        xparam0 = LB + (UB-LB).*rand(length(xparam0),1);
        if all(isnan(rbc_static_resid(xparam0,exo_vars,MODEL.params))==0)
            no_good = 0;
        end
    end
end
[stst_n1,resid_n1] = fsolve(fun, xparam0, optimset('Display','iter','TolX',1e-7,'TolFun
    ',1e-7));
ssq_n1 = sum(resid_n1.^2);

[stst_n2,ssq_n2,resid_n2] = lsqnonlin(fun,xparam0,LB,UB,optimset('Display','iter','TolX
    ',1e-7,'TolFun',1e-7));

[stst_n3,ssq_n3] = fminunc(fun_ssq, xparam0, optimset('Display','iter','TolX',1e-7,'
    TolFun',1e-7));
```

```
resid_n3 = rbc_static_resid(stst_n3,exo_vars,MODEL.params);

[stst_n4,ssq_n4] = fminsearch(fun_ssq, xparam0, optimset('Display','iter','TolX',1e-7,'
    TolFun',1e-7,'MaxFunEvals',10000));
resid_n4 = rbc_static_resid(stst_n4,exo_vars,MODEL.params);

[stst_n5,ssq_n5] = fmincon(fun_ssq, xparam0, [],[],[],[],LB,UB,[],optimset('Display','
    iter','TolX',1e-7,'TolFun',1e-7,'MaxFunEvals',10000));
resid_n5 = rbc_static_resid(stst_n5,exo_vars,MODEL.params);

[stst_n6,ssq_n6] = simulannealbnd(fun_ssq, xparam0, LB,UB,optimset('Display','iter','
    TolX',1e-7,'TolFun',1e-7));
resid_n6 = rbc_static_resid(stst_n6,exo_vars,MODEL.params);

[stst_n7,ssq_n7] = patternsearch(fun_ssq, xparam0, [],[],[],[],LB,UB,[],optimset('
    Display','iter','TolX',1e-7,'TolFun',1e-7,'MaxIter',200000,'MaxFunEvals',300000));
resid_n7 = rbc_static_resid(stst_n7,exo_vars,MODEL.params);


%% Option 2: Compute the steady-state in closed-form (as much as possible)
ea = 0;
a = 1;
MC = 1;
r = 1/BETTA + DELT -1;
K_L = (MC*ALPH*a/r)^(1/(1-ALPH));
w = MC*(1-ALPH)*a*K_L^ALPH;
IV_L = DELT*K_L;
Y_L = a*(K_L)^ALPH;
C_L = Y_L - IV_L;
l = GAMA/PSSI*C_L^(-1)*w/(1+GAMA/PSSI*C_L^(-1)*w);
c  = C_L*l;
y  = Y_L*l;
iv = IV_L*l;
k  = K_L*l;

% store into array, keep same ordering as endo_names
for j=1:MODEL.endo_nbr
    stst_a(j,1) = eval(MODEL.endo_names(j));
end
resid_a = rbc_static_resid(stst_a,exo_vars,MODEL.params);
ssq_a = sum(resid_a.^2);



%% Comparison
format long

fprintf('Compare steady-states:\n')
disp(array2table([stst_a stst_dyn stst_n1 stst_n2 stst_n3 stst_n4 stst_n5 stst_n6
    stst_n7],...
                 'VariableNames',{'Analytical','Dynare initval','fsolve','lsqnonlin','
    fminunc','fminsearch','fmincon','simulannealbnd','patternsearch'},...
                 'RowNames',MODEL.endo_names));

fprintf('Compare residuals:\n')
disp(array2table([[resid_a resid_dyn resid_n1 resid_n2 resid_n3 resid_n4 resid_n5
    resid_n6 resid_n7]; ...
                  [ssq_a   ssq_dyn   ssq_n1   ssq_n2   ssq_n3   ssq_n4   ssq_n5
    ssq_n6   ssq_n7]],...
                 'VariableNames',{'Analytical','Dynare initval','fsolve','lsqnonlin','
    fminunc','fminsearch','fmincon','simulannealbnd','patternsearch'},...
                 'RowNames',[strcat('eq_',num2str(transpose(1:MODEL.endo_nbr)));"Sum Of
    Squared Residuals"]));
format short
```