Programación: MATLAB y Dynare

Diplomado de Estudios Avanzados en Macroeconomía

Felipe Martínez

Pontificia Universidad Católica de Chile

11 de julio de 2023

Agenda

Agenda

Operaciones básicas en MATLAB

Operador	Definición
+	Suma
-	Resta
*	Multiplicación matricial
.*	Multiplicación por elementos
/	División matricial por la derecha
./	División por elementos por la derecha
\	División matricial por la izquierda
.\	División por elementos por la izquierda
^	Exponenciación matricial
.^	Exponenciación por elemento

Operaciones Matriciales

Operador	Definición
A'	Transponer
[A,B]	Concatenación horizontal
[A;B]	Concatenación vertical
inv(A)	Matriz inversa de A
eye(n)	Matriz identidad de dimensión n
ones(m,n)	Matriz de ones de dimensión *mxn*
zeros(m,n)	Matriz de zeros de dimensión *mxn*

Indexación

- Sea X una matriz de *mxn*:

Operador	Definición
X(i,j)	Elemento de la fila i y columna j
X(i,:)	Fila i de la matriz X
X(:,j)	Columna j de la matriz X
X(i,s:end)	Vector de elementos desde s hasta n de la fila i
X(s:end,j)	Vector de elementos desde s hasta m de la columna j

Gráfico simple

```
%% Grafico simple
x=linspace(-2*pi,2*pi);    % grilla de valores
   para x
y=sin(x);

figure(1)
plot(x,y);

saveas(gcf,[path,'\fig1'],'epsc');  % Se guarda
   grafico en formato eps con color
```

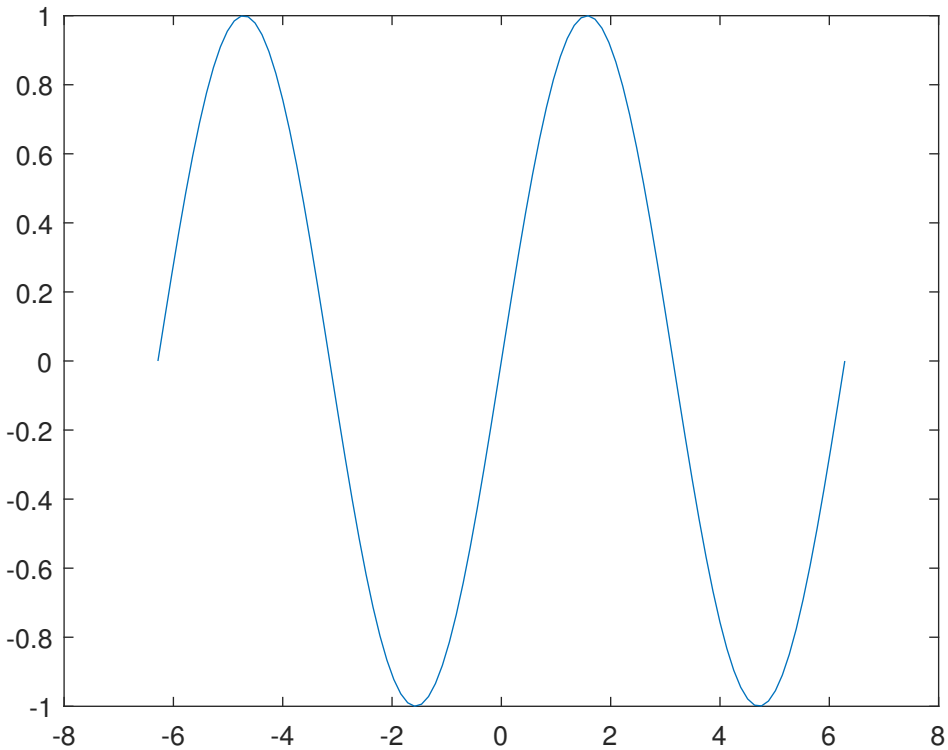


Gráfico con opciones

```
%% Grafico con opciones
figure(2)
plot(x,y,'--','LineWidth',2);

saveas(gcf,[path,'\fig2'],'epsc');
```

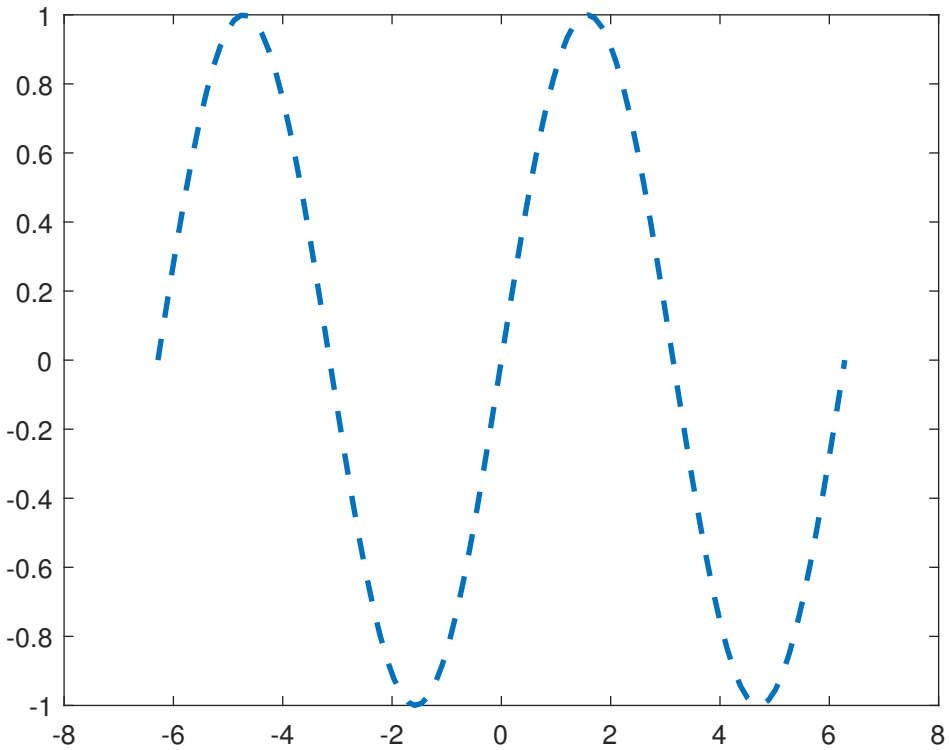


Gráfico con títulos

```
%% Grafico con titulos
figure(3)
plot(x,y,'--','LineWidth',2);
title('Grafico de ejemplo')
xlabel('x')
ylabel('y')

saveas(gcf,[path,'\fig3'],'epsc');
```

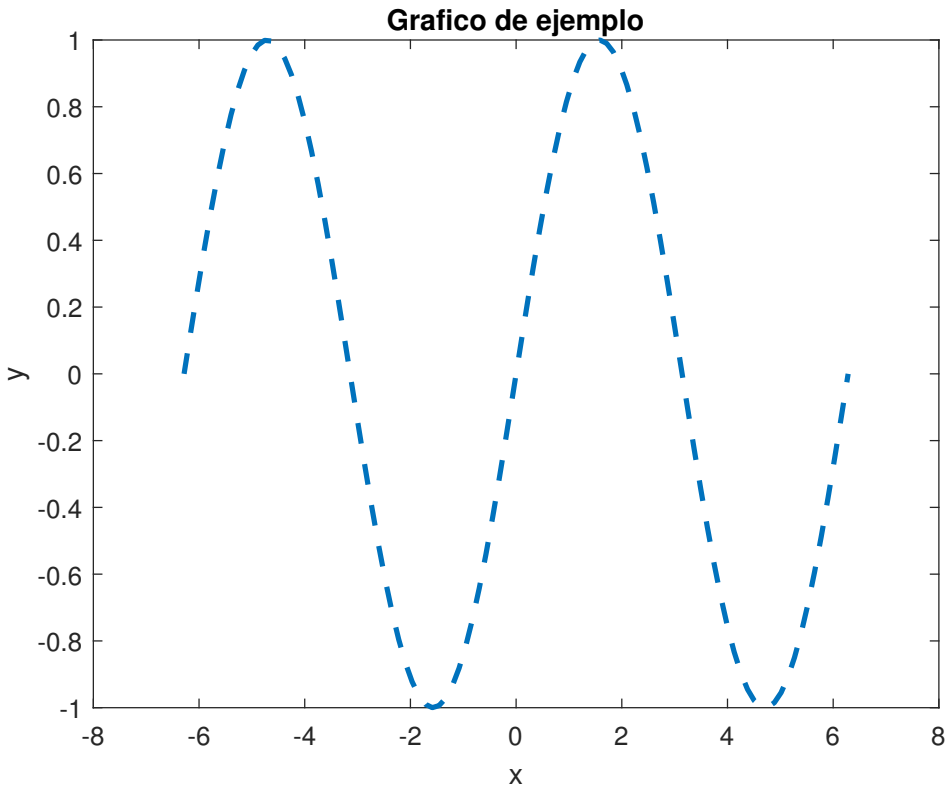


Gráfico con dos series

```
%% Graficando dos series
close all;
z=cos(x);
figure(4)
plot(x,y,'--','LineWidth',2);
hold on;
plot(x,z,'--','LineWidth',2);
hold off
title('Grafico de ejemplo');
xlabel('x');
ylabel('y');

saveas(gcf,[path,'\fig4'],'epsc');
```

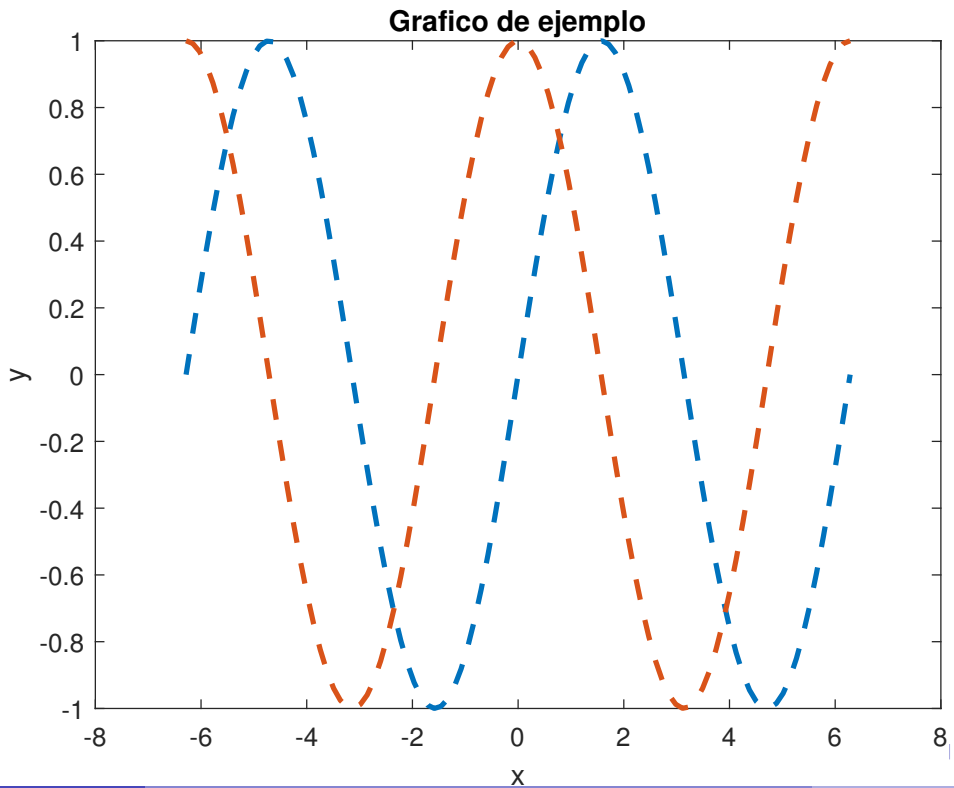


Gráfico múltiple (subplot)

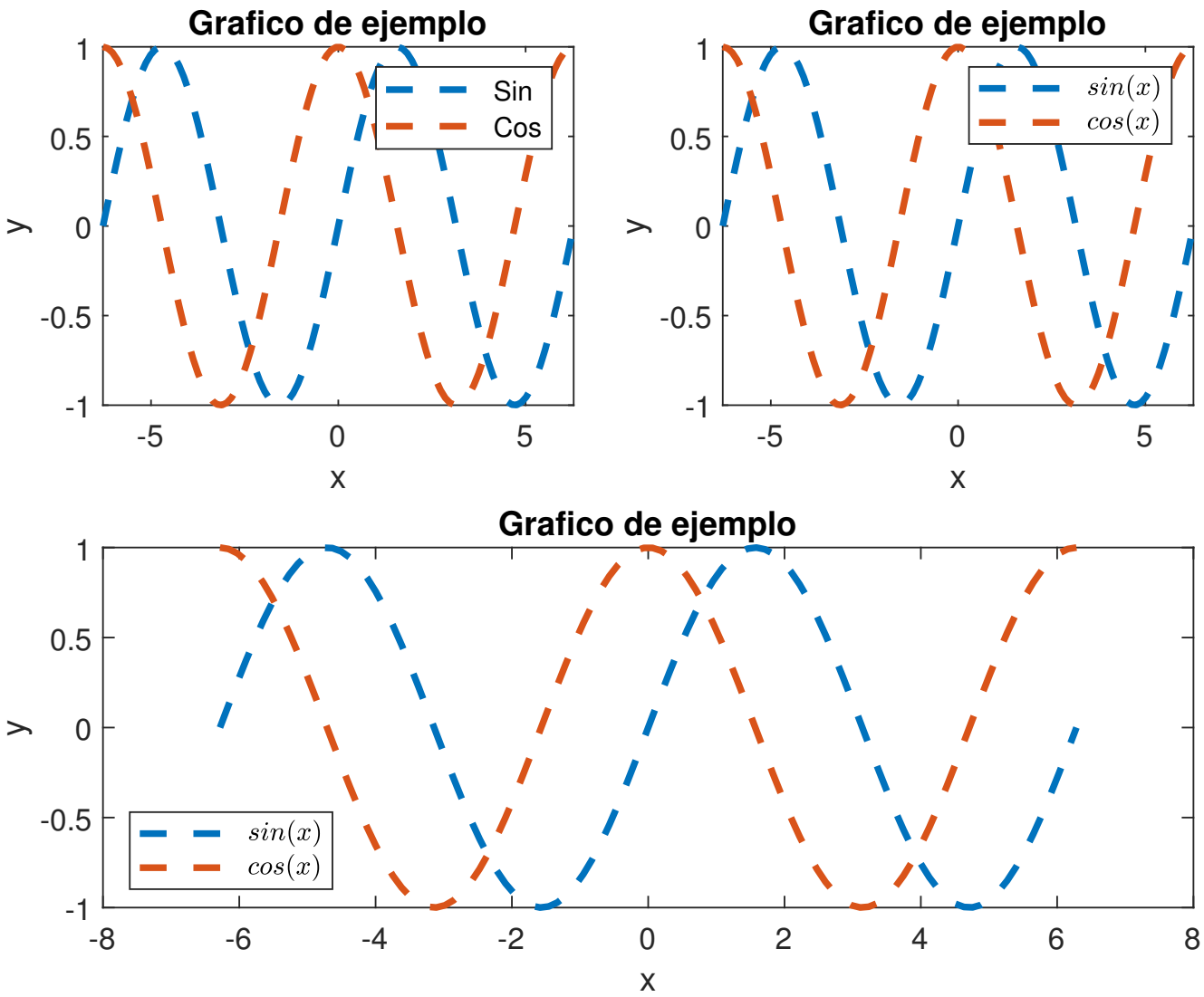
```
%% Subplot
figure(5)
subplot(2,2,1) % 4 graficos
plot(x,y,'--','LineWidth',2);
hold on;
plot(x,z,'--','LineWidth',2);
hold off
title('Grafico de ejemplo');
xlabel('x');
ylabel('y');
legend('Sin','Cos');

subplot(2,2,2);
plot(x,y,'--','LineWidth',2);
hold on;
plot(x,z,'--','LineWidth',2);
hold off
title('Grafico de ejemplo')
xlabel('x');
ylabel('y');
legend('$sin(x)$','$cos(x)$','Interpreter','latex
    ');

subplot(2,2,[3,4]); % un grafico en la parte
    inferior
plot(x,y,'--','LineWidth',2);
hold on;
plot(x,z,'--','LineWidth',2);
hold off
title('Grafico de ejemplo')
xlabel('x');
ylabel('y');
legend('$sin(x)$','$cos(x)$','Interpreter','latex
    ','Location','southwest');

saveas(gcf,[path,'\fig5'],'epsc');
```

Gráfico múltiple (subplot)



Operadores relacionales

Operador	Definición
==	Igual a
=	Distinto a
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Operadores relacionales: ejemplo

```
>> A=magic(3)
A =
8     1     6
3     5     7
4     9     2

>> A<4
ans =
0   1   0
1   0   0
0   0   1

>> A>=5
ans =
1   0   1
0   1   1
0   1   0
```

Operadores lógicos

Operador	Definición
&	AND elemento por elemento (element-wise)
&&	AND secuencial (Short-circuit)
|	OR elemento por elemento (element-wise)
||	OR secuencial (Short-circuit)
	No

- En bucle *if* o *while* & y | se utilizan como *Short-circuit* para evaluar expresiones.

Operadores lógicos: ejemplos

```
>> A=magic(3)
A =
8     1     6
3     5     7
4     9     2

>> A>5 | A<3
ans =
1   1   1
0   0   1
0   1   1
```

```
>> a=.1;

>> b=20;

>> x=(a<=.2) && (b/a>18)
x =
1

>> a=.3;

>> x=(a<=.2) && (b/a>18)
x =
0
```

Agenda

If conditional

- Permite ejecutar un grupo de comandos solo sí se cumple una determinada condición.

```
1 if conditions1
2  % statements1 if conditions1 are True
3 elseif conditions2
4  % statements2 if conditions2 are True
5 else
6  %statements3 if conditions1 and consitions2 are
      FALSE
7 end
```

If conditional: ejemplo

```
x=1;
y=0;

if x==1 & y==1
disp('x+y=2')
elseif (x==1 & y==0) | (x==0 & y==1)
disp('x+y=1')
else
disp('x+y=0')
end
```

For loop

- For es un bucle (loop) que ejecuta un **número de veces determinado** un conjunto de instrucciones.

```
for i=in:step:end
% statements
end
```

For loop: ejemplo

```
%% For loop
t=10;
x=zeros(t,2);

for i=1:10
        x(i,1)=i;
        x(i,2)=10*i+1;
end
disp(x)
```

While loop

- While es un bucle que ejecuta un conjunto de comandos. En este caso la ejecuación se realiza **tantas veces** como la condición definida sea cierta (no conocemos la cantidad de veces!!!).

```
while conditions
% statements
end
```

While loop: ejemplo

```
%% While loop
x=0;
iter=1;
while x^2<30
fprintf('iter %d x^2 %d \n',[iter x^2])
x=x+1;
iter=iter+1;
end
```

Agenda

Funciones en Matlab: dos alternativas

- m-file functions:
  - La función se guarda en un archivo .m.
  - El nombre del archivo debe ser igual al nombre de la primera fucnión en el archivo.
- Anonymous functions:
  - No se almacenan en un archivo. Se generan dentro de un código que contiene otros comandos.
  - Útiles para entregar como input una función a otra función.

M-file function: ejemplo

```
1  function [betas,sig,lower,upper]=ols_est(y,x,c)
2  % Funcion para estimar por OLS. Entrega
       parametros estimados e intervalo de
3  % confianza al 95%.
4  % Inputs:
5  %   y = variable dependiente
6  %   x = variables independientes
7  %   c = 0 estimacion sin constante
8  %     = 1 estimacion con cosntante
9
10 if c==1
11         xc=[ones(length(y),1) x];
12 else
13         xc=x;
14 end
15
16 % Estiamcion de parametros
17 betas=inv(xc'*xc)*xc'*y;
18
19 res=y-xc*betas;
20 df=length(xc)-length(betas);
21 sigma=(res'*res)/df;
22 sig=sqrt(sigma*inv(xc'*xc));
23 upper=betas+1.96*diag(sig);
24 lower=betas-1.96*diag(sig);
25
26 end
```

Agenda

Introducción

- Dynare es un software que permite trabajar con una gran variedad de modelos económicos, como DSGE o genenraciones traslapadas.
- Este spftware permite una forma amigable e intuitiva para describir estos mdelos.
- Dynare está disponible para Windows, Mac y Linux y es necesario contar con Matlab para ejecutarlo.
- Aunque es útil en muchos casos, existen ciertos modelos que no se puden resolver con Dynare.

Generalidades

- El modelo y la definición de las variables debe ser generada en un archivo .mod.
- Un archivo .mod contine los siguientes bloques:
  - **preamble**: declaración de variables y parámetros.
  - **model**: ecuaciones del modelo.
  - **steady state or initial value**: cálculo de estado estacionario por Dynare o se le entregan puntos de partida.
  - **shocks**: se definen los shocks al sistema.
  - **computation**: se definen las operaciones que debe realizar Dynare, por ejemplo, simular el modelo, obtener las IRFs, estimar el modelo, etc.

Generalidades

- En el archivo .mod:
  - Cada comando y elemento de un bloque se termina con ";". Los bloques se terminan con "end".
  - Comentarios en un línea se inician con // y se terminan al final de la línea.

```
// Comentario a la linea
var x; // Comentario sobre x
```

  - Comentarios de más de una línea se abre con "/*" y se terminan con "*/".

```
/* Comentario de más
de una línea */
```

Generalidades: declaración de variables y parámetros I

- Variables endógenas.

```
command: var VAR_NAME [$TEX_NAME$]
[(long_name=QUOTED_STRING|NAME=QUOTED_STRING)]...;
example : var c $c$ { long_name = ' Consumption
'};
```

- Variables exógeneas.

```
command varexo VAR_NAME [$TEX_NAME$]
[(long_name=QUOTED_STRING|NAME=QUOTED_STRING)...];
Example : varexo eps $epsilon$  { long_name = ' Product
shock'};
```

Generalidades: declaración de variables y parámetros II

- Parámetros.

```
command: parameters PARAM_NAME [$TEX_NAME$]
[(long_name=QUOTED_STRING|NAME=QUOTED_STRING)...];
example: alph $\alpha$  { long_name = ' Share
of labor'};
```

- Variables predeterminadas. La convención en Dynare es que el timing de la variable refleja cuando esta es decidida. Entonces en el modelo básico con capital k_t es definido en $t-1$, por lo que en Dynare, la ley de movimiento de capital sería:

```
k=(1-delta)*k(-1)+i
```

Generalidades: declaración de variables y parámetros III

- El comando "predetermined_variables" permite cambiar lo anterior:

```
predetermined\_variables k;
k(+1)=(1-delta)*k+i
```

** IMPORTANTE: En las IRFs, Dynare muestra el resultado de las variables al final del período, por lo que k contiene a la inversión de hoy y reacciona al shock.

Bloque de Modelo

- Bloque para escribir las n ecuaciones para las n variables del modelo.

```
model [OPTIONS];
eq1
eq2
end;
```

- Si el modelo ya está linealizado:

```
model(linear);
eq1
eq2
end;
```

Bloque de Modelo I

- Se pueden utilizar ecuaciones auxiliares para simplificar la escritura del modelo.

```
model;
# gamma = 1 - 1/sigma;
u1 = c1^gamma/gamma;
u2 = c2^gamma/gamma;
end;
```

lo que es equivalente a:

```
model;
u1 = c1^(1-1/sigma)/(1-1/sigma);
u2 = c2^(1-1/sigma)/(1-1/sigma);
end;
```

Bloque de Modelo II

- También podemos generar un archivo .tex con las ecuaciones del modelo utilizando:

```
command: write_latex_original_model(OPTIONS);
```

Estado Estacionario

- Dynare requiere de un estado estacionario para generar las aproximaciones.
- En general contamos con 3 formas de calcular este estado estacionario:
  - Usar initval + steady.
  - Declarar las ecuaciones del estado estacionario dentro del .mod.
  - Obtener el estado estacionario a partir de una funcuón de Matlab.

Estado Estacionario: initval

- En un modelo estocástico, el bloque *inival* provee un *guess* para el cálculo del estado estacionario.
- Inmediatamente después del bloque *initval* se escribe el comando *steady* que calculará el estado estacionario. Este comando utiliza un algoritmo iterativo de Newton no lineal.
- Para modelos complejos, el algoritmo de Dynare puede tener problemas.

```
initval;
c = 1.2;
k = 12;
x = 1;
end;

steady;
```

Estado estacionario: función de matlab I

- En caso de conocer como computar el estado estacionario, se puede generar una función en Matlab para calcularlo (es más eficiente en la resolución, pero computacionalmente más pesado) en vez de usar *steady*.

```
command: steady_state_model;
```

- Para hacerlo, si el archivo .mod es FILENAME.mod, entonces el archivo .m de llamarse FILENAME_steadystate.m

Estado estacionario: steady state block I

- En caso de poder conocer la forma analítica del estado estacionario, lo más eficiente es entregar las ecuaciones a Dynare.

```
command: steady_state_model;
eq1;
eq2;
end;
```

Estado estacionario: steady state block II

- La forma de ingresar las ecuaciones es:

```
VARIABLE_NAME = EXPRESSION;
```

o si se utiliza una función de Matlab (se puede definir varias variables a la vez):

```
[ VARIABLE_NAME, VARIABLE_NAME... ] = EXPRESSION;
```

Estado estacionario: steady state block III

- Ejemplo

```
var m P c e W R k d n l gy_obs gp_obs y dA;
varexo e_a e_m;

parameters alp bet gam mst rho psi del;

// parameter calibration, (dynamic) model declaration

steady_state_model;
dA = exp(gam);
gst = 1/dA; // A temporary variable
m = mst;

// Three other temporary variables
khst = ( (1-gst*bet*(1-del)) / (alp*gst^alp*bet) )^(1/(alp-1));
xist = ( ((khst*gst)^alp - (1-gst*(1-del))*khst)/mst )^(-1);
nust = psi*mst^2/( (1-alp)*(1-psi)*bet*gst^alp*khst^alp );
```

Estado estacionario: steady state block IV

```
n   = xist/(nust+xist);
P   = xist + nust;
k   = khst*n;

l   = psi*mst*n/( (1-psi)*(1-n) );
c   = mst/P;
d   = l - mst + 1;
y   = k^alp*n^(1-alp)*gst^alp;
R   = mst/bet;

// You can use MATLAB functions which return several arguments
[W, e] = my_function(l, n);

gp_obs = m/dA;
gy_obs = dA;
end;

steady;
```

Shocks de variables exógenas I

- En un modelo estocástico, las variables exógenas siguen un proceso aleatorio. En Dynare, estas variables se dsitribuyen $\mathcal{N}(0, \Sigma)$, donde σ^2 es definido por el usuario.
- Los valores de Σ son definidos por el usuario.
- Definición de desviación estándar de una variable:

```
var VARIABLE_NAME; stderr EXPRESSION;
```

Ejemplo:

```
var eps; stderr 0.01;
```

lo que es equivalente a:

Shocks de variables exógenas II

```
param sigmma;
varexo eps;
model;
a=rho*a(-a)+sigmma*eps
var eps; stderr 1;
```

- Especificando la varianza de la variable:

```
var VARIABLE_NAME = EXPRESSION;
```

- La covarianza entre dos variables:

```
var VARIABLE_NAME, VARIABLE_NAME = EXPRESSION;
```

Shocks de variables exógenas III

- La correlación entre dos variables:

```
corr VARIABLE_NAME, VARIABLE_NAME = EXPRESSION;
```

Solución estocástica y simulación I

- El comando para especificar la solución estocástica es:

```
Command:stoch_simul(OPTIONS...) [VARIABLE_NAME...];
```

- El algoritmo de solución utiliza expansiones de Taylor alrededor del estado estacionario de hasta tercer orden.
- Cuando se definen variables en [VARIABLE_NAME], los resultados se muestran solo para las variables especificadas.
- Ejemplo: Simualción con aproximación de segundo orden (default).

```
stoch_simul;
```

Solución estocástica y simulación II

- Ejemplo: Simulación de primer orden con IRFs de 20 períodos para variables y y k:

```
stoch_simul(order=1,irf=20) y k;
```

Ejecutar Dynare

- Para simular el modelo, debemos ejecutar Dynare desde Matlab, ya sea directamente desde *command window* o desde un archivo .m (lo último es mejor)

```
MATLAB command: dynare FILENAME[.mod] [OPTIONS...]
```

- Para evitar que Dynare elimine elementos del workspace se utiliza la opción *noclearall*.

```
dynare FILENAME.mod noclearall
```

Resultados en Matlab Workspace

Almacenado en	Desccripción
oo_.mean	Promedio de variables endógenas. Si la opción *periods* no se especifica, muestra la media teórica.
oo_.var	Matriz de varianzas y covarianzas de las variables endógenas.
oo_.var_list	Listado de variables para las que se muestran los resultados
oo_.autocorr	Matriz de autocorrelación de variables endógenas. Si la opción *periods* no se especifica, muestra las autocorrelaciones teóricas para 5 rezagos.
oo_.irfs	IRFs de las varaibles endógenas a un shock exógeno.
oo_.dr	Información de las funciones de política de cada variable
oo_.dr.ghx	Matriz de coeficientes de las variables de estado endógenas en las funciones de política
oo_.dr.ghu	Matriz con los coeficientes relacionados a los shocks exógenos en las funciones de política
oo_.dr.order_var	Orden de las variables en las funciones de política.
oo_.dr.ys	Constante de las funciones de política (el estado estacioanrio).

Resultados en Matlab Workspace

Almacenado en	Desccripción
M_	Estructura del modelo
M_.endo_names	Orden de las variables endógenas en la declaración de variables.
M_.exo_names	Orden de las variables exógenas en la declaración de variables.
M_.endo_nbr	Número de variables endógenas
M_.exo_nbr	Número de variables exógenas
options_	Estructura con todas las opciones de Dynare

Agenda

RBC básico

- El modelo es:

$$\begin{aligned}
\max_{c_t, i_t, k_{t+1}} \ & \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\sigma}}{1-\sigma} \\
\text{s.t. } & c_t + i_t = y_t \\
& k_{t+1} = (1-\delta)k_t + i_t \\
& y_t = A_t k_t^{\alpha} \\
& \ln A_t = \rho \ln A_{t-1} + \sigma_\epsilon \epsilon \\
& \lim_{t \to \infty} \beta^t c_t^{-\sigma} k_{t+1} = 0 \\
& \epsilon \sim \mathcal{N}(0,1)
\end{aligned}$$

RBC básico CPOs

- Al resolver el modelo, este queda caracterizado por:

$$c_t^{-\sigma} = \beta \mathbb{E}_t c_{t+1}^{-\sigma} (\alpha A_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta) \quad \text{Ecuación de Euler}$$

$$k_{t+1} = (1-\delta) k_t + i_t \quad \text{Acumulación de capital}$$

$$y_t = A_t k_t^{\alpha} \quad \text{Función de producción}$$

$$y_t = c_t + i_t \quad \text{Clareo de mercado}$$

$$\ln A_t = \rho \ln A_{t-1} + \sigma_\epsilon \epsilon \quad \text{Productividad}$$

RBC básico: estado estacionario

- El estado estacionario está caracterizado por:

$$
\begin{aligned}
k_{ss} &= \left(\frac{1 - \beta(1-\delta)}{\beta \alpha A} \right)^{\frac{1}{\alpha - 1}} \\
y_{ss} &= A k_{ss}^{\alpha} \\
i_{ss} &= \delta k_{ss} \\
c_{ss} &= y_{ss} - c_{ss} \\
A &= 1
\end{aligned}
$$

- Parametrización: $\alpha = 0{,}36$, $\beta = 0{,}99$, $\sigma = 2$, $\delta = 0{,}025$, $\sigma_\epsilon = 0{,}01$, $\rho = 0{,}95$.

RBC básico: rbc.mod I

- Comenzamos definiendo las variables endógenas y exógenas:

```
// Preamble

// Variables endógenas
var c $c$ (long_name='consumption')
i $i$ (long_name='investment')
k $k$ (long_name='capital')
y $y$ (long_name='output')
a $a$ (long_name='AR(1) technology shock')
;

// List of predetermined variables
predetermined_variables k;

// Exogeneous variables
varexo e;
```

RBC básico: rbc.mod II

- Definición de parámetros

```
// Parameters name
parameters alph     $\alpha$    (long_name='capital share')
   betta    $\beta$     (long_name='discount factor')
   sigm     $\sigma$    (long_name='risk aversion')
   delta    $\delta$    (long_name='depreciation rate')
   sigm_e   $\sigma_e$  (long_name='standard deviation')
   rho      $\rho$      (long_name='autocorr. tech. shock')
;

// Parameters value
alph = 0.36;
betta = 0.99;
sigm = 2;
delta = 0.025;
sigm_e = 0.01;
rho = 0.95;
```

RBC básico: rbc.mod III

- Enrtegamos las ecuaciones del modelo y generamos archivo .tex del modelo

```
// Model block
model;
c^(-sigm)=betta*c(+1)^(-sigm)*(alph*exp(a(+1))*k(+1)^(alph-1
k(+1)=(1-delta)*k+i;
y=exp(a)*k^alph;
y=c+i;
a=rho*a(-1)+e;
end;                    // close the model block

// Modelo en latex
write_latex_original_model;
```

RBC básico: rbc.mod IV

- Estado estacionario con initval:

```
// Steady state
initval;
a=1;
k=(((1-betta*(1-delta))/alph*a)^(1/(alph-1)));
y=(exp(a)*k);
i=(delta*k);
c=(y-i);
end;                    // close steady state

// Checking Blanchard-Kahn rank condition
resid(1);
steady;
check;
```

RBC básico: rbc.mod V

- Definimos el bloque de shock y la simulación:

```
// Shocks
shocks;
var e=sigm_e^2;
end;

// Computation

stoch_simul(order=1,irf=40);
```

Agenda

Dynare en Matlab

- Podemos ejecutar Dynare desde la *command window* o desde un archivo.m.

```
1 close all;
2 clear all;
3 clc;
4
5 %% Se ejecuta modelo simple
6 dynare rbc noclearall
```

Output en Matlab I

- La opción resid nos permite ver los residuos de cada ecuación:

```
Residuals of the static equations:

Equation number 1 : -5.44e-06 : 1
Equation number 2 : 0 : k
Equation number 3 : 94.7742 : y
Equation number 4 : 0 : 4
Equation number 5 : 0.05 : a
```

Output en Matlab II

- También nos entrega el estado estacionario;

```
STEADY-STATE RESULTS:

c    2.75433
i    0.94973
k    37.9892
y    3.70406
a    0
```

Output en Matlab III

- La opción *check* brinda información sobre los valores propios del sistema:

```
EIGENVALUES:
Modulus                 Real          Imaginary

0.95                    0.95                  0
0.9765                  0.9765                0
1.034                   1.034                 0
1.25e+18                1.25e+18              0


There are 2 eigenvalue(s) larger than 1 in modulus
for 2 forward-looking variable(s)

The rank condition is verified.
```

Output en Matlab IV

- También observamos informción del modelo:

```
MODEL SUMMARY

Number of variables:          5
Number of stochastic shocks: 1
Number of state variables:    2
Number of jumpers:            2
Number of static variables:   2


MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS
Variables              e
e                 0.000100
```

Output en Matlab V

- Uno de los elementos más importantes del output son las funciones de política:

```
POLICY AND TRANSITION FUNCTIONS
____________     c          i          k          y          a
Constant     2.754328   0.949730  37.989218   3.704058          0
k(-1)        0.033561   0.001540   0.976540   0.035101          0
a(-1)        0.921470   2.597384   2.597384   3.518855   0.950000
e            0.969969   2.734089   2.734089   3.704058   1.000000
```

En este caso, la función de política del consumo es:

$$c_t = c_{ss} + a_{11}(k_t - k_{ss}) + b_{11}(a_{t-1} - a_{ss}) + c_{11}\epsilon$$
$$\iff c_t = 2{,}75 + 0{,}03(k_t - k_{ss}) + 0{,}92(a_{t-1} - a_{ss}) + 0{,}97\epsilon_t$$

Output en Matlab VI

- Los momentos teóricos de la variables:

```
THEORETICAL MOMENTS
VARIABLE          MEAN   STD. DEV.    VARIANCE
c                2.7543      0.0913      0.0083
i                0.9497      0.0894      0.0080
k               37.9892      2.0999      4.4094
y                3.7041      0.1706      0.0291
a                0.0000      0.0320      0.0010
```

Output en Matlab VII

- La matriz de correlaciones:

```
MATRIX OF CORRELATIONS
Variables          c        i        k        y        a
c             1.0000   0.7827   0.9680   0.9453   0.7635
i             0.7827   1.0000   0.6013   0.9429   0.9995
k             0.9680   0.6013   1.0000   0.8331   0.5769
y             0.9453   0.9429   0.8331   1.0000   0.9324
a             0.7635   0.9995   0.5769   0.9324   1.0000
```

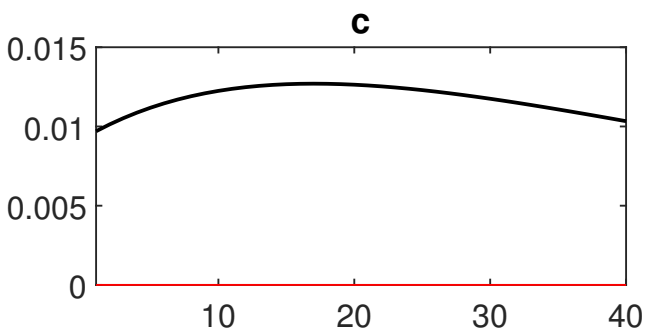
Output en Matlab VIII

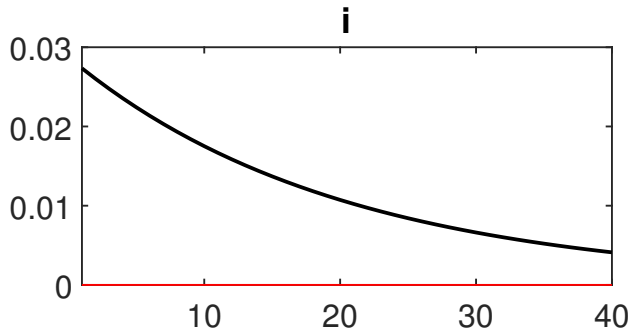
- La matriz de coeficientes de autocorrelacón (5 rezagos);

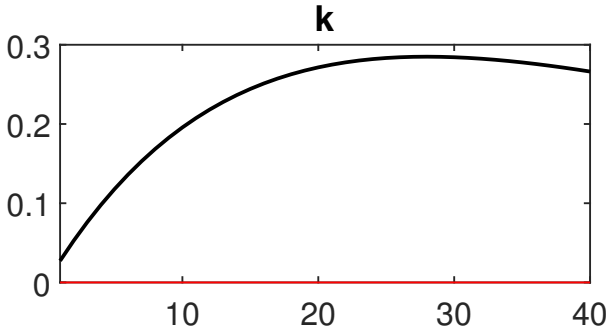
```
COEFFICIENTS OF AUTOCORRELATION
Order          1       2       3       4       5
c         0.9942  0.9877  0.9805  0.9726  0.9642
i         0.9521  0.9065  0.8631  0.8219  0.7826
k         0.9994  0.9977  0.9949  0.9911  0.9865
y         0.9761  0.9528  0.9301  0.9079  0.8863
a         0.9500  0.9025  0.8574  0.8145  0.7738
```

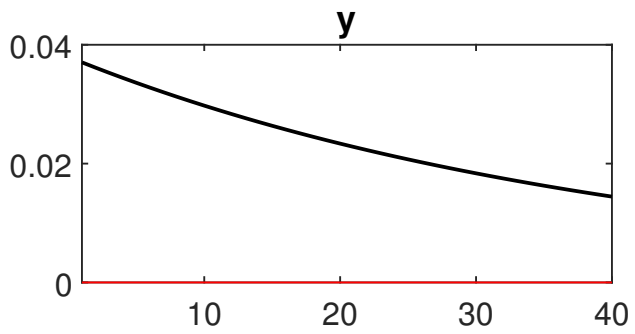
- IRFs del modelo:

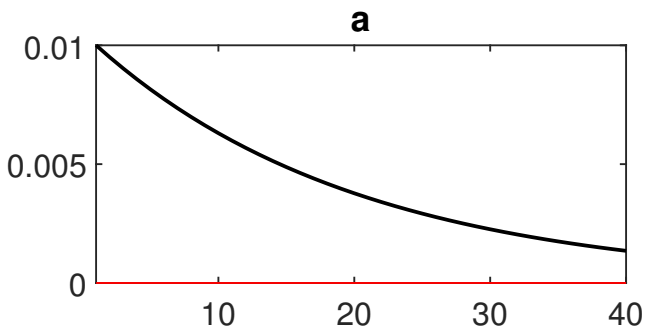
Output en Matlab IX











RBC log linealizado I

- El modelo anterior se resuelve como una aproximación de primer orden en niveles en torno al estado estacionario $(y_t - y_{ss})$.
- Otro forma de presentar los resultados es log linealizando el modelo. Esto nos permite expresar las variables como una desviación porcentual respecto al estado estacionario.
- Supongamos una aproximación de Taylor en torno a y_{ss} del logaritmo de y:

$$\ln y \approx \ln y_{ss} + \frac{1}{y_{ss}}(y - y_{ss})$$

lo que se puede reescribir como:

$$\ln y - \ln y_{ss} \approx \frac{y - y_{ss}}{y_{ss}}$$

RBC log linealizado II

- Si queremos analizar el modelo log-linealizado, tenemos dos opciones: log linealizar a mano (tedioso!!!) o utilizar alternativas de Dynare para conseguirlo.
- En Dynare, podemos implementar la log linealización de dos formas:
  - Reescribiendo en las ecuaciones del modelo las varibles y_t como $exp(y_t)$ (ya que $exp(y_t) = Y \iff \ln Y = y_t$). Dynare interpretará a la variable declarada y_t como el logaritmo de la variable Y.
  - Escribir el modelo de forma normal (en nivel) y utilizar la opción *loglinear* en el bloque *stoch_simul*. Requiere que todas las variables en estado estacionario sean estrictamente positivas.

RBC log linealizado III

- Ajustando las ecuaciones en el archivo .mod tenemos:

```
// Model block
model;
exp(c)^(-sigm)=betta*exp(c(+1))^(-sigm)*..
(alph*exp(a(+1))*exp(k)^(alph-1)+1-delta);
exp(k)=(1-delta)*exp(k(-1))+exp(i);
exp(y)=exp(a)*exp(k(-1))^alph;
exp(y)=exp(c)+exp(i);
a=rho*a(-1)+e;
end;                    // close the model block
```

- Como $A_{ss} = 0$, no podemos aplicar la opción de *stoch_simul*, salvo que ajustemos el proceso de la productividad, por ejemplo definiendo la productividad en nivel.

Resultados RBC log linealizado I

- Las funciones de política vienen dadas por:

```
POLICY AND TRANSITION FUNCTIONS
----------      c           i           k           y           a
Constant    1.013174   -0.051576   3.637302   1.309429           0
k(-1)       0.462887    0.061616   0.976540   0.360000           0
a(-1)       0.334554    2.734863   0.068372   0.950000    0.950000
e           0.352162    2.878803   0.071970   1.000000    1.000000
```

en este caso la función de política la expresamos como:

$$\ln c_t \approx \ln c_{ss} + a_{11}(\ln k_t - \ln k_{ss}) + b_{11}(a_t - a_{ss}) + c_{11}\epsilon_t$$

$$\iff \ln c_t \approx 0{,}84 + 0{,}44(\ln k_t - \ln k_{ss}) + 0{,}35(a_t - a_{ss}) + 0{,}37\epsilon_t$$

Relación entre modelo en nivel y log linealización I

- De lo anterior tenemos que:

$$\ln c_t = \ln c_{ss} + a_{11}(\ln k_t - \ln k_{ss}) + b_{11}(a_t - a_{ss}) + c_{11}\epsilon_t$$

Reorganizando los términos:

$$\ln c_t - \ln c_{ss} = +a_{11}(\ln k_t - \ln k_{ss}) + b_{11}(a_t - a_{ss}) + c_{11}\epsilon_t$$

Utilizando el resultado de la aproximación de $\ln c_t$ en torno al estado estacionario:

$$\frac{c_t - c_{ss}}{c_{ss}} \approx a_{11}\left(\frac{k_t - k_{ss}}{k_{ss}}\right) + b_{11}(a_t - a_{ss}) + c_{11}\epsilon_t$$
$$\iff c_t = c_{ss} + a_{11}\frac{c_{ss}}{k_{ss}}(k_t - k_{ss}) + b_{11}c_{ss}(a_t - a_{ss}) + c_{11}c_{ss}\epsilon_t$$

Relación entre modelo en nivel y log linealización II

- Reemplazando los valores obtenemos:

$$c_t = 2{,}75 + 0{,}03(k_t - k_{ss}) + 0{,}92(a_t - a_{ss}) + 0{,}97\epsilon_t$$

- Antes habíamos obtenido:

$$c_t = 2{,}75 + 0{,}03(k_t - k_{ss}) + 0{,}92(a_{t-1} - a_{ss}) + 0{,}97\epsilon_t$$

- Comparemos las IRFs en Matlab.

Iteración de parámetros

- Una situación común es analizar la sensibilidad de los resultados obtenidos ante variaciones de algunos parámetros.
- Para ello tenemos dos opciones:
  - Recalcular el modelo cada vez que modificamos un parámetro.
  - Solo ajustar aspectos necesarios del modelo (funciones de política, estado estacionario, etc).

Iteración de parámetros: recalculando el modelo I

- En el entorno Matlab definimos:

```
% Se itera sobre parametro de depreciacion
deltas=[0 0.025 0.5 1];

for i=1:length(deltas)
% Se define el parametro
delta=deltas(i);
% Se guarda para entregarlo al archivo .mod

save delta ;
dynare rbc_param noclearall;
end
```

Iteración de parámetros: recalculando el modelo II

- En el archivo .mod correspondiente:

```
// Parameters value
load delta;
set_param_value('delta', delta);
alph=0.36;
betta=0.99;
// delta=0.025;
sigm=2;
sigm_e=0.01;
rho=0.95;
```

Iteración de parámetros: recalculando estructuras I

- En el entorno Matlab definimos:

```
% Se itera sobre parametro de depreciacion
deltas=[0 0.025 0.5 1];

first_time = 1;
for i=1:length(deltas)
  if first_time
    set_param_value('delta',deltas(i));
    dynare rbc_param2 noclearall;
    first_time = 0;
  else
        set_param_value('delta',deltas(i));
        [info, oo_] = stoch_simul(M_, options_
           , oo_, var_list_);
  end
end
```

Iteración de parámetros: recalculando estructuras II

- En el archivo .mod correspondiente:

```
// Parameters value
set_param_value('delta', delta);
alph=0.36;
betta=0.99;
// delta=0.025;
sigm=2;
sigm_e=0.01;
rho=0.95;
```

Iteración de parámetros: recalculando estructuras III

- Comparemos los resultados de ambas formas....