Appendix: Matlab codes

This appendix presents the Matlab and Dynare codes that were used to solve and simulate the model.
The first Matlab program defines variables and solves the steady state of assets:

```
%%%%%%%%%%%%%% Steady state solver %%%%%%%%%%%%%
  gamma  = 2;          % risk aversion
  delta  = 0.025;      % depreciation
  betta  = 0.98;       % time preference
  alfa   = 0.36;       % share of capital on output
  miz    = 1;          % steady state productivity
  roz    = 0.75;       % adjustment productivity
  sigmaz = 0.013;      % volatility productivity
  phi    = 0.05;       % barrier parameter
  rok    = 0.7;        % initial parameter for law of motion
  mie    = 1;          % steady state employment
  roe    = 0.7;        % adjustment employment
  sigmae1 = 0.05;      % volatility employment capitalists
  sigmae2 = 0.1;       % volatility employment workers
  roez   = 0.3;        % cyclicality of employment workers
  lambda = 0.8;        % share of capitalists
  tau    = 0;          % tax rate


pi = (1-alfa)*lambda^alfa + (1-tau)*alfa*lambda^(alfa-1);
syms a
solve(2*phi*((a^alfa)*pi - delta*a)^gamma +
+ (a^3)*(betta - betta*delta -1) +
+ betta*alfa*(1-tau)*(lambda^(alfa-1))*a^(alfa+2) == 0, a)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Choose the stable steady state (real, positive) and plug it into all subsequent programs where "ass" is required (assets steady state). The following program is the mother Matlab program that runs Dynare program in a loop (note1: run only the mother program and not the Dynare programs; note2: when in the following programs a line of code spans several lines of text, in Matlab, compress it to a single line):

```
%%%%%%%%% Mother program %%%%%%%%%%%%%
```

```
ass = 31.7838916986589918973;     % st.st. assets
lambda = 0.8;                     % share of capitalists
kss = lambda*ass;                 % st.st capital
rok = 0.7;                        % guess value for law of motion
vZetaOld = ones(1,3);             % var. that stores old coefs.
vZetaOld(1) = (1-rok)*kss - rok;  %  - of aggregate law of motion
vZetaOld(2) = rok;
vZetaOld(3) = rok;
vZetaNew = ones(1,3);             % the same for new coefficients
convergence = ones(3,2,100);      % var. for convergence process
dLambda = 0.1;                    % speed of convergence
vTheta = ones(1,5);               % variable for coefs. in -
pZeta0 = vZetaOld(1);             % - individual law of motion
pZeta1 = vZetaOld(2);
pZeta2 = vZetaOld(3);
save InitParams.mat pZeta0 pZeta1 pZeta2; % saves coefs for Dynare

% The following solves the model iteratively using old coeffs of
% aggregate law  of motion and computes and stores the new ones:
for i = 1:100
dynare diplomkaNEWTAX1.mod noclearall;        % runs model (dynare)

% computes new coefs of aggregate law of motion out of
% individual coefs.
vZetaNew(1) = lambda*(vTheta(1) + vTheta(3));
vZetaNew(2) = vTheta(2) + lambda*vTheta(5);
vZetaNew(3) = lambda*vTheta(4);

% now store both new and old coefs of aggregate law of motion
convergence(1,1,i) = vZetaOld(1);
convergence(2,1,i) = vZetaOld(2);
convergence(3,1,i) = vZetaOld(3);
convergence(1,2,i) = vZetaNew(1);
convergence(2,2,i) = vZetaNew(2);
convergence(3,2,i) = vZetaNew(3);

% The new coefs become old coefs for the next iteration
vZetaOld = dLambda * vZetaNew + (1-dLambda) * vZetaOld;
pZeta0 = vZetaOld(1);
pZeta1 = vZetaOld(2);
pZeta2 = vZetaOld(3);
```

```
delete InitParams.mat;     % delete the old stored values
save InitParams.mat pZeta0 pZeta1 pZeta2; % store new ones
end  % end of the convergence loop

% And finally run the model with converged coefficients
dynare diplomkaNEWTAX2.mod noclearall;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The following Dynare code solves the model for the use of iterations - it has a compressed form and does not include all variables (the ones that are not necessary and are only a product of identities (i.e. investment, output...). The name of the file is "diplomkaNEWTAX1.mod":

```
// Model solver (compressed) -  diplomkaNEWTAX1.mod //
var cc cw a r w z k ec ew l;

varexo epsz epse1 epse2;

parameters gamma delta betta alfa miz roz sigmaz phi
cssc cssw ass wss rss kss rok mie roe roez sigmae1
sigmae2 pZeta0 pZeta1 pZeta2 lambda tau;

  gamma  = 2;           // risk aversion
  delta  = 0.025;       // depreciation
  betta  = 0.98;        // time preference
  alfa   = 0.36;        // share of capital on output
  miz    = 1;           // steady state productivity
  roz    = 0.75;        // adjustment productivity
  sigmaz = 0.013;       // volatility productivity
  phi    = 0.05;        // barrier parameter
  rok    = 0.7;         // initial parameter for law of motion
  mie    = 1;           // steady state employment
  roe    = 0.7;         // adjustment employment
  sigmae1 = 0.05;       // volatility employment capitalists
  sigmae2 = 0.1;        // volatility employment workers
  roez   = 0.3;         // cyclicality of employment workers
  lambda = 0.8;         // share of capitalists
  tau    = 0;           // tax rate

  ass =   31.783891698658;        // steady state assets
  kss = lambda*ass;               // steady state capital
  rss   = alfa*(kss)^(alfa-1);    // steady state interest rate
```

```
  wss   = (1-alfa)*(kss)^(alfa);  // steady state wage
  cssc  = (ass^alfa)*(1-tau*alfa) - delta*ass;
  // steady state consumption capitalists
  cssw  = wss +tau*rss*kss/(1-lambda);
  // steady state consumption workers

// load coefs for aggregate law of motion
load InitParams;
set_param_value('pZeta0',pZeta0);
set_param_value('pZeta1',pZeta1);
set_param_value('pZeta2',pZeta2);

model;
//(1) euler equation capitalists
cc^(-gamma) = phi*2/(a^3) +
+ betta*(cc(+1))^(-gamma)*(1 + (1-tau)*r(+1) - delta);

//(2) interest rate
r = alfa*z*((k(-1))^(alfa-1))*l^(1-alfa);

// (3) wage
w = (1-alfa)*z*((k(-1))^(alfa))*l^(-alfa);

// (4)budget constraint capitalists
cc + a  = (1-tau)*r*a(-1) + w*ec + (1-delta)*a(-1);

// (5)budget constraint workers
cw  =  w*ew + tau*r*k(-1)/(1-lambda);

// (6) aggregate shock
z = (1-roz)*miz + roz*z(-1) + epsz;

// (7) law of motion for capital
k = pZeta0 + pZeta1*k(-1) + pZeta2*z;

// (8) law of motion for labor
l = mie + ((1-lambda)*roez/(1-roe))*(z - 1);

// (9) idiosyncratic shock capitalists
ec = (1-roe)*mie + roe*ec(-1) + epse1;
```

```
// (10) idiosyncratic shock workers
ew = (1-roe)*mie + roe*ew(-1) + roez*(z-1) + epse2;

end;


initval;
  cc  = cssc; // consumption capitalists
  cw  = cssw; // consumption workers
  a   = ass;  // assets
  r   = rss;  // interest rate
  w   = wss;  // wage
  z   = 1;    // productivity
  k   = kss;  // capital
  ec  = 1;    // employment capitalists
  ew  = 1;    // employment workers
  l   = 1;    // total labor
end;

steady;

check;

shocks;
var epsz = sigmaz^2;
var epse1 = sigmae1^2;
var epse2 = sigmae2^2;
end;

stoch_simul(order=1,nocorr,noprint,nomoments,IRF=0);

// Now read the coefficients of individual law of motion
mPolicy = [oo_.dr.ys'; oo_.dr.ghx'; oo_.dr.ghu'];
mPolA = mPolicy(:,4);
mPolA(1) = mPolicy(1,3);

// Rearrange parameters
dTheta0 = mPolA(1)-mPolA(2)*mPolA(1)-mPolA(5)-mPolA(3)-
- mPolA(4)*mPolicy(1,7);
dTheta1 = mPolA(2);
dTheta2 = mPolA(5);
```

```
dTheta3 = mPolA(3);
dTheta4 = mPolA(4);

// Save parameters
vTheta = [dTheta0 dTheta1 dTheta2 dTheta3 dTheta4];
////////////////////////////////////////////////
```

And finally the following program is for the model with all variables and retrieves the IRFs:

```
// Model solver full -  diplomkaNEWTAX2.mod //
var cc cw a r w z k ec ew l yc yw i;

varexo epsz epse1 epse2;

parameters gamma delta betta alfa miz roz sigmaz phi
cssc cssw ass wss rss kss rok mie roe roez sigmae1
sigmae2 pZeta0 pZeta1 pZeta2 lambda tau;

  gamma  = 2;           // risk aversion
  delta  = 0.025;       // depreciation
  betta  = 0.98;        // time preference
  alfa   = 0.36;        // share of capital on output
  miz    = 1;           // steady state productivity
  roz    = 0.75;        // adjustment productivity
  sigmaz = 0.013;       // volatility productivity
  phi    = 0.05;        // barrier parameter
  rok    = 0.7;         // initial parameter for law of motion
  mie    = 1;           // steady state employment
  roe    = 0.7;         // adjustment employment
  sigmae1 = 0.05;       // volatility employment capitalists
  sigmae2 = 0.1;        // volatility employment workers
  roez   = 0.3;         // cyclicality of employment workers
  lambda = 0.8;         // share of capitalists
  tau    = 0;           // tax rate

  ass =   31.783891698658;        // steady state assets
  kss = lambda*ass;               // steady state capital
  rss   = alfa*(kss)^(alfa-1);    // steady state interest rate
  wss   = (1-alfa)*(kss)^(alfa);  // steady state wage
  cssc  = (ass^alfa)*(1-tau*alfa) - delta*ass;
  // steady state consumption capitalists
```

```
  cssw  = wss +tau*rss*kss/(1-lambda);
  // steady state consumption workers
  ycss  = wss + (1-tau)*rss*ass;
  // steady state income capitalists
  ywss  = wss + tau*rss*kss/(1-lambda);
  // steady state income workers
  iss   = delta*ass; // investment

// load coefs for aggregate law of motion
load InitParams;
set_param_value('pZeta0',pZeta0);
set_param_value('pZeta1',pZeta1);
set_param_value('pZeta2',pZeta2);

model;
//(1) euler equation capitalists
cc^(-gamma) = phi*2/(a^3) +
+ betta*(cc(+1))^(-gamma)*(1 + (1-tau)*r(+1) - delta);

//(2) interest rate
r = alfa*z*((k(-1))^(alfa-1))*l^(1-alfa);

//(3) wage
w = (1-alfa)*z*((k(-1))^(alfa))*l^(-alfa);

// (4) budget constraint capitalists
cc + a  = (1-tau)*r*a(-1) + w*ec + (1-delta)*a(-1);

// (5) budget constraint workers
cw  =  w*ew + tau*r*k(-1)/(1-lambda);

// (6) aggregate shock
z = (1-roz)*miz + roz*z(-1) + epsz;

// (7) law of motion for capital
k = pZeta0 + pZeta1*k(-1) + pZeta2*z;

// (8) law of motion for labor
l = mie + ((1-lambda)*roez/(1-roe))*(z - 1);

// (9) idiosyncratic shock capitalists
```

```
ec = (1-roe)*mie + roe*ec(-1) + epse1;

// (10) idiosyncratic shock workers
ew = (1-roe)*mie + roe*ew(-1) + roez*(z-1) + epse2;

// (11) income capitalists
yc = (1-tau)*r*a(-1) + w*ec;

// (12) investment
i = yc - cc;

// (13) income workers
yw  =  w*ew + tau*r*k(-1)/(1-lambda);
end;


initval;
  cc  = cssc; // consumption capitalists
  cw  = cssw; // consumption workers
  a   = ass;  // assets
  r   = rss;  // interest rate
  w   = wss;  // wage
  z   = 1;    // productivity
  k   = kss;  // capital
  ec  = 1;    // employment capitalists
  ew  = 1;    // employment workers
  l   = 1;    // total labor
  yc  = ycss; // income capitalists
  yw  = ywss; // income workers
  i   = iss;  // investment

end;

steady;

check;

shocks;
var epsz = sigmaz^2;
var epse1 = sigmae1^2;
var epse2 = sigmae2^2;
```

```
end;


stoch_simul(periods=2100, order = 1,irf =100);
/////////////////////////////////////////////////
```

After running the mother program and getting the converged laws of motion, it is possible to simulate the economy na compute various inequality measures using the following program:

```
%% Economy simulator %%
  gamma  = 2;           % risk aversion
  delta  = 0.025;       % depreciation
  betta  = 0.98;        % time preference
  alfa   = 0.36;        % share of capital on output
  miz    = 1;           % steady state productivity
  roz    = 0.75;        % adjustment productivity
  sigmaz = 0.013;       % volatility productivity
  phi    = 0.05;        % barrier parameter
  rok    = 0.7;         % initial parameter for law of motion
  mie    = 1;           % steady state employment
  roe    = 0.7;         % adjustment employment
  sigmae1 = 0.05;       % volatility employment capitalists
  sigmae2 = 0.1;        % volatility employment workers
  roez   = 0.3;         % cyclicality of employment workers
  lambda = 0.8;         % share of capitalists
  tau    = 0;           % tax rate
  ass =   31.783891698658;        % steady state assets
  kss = lambda*ass;               % steady state capital

t = 15000;  %% number of periods
n = 5000;   %% number of households
eshocks1= normrnd(0,sigmae1,[lambda*n t]);  % shocks c.
eshocks2= normrnd(0,sigmae2,[(1-lambda)*n t]); % shocks w.
zshocks = normrnd(0,sigmaz,[1 t]); % productivity shocks

% Variables:
a = ones(t,n);
c = ones(t,n);
y = ones(t,n);
k = ones(1,t);
w = ones(1,t);
```

```
r = ones(1,t);
z = ones(1,t);
l = ones(1,t);
e = ones(t,n);

% Starting values of variables:
a(1:2,1:lambda*n) = ass;
a(1:t,(lambda*n+1):n)=0;
k(1:2) = lambda*ass;
c(1:lambda*n,1) = (ass^alfa)*(1-tau*alfa) - delta*ass;

% Simulation loop
for i = 2:(t-1)
    k(i) = lambda*mean(a(i,1:lambda*n));
    z(i) = (1-roz) + roz*z(i-1)+zshocks(i);
    l(i) = 1 + (1-lambda)*roez*(z(i)-1)/(1-roe);
    w(i) = (1-alfa)*z(i)*((k(i))^(alfa))*l(i)^(-alfa);
    r(i) = alfa*z(i)*((k(i))^(alfa-1))*l(i)^(1-alfa);
    for j = 1:lambda*n
        e(i,j) = (1-roe) + roe*e(i-1,j)+eshocks1(j,i);
        a(i+1,j) = vTheta(1) + vTheta(2)*a(i,j) +
        + vTheta(3)*e(i,j) + vTheta(4)*z(i) + vTheta(5)*k(i);
        c(i,j)   = (1-tau)*r(i)*a(i,j) + w(i)*e(i,j) +
        + (1-delta)*a(i,j) - a(i+1,j);
        y(i,j)   = (1-tau)*r(i)*a(i,j) + w(i)*e(i,j);
    end
    for m = 1:(1-lambda)*n
        e(i,m+lambda*n) = (1-roe) + roe*e(i-1,m+lambda*n) +
        + roez*(z(i)-1) + eshocks2(m,i);
        c(i,m+lambda*n) = w(i)*e(i,m+lambda*n) +
        + tau*r(i)*k(i)/(1-lambda);
        y(i,m+lambda*n) = c(i,m+lambda*n);
    end
end

% Sort variables for the Gini
asort = ones(t,n);
csort = ones(t,n);
ysort = ones(t,n);
for i = 1:(t-1)
    asort(i,:)=sort(a(i,:));
```

```
    csort(i,:)=sort(c(i,:));
    ysort(i,:)=sort(y(i,:));
end

% Compute wealth gini
giniA = ones(1,t);
for i = 1:(t-1)
    sum1 = 0;
    sum2 = 0;
    for j = 1:n
        sum1 = (n+1-j)*asort(i,j) + sum1;
        sum2 = asort(i,j) + sum2;
    end
    giniA(i) = (n+1-2*(sum1/sum2))/n;
end

% Compute consumption gini
giniC = ones(1,t);
for i = 1:(t-1)
    sum1 = 0;
    sum2 = 0;
    for j = 1:n
        sum1 = (n+1-j)*csort(i,j) + sum1;
        sum2 = csort(i,j) + sum2;
    end
    giniC(i) = (n+1-2*(sum1/sum2))/n;
end

% Compute income gini
giniY = ones(1,t);
for i = 1:(t-1)
    sum1 = 0;
    sum2 = 0;
    for j = 1:n
        sum1 = (n+1-j)*ysort(i,j) + sum1;
        sum2 = ysort(i,j) + sum2;
    end
    giniY(i) = (n+1-2*(sum1/sum2))/n;
end

% Compute Lorenz curve (lor)
```

```
sum = 0;
lor = ones(n);
atrim = asort(t-1,1:n);
atrim(1:350)=0;
for i = 1:n
    sum = atrim(i) + sum;
end
cum = 0;
ind = 1:n;
for i = 1:n
    cum = atrim(i)+cum;
    lor(i) = cum/sum;
end

% Compute theil coefficient
theilY = ones(1,t);
for i = 1:(t-1)
    ybar = mean(y(i,:));
    sum1 = 0;
    for j = 1:n
        sum1 = (y(i,j)/ybar)*log(y(i,j)/ybar)+sum1;
    end
    theilY(i) = sum1/n;
end

% Compute percentile income ratios
ratio1 = ones(1,t);
for i = 1:(t-1)
    ratio1(i) = ysort(i,0.9*n)/ysort(i,0.1*n);
end

ratio2 = ones(1,t);
for i = 1:(t-1)
    ratio2(i) = ysort(i,0.9*n)/ysort(i,0.5*n);
end

ratio3 = ones(1,t);
for i = 1:(t-1)
    ratio3(i) = ysort(i,0.5*n)/ysort(i,0.1*n);
end
```

```
% Compute theil coefficient decomposition
theilC = ones(1,t);
ytotalC = ones(1,t);
for i = 1:(t-1)
    ybarC = mean(y(i,1:lambda*n));
    sum1 = 0;
    sum2 = 0;
    for j = 1:lambda*n
        sum1 = (y(i,j)/ybarC)*log(y(i,j)/ybarC)+sum1;
        sum2 = y(i,j) + sum2;
    end
    ytotalC(i) = sum2;
    theilC(i) = sum1/(lambda*n);
end

theilW = ones(1,t);
ytotalW = ones(1,t);
for i = 1:(t-1)
    ybarW = mean(y(i,(1+lambda*n):n));
    sum1 = 0;
    sum2 = 0;
    for j = 1:(1-lambda)*n
        sum1 = (y(i,j+lambda*n)/ybarW)*log(y(i,j+
        +lambda*n)/ybarW)+sum1;
        sum2 = y(i,j+lambda*n) + sum2;
    end
    ytotalW(i) = sum2;
    theilW(i) = sum1/((1-lambda)*n);
end

ytotal = ones(1,t);
for i = 1:(t-1)
    sum1 = 0;
    for j = 1:n
        sum1 = y(i,j) + sum1;
    end
    ytotal(i)=sum1;
end

shareC = ones(1,t);
shareW = ones(1,t);
```

```
meanC = ones(1,t);
meanW = ones(1,t);
meantotal = ones(1,t);
theilCW = ones(1,t);
for i = 1:(t-1)
shareC(i) = ytotalC(i)/ytotal(i);
shareW(i) = ytotalW(i)/ytotal(i);
meanC(i) = mean(y(i,1:lambda*n));
meanW(i) = mean(y(i,(1+lambda*n):n));
meantotal(i) = mean(y(i,:));
theilCW(i) = shareC(i)*theilC(i) + shareW(i)*theilW(i) +
+ shareC(i)*log(meanC(i)/meantotal(i)) +
+ shareW(i)*log(meanW(i)/meantotal(i));
end
```