Online Appendix

A The Design of the GDSGE Toolbox

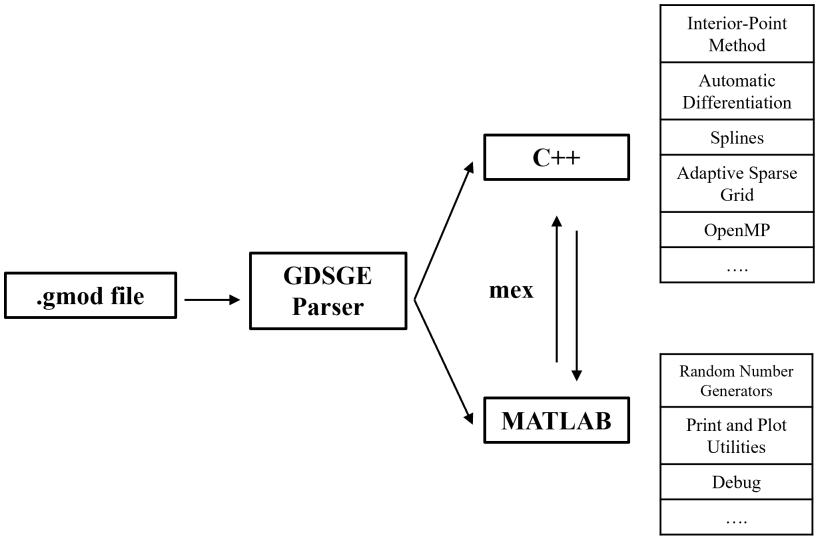

Figure 11: Toolbox Design and Implementations

In this section, we described in detail how the toolbox is designed and implemented. The design of the toolbox is depicted in Figure **??**. Users create and edit their own gmod file that describes the dynamic equilibrium of their model in the general form (1) of the general framework. Gmod stands for global model. The structure of the gmod file is given in Subsection **??**. The gmod files can be uploaded to the toolbox's website, and the toolbox compiles the files into MATLAB script files and C++ dynamic libraries that solve for recursive equilibria using policy function iterations and simulate the equilibrium dynamics. The functions of the complied files, which consist of solving system of equations, discretizing, and approximating policy functions, are described in Subsection **??**

The MATLAB script files and C++ dynamic libraries should run locally on users' computers. After they finish running, they return the policy and state transition functions from converged time iterations and the Monte Carlo simulation samples.

A.1 User Inputs: the gmod Files

The toolbox asks users to provide gmod files that contain the equilibrium system (1) of their models. In this subsection, we provide the description for a minimal gmod

file such as that for the leading example in Section 2, and refer readers to the toolbox's website for a detailed user manual. A minimal gmod file should contain the following components:

parameters. Exogenous parameters that do not vary across states or over time.

var_shock. Exogenous state variables z in system (1). These states need to be specified as discretized points.[30]

shock_num. The number of discretized points for *var_shock*. For multidimensional *var_shock*, this should be the size of the Cartesian products of the discretized sets for all dimensions.

shock_trans. The Markov transition matrix for exogenous state variables.

var_state. Endogenous state variables s in system (1). The toolbox requires users to specify the grid for each of these variables.[31]

var_policy. Policy variables x in system (1). For state variables with implicit laws of motion, we include vectors of these variables in future states among the policy variables.

var_aux. Some policy variables can be directly computed as relatively simple, explicit functions of other variables in x, s, x', s'. We use the keyword *var_aux* for these variables. We exclude them from the var_policy in order to reduce the number of equations and unknowns to be solved in each policy function iteration.

var_interp. These are policy variables x that appear in equilibrium system (1) as future states $x'(z')$. In the policy iteration at step n, $x'(z')$ can be evaluated by interpolating the policy function solved in the last iteration, i.e., $\mathcal{P}^{(n-1)}(z', s')$.[32] Even though the general

[30] To accommodate exogenous continuous shocks such as AR(1) processes, treat continuous shocks as endogenous state variables and approximate the shock processes with discretized innovations as exogenous states.

[31] For function approximations using fixed-grid discretization such as splines, the grids will be used by default; for the adaptive sparse grid method, the two end points of the grids will be used as the range of the state variables.

[32] As emphasized in Section 3.2, our framework allows for state variables with implicit state transitions, denoted by $\bar{s}$ following the notation in the section. This is made possible by including the future values of these state variables, $\bar{\bar{s}}'$, as unknowns and supplying additional *consistency equations* that $\bar{\bar{s}}'$ need to satisfy in equilibrium. Therefore, it only requires a single-step interpolation to evaluate $\mathcal{P}^{(n-1)}(z', (\bar{s}', \bar{\bar{s}}'))$ and obtain future policy variables x' that are necessary in evaluating the current equation system. An alternative approach to deal with implicit state transitions, adopted by Elenev et al. (2021) for example, is to first use the transition function $\mathcal{T}^{(n-1)}(z, z', s)$ solved in the last iteration to forecast $\bar{\bar{s}}' = \mathcal{T}^{(n-1)}_{\bar{s}'}(z, z', s)$, and then evaluate $\mathcal{P}^{(n-1)}(z', (\bar{s}', \bar{\bar{s}}'))$ to get future policy variables x'. This alternative approach can be implemented in our toolbox by declaring future values of these state variables as *var_interp*, and interpolate $\mathcal{T}^{(n-1)}$ and $\mathcal{P}^{(n-1)}$ sequentially as described above. This alternative approach, however, does not solve some of our examples. The reason is outlined in the introduction and should be clearer now—this alternative approach directly uses the last-period transition function $\mathcal{T}^{(n-1)}$ to evaluate $\bar{\bar{s}}'$ as a function of current state variables only, and does not allow the current policy variables to affect $\bar{\bar{s}}'$. In the context of the Heaton and Lucas (1996) example, this approach basically does not allow the current choice of bonds and shares to affect the future wealth share, which is slightly at odds with equilibrium properties. However, using dampening,

formulation allows any policy variable in x to appear as a future state, in practice not all of them are necessary. Here, we only include those variables that need to be interpolated in the policy function iteration steps. When the time iteration converges, *var_interp* also delivers the state transition functions.

The updates of each *var_interp* after each time iteration should be specified after declaring the *var_interp*. The updates can use functions of solutions of policy variables in *var_policy* or *var_aux*, combining any parameters or exogenous states.

The model block. The model definition is enclosed in a block starting with *model;* and ending with *end;*. The model block should include an *equations* block, in which each line represents one equation of the equilibrium system (1) to be solved. Other variables required to be evaluated in these equations should be put into the model block preceding the *equations* block. A variable followed by a prime (') indicates that the variable is a vector of length *shock_num*, and it is usually used to represent future states, z' or s', or future policies, x', in the notation for the general framework. The model block can use the following built-in functions.

GDSGE_EXPECT. Calculate the conditional expectation of the model objects using the default transition matrix specified in *shock_trans*. This function can also accommodate a different transition matrix than *shock_trans* so that the toolbox can be used to solve models with heterogeneous beliefs (see Cao (2018) and the associated gmod file in the toolbox's website for an example).

GDSGE_INTERP_VEC. Evaluate function approximations specified in *var_interp*. This function, when followed by a prime ('), indicates that the approximation is evaluated for a vector of arguments of length *shock_num*; accordingly, the input and output variables in this case should also be followed by a prime. The output is thus a vector corresponding to $s'(z')$ or $x'(z')$ in system (1) for all possible realizations of exogenous states z'.

The simulate block. This optional block specifies the Monte Carlo simulations after the convergence of time iterations. It should specify *num_samples* for the number of sample paths, *num_periods* for the number of simulation periods of each path, *initial* for initial values of endogenous and exogenous states, *var_simu* for the variables to be recorded in the simulation, and the transitions for each endogenous state (the transition for exogenous states are handled automatically by the toolbox).

By default, the simulation resolves the system of equations (with $s'(z')$ and $x'(z')$ given by the converged policy and state transition functions) at each time step. This minimizes numerical errors within a time step. We also implement the conventional and fast,

the method can still solve Heaton and Lucas's model with reasonable accuracy. We provide its GDSGE implementation on the toolbox's website.

albeit less accurate, simulation method based on interpolating the policy and state transition functions directly. To use this method, the users should specify *SIMU_INTERP*=1 and declare interpolated variables in *var_output*. See the user manual on the toolbox's website for details.

These simulations are important to compute stationary recursive equilibria, i.e., recursive equilibria with an ergodic distribution over the state variables, from which the model moments are calculated (the rigorous definition is provided in Duffie et al. (1994) and Cao (2020)). They can also be used to calculate nonlinear impulse response functions (see Cao and Nie (2017) for example) to understand the transmission mechanisms, or to estimate the models.

A.2 Implementations

Once a gmod file is processed by the toolbox, it returns MATLAB files that can be run locally in the user' computer to solve and simulate their model.

General Implementations The gmod file is first parsed into an internal model structure, based on which the toolbox generates the C++ and MATLAB source codes. The toolbox then compiles the C++ source code to a dynamic library that MATLAB can call. All the actual computations are implemented in the native C++ code to achieve maximum performance and are contained in the dynamic library, while the MATLAB script file provides a convenient interface to print, debug, and specify options. To reach maximum computation efficiency, the actual calculations including equation solver, interpolation, automatic differentiation, and parallel computation, are implemented in C++. Below we discuss each of them in detail.[33]

Equation Solver The time iteration step requires solving the systems of equations (2) at collocation points in the state space. Since evaluating the function to be solved is rather costly, it is crucial that we design an efficient equation solver. We implement the Powell's dogleg algorithm augmented with an projection-based interior-point method to respect the box constraints (Powell, 1970; Coleman and Li, 1996; Bellavia et al, 2012).[34]

We also provide interfaces to commercial optimization software SNOPT and Knitro for users with licenses.[35] The equation solvers are supplemented with randomization

[33]For each of the implementation details, we also provide a separate library when possible so that they can be used independently of the toolbox.

[34]It is important that the equation solver can respect box constraints for unknowns, as the toolbox needs to deal with inequality conditions such as complementary slackness conditions where the unknowns are imposed to have a sign restriction.

[35]Our own implementation of the algorithm turns out to be more efficient both in terms of number of function calls and overhead, for a large class of test problems. This is partly because the algorithm

over initial guesses at each collocation point.

Automatic Differentiation Since we use a gradient-based equation solver and the function evaluation is expensive, it is crucial to calculate the gradients efficiently. We use the automatic differentiation method implemented by Adept (Hogan, 2014) (with some modification to facilitate parallelization). This library utilizes the expression template feature of C++ and brings the computational costs on par with evaluating analytical gradients.

Interpolation The time iteration step (2) involves function approximations because $(z', s'(z'))$ might fall outside the set of collocation points, $\mathcal{C}^{(n)}$. The default option is multidimensional linear interpolation or splines. We also implement a multidimensional adaptive sparse grid method with hierarchical hat basis functions developed in Ma and Zabaras (2009) and recently applied in economic applications by Brumm and Scheidegger (2017). We provide analytical gradients to these approximation procedures, which complement the automatic differentiation method to achieve maximum performance.

Parallel Computation Within a policy function iteration, the systems of equations (2) are independent of each other, while they share a large chunk of data for function approximations. To utilize this structure, we use multi-threaded parallel computation, thus all problems share a same block of memory for function approximation parameters, minimizing the overhead for data communications. When we evaluate the interpolations with splines or the adaptive sparse grid method, we design the data structure such that it can exploit the single-instruction-multiple-data (SIMD) CPU instructions. This design of parallelism turns out to be efficient—the program executes fast on a single processor and scales well with the number of CPU cores.

B Example Toolbox Codes

In this appendix, we provide the gmod files for the models discussed in Section 4. These codes can also be downloaded from the toolbox's website, together with the gmod codes for many other models.

we implement is designed for solving equations, while the commercial software target a more general class of optimization problems. Besides, the equation solver we implement targets small to medium scale problems (less than 1000 unknowns), which are adequate for most applications in economics while the commercial software accommodates much larger problems and thus incurs more overhead.

B.1 Mendoza (2010)

```
% Parameters
parameters gamma sigma omega beta alpha eta delta a phi kappa tau;
parameters flow_scale;

beta    = 0.92;     % Discount factor
sigma   = 2;        % CRRA
omega   = 1.846;    % Labor elasticity
gamma   = 0.306;    % Capital share
alpha   = 0.592;    % Labor share
eta     = 0.102;    % Intermediate share
delta   = 0.088;    % Depreciation rate
a       = 2.75;     % Capital adjustment cost
phi     = 0.26;     % Working capital exernal finance share
kappa   = 0.2;      % Tightness of collateral constraint
tau     = 0.17;     % Consumption tax

flow_scale = 1e4;   % Budget normalization factor

% Toolbox options
PrintFreq = 10;
SaveFreq = inf;
SIMU_RESOLVE=0; SIMU_INTERP=1; % Use interp for fast simulation

% Shocks
var_shock A R p;
shock_num = 8;
% Markov process of shocks (from Mendoza AER (2010):
shock_trans =  [0.51363285, 0.16145114, 0.07773228, 0.02443373, 0.16145114, 0.03201937, 0.02443373, 0.00484575;...
                0.03201937, 0.64306463, 0.00484575, 0.09732026, 0.16145114, 0.03201937, 0.02443373, 0.00484575;...
                0.07773228, 0.02443373, 0.51363285, 0.16145114, 0.02443373, 0.00484575, 0.16145114, 0.03201937;...
                0.00484575, 0.09732026, 0.03201937, 0.64306463, 0.02443373, 0.00484575, 0.16145114, 0.03201937;...
                0.03201937, 0.16145114, 0.00484575, 0.02443373, 0.64306463, 0.03201937, 0.09732026, 0.00484575;...
                0.03201937, 0.16145114, 0.00484575, 0.02443373, 0.16145114, 0.51363285, 0.02443373, 0.07773228;...
                0.00484575, 0.02443373, 0.03201937, 0.16145114, 0.09732026, 0.00484575, 0.64306463, 0.03201937;...
                0.00484575, 0.02443373, 0.03201937, 0.16145114, 0.02443373, 0.07773228, 0.16145114, 0.51363285];
AMean = 7 % Scale to match average output with SCU preference in Mendoza (2010)
A = AMean*[0.986595988257; 1.013404011625; 0.986595988257; 1.013404011625; 0.986595988257; 1.013404011625; 0.986595988257;
     1.013404011625];
R = [1.064449652804; 1.064449652804; 1.064449652804; 1.064449652804; 1.106970405389; 1.106970405389; 1.106970405389; 1.106970405389];
p = [0.993455002946; 0.993455002946; 1.062225686321; 1.062225686321; 0.993455002946; 0.993455002946; 1.062225686321; 1.062225686321];

% State
var_state cTilde k;

cTilde_min = 100;
cTilde_max = 300;
cTilde_shift = 10;
cTilde_pts = 30;
cTilde = linspace(cTilde_min,cTilde_max,cTilde_pts);

k_min = 600;
k_max = 900;
k_shift = 10;
k_pts = 30;
k = linspace(k_min,k_max,k_pts);

% Initial period
var_policy_init L v;
inbound_init L 0 1000;
inbound_init v 0 1000;

var_aux_init flow q d;

model_init;
  % Some trivial equations
  w = L^(omega-1)*(1+tau);
  Y = A* k^gamma * L^alpha * v^eta;
  F_1 = gamma*Y/k;
  F_2 = alpha*Y/L;
  F_3 = eta*Y/v;

  NL = L^omega/omega;
```

```
  flow = Y - p*v - phi*(R-1)*(w*L+p*v) - NL*(1+tau);

  lambda = cTilde^(-sigma);
  q = 1;
  d = F_1-delta;

  equations;
    F_2 - w*(1+phi*(R-1));
    F_3 - p*(1+phi*(R-1));
  end;
end;

% Interpolation
var_interp flow_interp q_plus_d_interp;
% Initial
initial flow_interp flow/flow_scale;
initial q_plus_d_interp q + d;
% Transition
flow_interp = flow/flow_scale;
q_plus_d_interp = q + d;

% Policy
ADAPTIVE_FACTOR = 1.5;
var_policy L v mu nbNext kNext cTildeNext[8];
inbound L 0 1000 adaptive(ADAPTIVE_FACTOR);
inbound v 0 1000 adaptive(ADAPTIVE_FACTOR);
inbound mu 0 1;
inbound nbNext 0 2000 adaptive(ADAPTIVE_FACTOR);
inbound kNext 0 1000 adaptive(ADAPTIVE_FACTOR);
inbound cTildeNext 0.0 500 adaptive(ADAPTIVE_FACTOR);

% Extra variables returned
var_aux lambda flow q d c Y bNext b inv nx gdp b_gdp nx_gdp lev wkcptl;
var_output c Y cTildeNext q mu inv kNext bNext b v nx gdp b_gdp nx_gdp lev wkcptl;

model;
  % Output and marginal product
  Y = A* k^gamma * L^alpha * v^eta;
  F_1 = gamma*Y/k;
  F_2 = alpha*Y/L;
  F_3 = eta*Y/v;
  w = L^(omega-1)*(1+tau);

  % Some calculations
  lambda = cTilde^(-sigma)/(1+tau);
  inv = kNext - k*(1-delta) + a/2*(kNext-k)^2/k;
  q = 1+a*(kNext-k)/k;
  d = F_1 - delta + a/2*(kNext-k)^2/(k^2);
  qb = 1/R;

  % Transform back bNext
  bNext = (nbNext + phi*R*(w*L+p*v) - kappa*q*kNext)/qb;

  % Interpolation
  [flow_future',q_plus_d_future'] = GDSGE_INTERP_VEC'(cTildeNext',kNext);
  flow_future' = flow_future'*flow_scale;

  % some named equations
  lambda_future' = cTildeNext'^(-sigma)/(1+tau);
  euler_bond_residual = -1 + mu + beta*GDSGE_EXPECT{lambda_future'}/(qb*lambda);
  euler_capital_residual = -1 + kappa*mu + beta*GDSGE_EXPECT{lambda_future'*q_plus_d_future'}/(q*lambda);
  % consistency equation
  cTilde_consis' = flow_future' + bNext - cTildeNext'*(1+tau);

  % Extra variables returned
  NL = L^omega/omega;
  flow = Y - p*v - phi*(R-1)*(w*L+p*v) - qb*bNext - inv - NL*(1+tau);
  c = cTilde + NL;
  b = cTilde*(1+tau) - flow;
  gdp = Y-p*v;
  b_gdp = b / gdp;
  nx = qb*bNext - b + phi*(R-1)*(w*L+p*v);
  nx_gdp = nx / gdp;
  lev = (qb*bNext-phi*R*(w*L+p*v)) / (q*kNext);
```

```
  wkcptl = phi*(w*L+p*v);

  equations;
    euler_bond_residual;
    nbNext*mu;
    euler_capital_residual;
    F_2 - w*(1+phi*(R-1)+mu*phi*R);
    F_3 - p*(1+phi*(R-1)+mu*phi*R);
    cTilde_consis';
  end;
end;

simulate;
  num_periods = 50000;
  num_samples = 24;
  initial cTilde 200;
  initial k 800;
  initial shock ceil(shock_num/2);

  var_simu c Y q inv b mu bNext v nx gdp b_gdp nx_gdp lev wkcptl;
  cTilde' = cTildeNext';
  k' = kNext;
end;
```

B.2 Barro et al. (2017)

```
% Parameters
parameters rho nu mu gamma1 gamma2;
period_length=0.25;          % a quarter
rho = 0.02*period_length;    % time preference
nu = 0.02*period_length;     % replacement rate
mu = 0.5;                    % population share of agent 1
P = 1-exp(-.04*period_length);   % disaster probability
B = -log(1-.32);             % disaster size
g = 0.025*period_length;     % growth rate
gamma1 = 3.1;
gamma2 = 50;

% Shocks
var_shock yn;
shock_num = 2;
shock_trans = [1-P,P;
               1-P,P];
yn = exp([g,g-B]);

% States
var_state omega1;
Ngrid = 501;
omega1 = [linspace(0,0.03,200),linspace(0.031,0.94,100),linspace(0.942,0.995,Ngrid-300)];

p = (1-nu)/(rho+nu);
pn = p;
Re_n = (1+pn)*yn/p;
% Endogenous variables, bounds, and initial values
var_policy shr_x1 Rf omega1n[2]
inbound shr_x1 0 1;             % agent 1's equity share
inbound Rf Re_n(2) Re_n(1);     % risk-free rate
inbound omega1n 0 1.02;         % state next period

% Other equilibrium variables
var_aux x1 x2 K1 b1 c1 c2 log_u1 log_u2 expectedRe;

% Implicit state transition functions
var_interp log_u1future log_u2future;
log_u1future = log_u1;
log_u2future = log_u2;
initial log_u1future (rho+nu)/(1+rho)*log((rho+nu)/(1+rho)) + (1-nu)/(1+rho)*log((1-nu)/(1+rho));
initial log_u2future (rho+nu)/(1+rho)*log((rho+nu)/(1+rho)) + (1-nu)/(1+rho)*log((1-nu)/(1+rho));

model;
```

```
    c1 = (rho+nu)/(1+rho);
    c2 = (rho+nu)/(1+rho);
    p = (1-nu)/(rho+nu);
    pn = p;

    log_u1n' = log_u1future'(omega1n');
    log_u2n' = log_u2future'(omega1n');
    u1n' = exp(log_u1n');
    u2n' = exp(log_u2n');

    Re_n' = (1+pn)*yn'/p;
    x1 = shr_x1*(Rf/(Rf - Re_n(2)));

    % Market clearing for bonds:
    b1 = omega1*(1-x1)*(1-c1)*(1+p);
    b2 = -b1;
    x2 = 1 - b2/((1-omega1)*(1-c2)*(1+p));
    K1 = x1*(1-c1)*omega1*(1+p)/p;
    K2 = x2*(1-c2)*(1-omega1)*(1+p)/p;

    R1n' = x1*Re_n' + (1-x1)*Rf;
    R2n' = x2*Re_n' + (1-x2)*Rf;

    % Agent 1's FOC wrt equity share:
    eq1 = GDSGE_EXPECT{Re_n'*u1n'^(1-gamma1)*R1n'^(-gamma1)} / GDSGE_EXPECT{Rf*u1n'^(1-gamma1)*R1n'^(-gamma1)} - 1;

    % Agent 2's FOC wrt equity share:
    log_u2n_R2n_gamma' = log_u2n'*(1-gamma2) - log(R2n')*gamma2;
    min_log_u2n_R2n_gamma = GDSGE_MIN{log_u2n_R2n_gamma'};
    log_u2n_R2n_gamma_shifted' = log_u2n_R2n_gamma' - min_log_u2n_R2n_gamma;
    eq2 = GDSGE_EXPECT{Re_n'*exp(log_u2n_R2n_gamma_shifted')} / GDSGE_EXPECT{Rf*exp(log_u2n_R2n_gamma_shifted')} - 1;

    % Consistency for omega:
    omega_future_consis' = K1 - nu*(K1-mu) + (1-nu)*Rf*b1/(yn'*(1+pn)) - omega1n';

    % Update the utility functions:
    ucons1 = ((rho+nu)/(1+rho))*log(c1) + ((1-nu)/(1+rho))*log(1-c1);
    ucons2 = ((rho+nu)/(1+rho))*log(c2) + ((1-nu)/(1+rho))*log(1-c2);
    log_u1 = ucons1 + (1-nu)/(1+rho)/(1-gamma1)*log(GDSGE_EXPECT{(R1n'*u1n')^(1-gamma1)});
    log_u2 = ucons2 + (1-nu)/(1+rho)/(1-gamma2)*( log(GDSGE_EXPECT{R2n'*exp(log_u2n_R2n_gamma_shifted')}) + min_log_u2n_R2n_gamma );

    expectedRe = GDSGE_EXPECT{Re_n'};

    equations;
      eq1;
      eq2;
      omega_future_consis';
    end;
end;

simulate;
    num_periods = 10000;
    num_samples = 50;
    initial omega1 .67;
    initial shock 1;

    var_simu Rf K1 b1 expectedRe;

    omega1' = omega1n';
end;
```

B.3 Guvenen (2009)

```
% Parameters
parameters beta alpha rhoh rhon theta delta mu xsi chi a1 a2 Kss Bbar bn_shr_lb bn_shr_ub varianceScale;

beta = 0.9966;      % discount factor
alpha = 6;          % risk aversion
rhoh = 1/.3;        % inv IES for stockholders
rhon = 1/.1;        % inv IES for non-stockholders
```

```
theta = .3;          % capital share
delta = .0066;       % depreciation rate
mu = .2;             % participation rate
xsi = .4;            % adjustment cost coefficient
chi = .005;          % leverage ratio
a1 = (((delta^(1/xsi))*xsi)/(xsi-1));
a2 = (delta/(1-xsi));
Kss = ((1/beta-1+delta)/theta)^(1/(theta-1));
Bbar = -0.6*(1-theta)*Kss^theta; %borrowing constraint
varianceScale = 1e4;

TolEq = 1e-4;
INTERP_ORDER = 4; EXTRAP_ORDER = 4;
PrintFreq = 100;
SaveFreq = inf;

% Shocks
var_shock Z;
shock_num = 15;
phi_z = 0.984;  % productivity AR(1)
mu_z = 0;
sigma_e = 0.015/(1+phi_z^2+phi_z^4).^0.5;
[z,shock_trans,~]=tauchen(shock_num,mu_z,phi_z,sigma_e,2);
Z = exp(z);

% States
var_state K bn_shr;
K_pts = 10;
K = exp(linspace(log(.84*Kss),log(1.2*Kss),K_pts));

bn_shr_lb = (1-mu)*Bbar/(chi*Kss);
bn_shr_ub = (chi*Kss - mu*Bbar)/(chi*Kss);
b_pts = 30;
bn_shr = linspace(bn_shr_lb,bn_shr_ub,b_pts);

% Last period
var_policy_init c_h c_n;

inbound_init c_h 1e-6 100;
inbound_init c_n 1e-6 100;

var_aux_init Y W vh vn vhpow vnpow Ps Pf Div Eulerstock Eulerbondh Eulerbondn Inv dIdK Eulerf;

model_init;
  Y = Z*(K^theta);
  W = (1-theta)*Z*(K^theta);
  resid1 = 1 - (W + (bn_shr*chi*Kss/(1-mu)))/c_n; % c_n: individual consumption
  resid2 = 1 - (W + (Div/mu) + ((1-bn_shr)*chi*Kss/mu))/c_h; % c_h: individual consumption
  vh = ((1-beta)*(c_h^(1-rhoh)))^(1/(1-rhoh));
  vn = ((1-beta)*(c_n^(1-rhon)))^(1/(1-rhon));
  vhpow = vh^(1-alpha);
  vnpow = vn^(1-alpha);
  Pf = 0;
  Ps = 0;
  Div = Y - W - (1-Pf)*chi*Kss;  % investment is zero

  Eulerstock = (vh^(rhoh-alpha))*(c_h^-rhoh)*(Ps + Div);
  Eulerbondh = (vh^(rhoh-alpha))*(c_h^-rhoh);
  Eulerbondn = (vn^(rhon-alpha))*(c_n^-rhon);

  Inv = 0;
  Knext = 0;
  dIdK = (Inv/K) - (1/a1)*(xsi/(xsi-1))*(Inv/(K*((1/a1)*((Knext/K)-(1-delta)-a2))))*(Knext/K);
  Eulerf = (vh^(rhoh-alpha))*(c_h^-rhoh)*(theta*Z*(K^(theta-1)) - dIdK);

  equations;
     resid1;
     resid2;
  end;
end;

var_interp EEulerstock_interp EEulerbondh_interp EEulerbondn_interp EEulerf_interp Evh_interp Evn_interp EPD_interp EPD_square_interp;
initial EEulerstock_interp shock_trans*reshape(Eulerstock,shock_num,[]);
initial EEulerbondh_interp shock_trans*reshape(Eulerbondh,shock_num,[]);
```

```
initial EEulerbondn_interp shock_trans*reshape(Eulerbondn,shock_num,[]);
initial EEulerf_interp shock_trans*reshape(Eulerf,shock_num,[]);
initial Evh_interp shock_trans*reshape(vhpow,shock_num,[]);
initial Evn_interp shock_trans*reshape(vnpow,shock_num,[]);
initial EPD_interp shock_trans*reshape(Div,shock_num,[]);
initial EPD_square_interp shock_trans*reshape(Div.^2,shock_num,[]) / varianceScale;

EEulerstock_interp = shock_trans*Eulerstock;
EEulerbondh_interp = shock_trans*Eulerbondh;
EEulerbondn_interp = shock_trans*Eulerbondn;
EEulerf_interp = shock_trans*Eulerf;
Evh_interp = shock_trans*vhpow;
Evn_interp = shock_trans*vnpow;
EPD_interp = shock_trans*(Ps+Div);
EPD_square_interp = shock_trans*(Ps+Div).^2 / varianceScale;

% Endogenous variables, bounds, and initial values
var_policy c_h c_n Ps Pf Inv bn_shr_next lambdah lambdan;

inbound c_h 1e-3 100;
inbound c_n 1e-3 100;
inbound Ps 1e-3 500;
inbound Pf 1e-3 10;
inbound Inv 1e-9 100;
inbound bn_shr_next bn_shr_lb bn_shr_ub;
inbound lambdah 0 2;
inbound lambdan 0 2;

% Other equilibrium variables
var_aux Y W b_h b_n Div dIdKp Eulerstock Eulerbondh Eulerbondn dIdK Eulerf vhpow vnpow omega PDratio Rs R_ep vh vn Knext std_ExcessR
     SharpeRatio;

model;
  Y = Z*(K^theta);              % output
  W = (1-theta)*Z*(K^theta);    % Wage = F_l
  Div = Y - W - Inv - (1-Pf)*chi*Kss;            % dividends

  Knext = (1-delta)*K + (a1*((Inv/K)^((xsi-1)/xsi))+a2)*K;
  dIdKp = (1/a1)*(xsi/(xsi-1))*(Inv/(K*((1/a1)*((Knext/K)-(1-delta)-a2))));

  b_h = (1-bn_shr)*chi*Kss/mu;
  b_n = bn_shr*chi*Kss/(1-mu);

  [EEulerstock_future,EEulerbondh_future,EEulerbondn_future,EEulerf_future,Evh_future,Evn_future,EPD_future,EPD_square_future] =
       GDSGE_INTERP_VEC(shock,Knext,bn_shr_next);
  EPD_square_future = EPD_square_future*varianceScale;

  vh =  ((1-beta)*(c_h^(1-rhoh)) + beta*(Evh_future^((1-rhoh)/(1-alpha))))^(1/(1-rhoh));
  vn =  ((1-beta)*(c_n^(1-rhon)) + beta*(Evn_future^((1-rhon)/(1-alpha))))^(1/(1-rhon));

  Eulerstock = (vh^(rhoh-alpha))*(c_h^-rhoh)*(Ps + Div);
  Eulerbondh = (vh^(rhoh-alpha))*(c_h^-rhoh);
  Eulerbondn = (vn^(rhon-alpha))*(c_n^-rhon);

  dIdK = (Inv/K) - (1/a1)*(xsi/(xsi-1))*(Inv/(K*((1/a1)*((Knext/K)-(1-delta)-a2))))*(Knext/K);
  Eulerf = (vh^(rhoh-alpha))*(c_h^-rhoh)*(theta*Z*(K^(theta-1)) - dIdK);

  vhpow = vh^(1-alpha);
  vnpow = vn^(1-alpha);

  omega = (Ps+Div+ mu*b_h)/(Ps+Div+chi*Kss);
  PDratio = Ps/Div;
  Rs = EPD_future/Ps;
  R_ep = Rs - 1/Pf;
  % The following inline implements
  % std_ExcessR = (GDSGE_EXPECT{(PD_future'/Ps - Rs)^2})^0.5;
  std_ExcessR = (EPD_square_future/(Ps^2) + Rs^2 - 2*EPD_future*Rs/Ps)^0.5;
  SharpeRatio = R_ep/std_ExcessR;

  % Equations:
  err_bdgt_h = 1 - (W + (Div/mu) + b_h - Pf*(chi*Kss*(1-bn_shr_next)/mu))/c_h; % these are individual consumptions
  err_bdgt_n = 1 - (W + b_n - Pf*(bn_shr_next*chi*Kss/(1-mu)))/c_n;
  foc_stock = 1 - (beta*EEulerstock_future*(Evh_future^((alpha-rhoh)/(1-alpha))))/((c_h^(-rhoh))*Ps);
  foc_bondh = 1 - (beta*EEulerbondh_future*(Evh_future^((alpha-rhoh)/(1-alpha))) + lambdah)/((c_h^(-rhoh))*Pf);
```

```
  foc_bondn = 1 - (beta*EEulerbondn_future*(Evn_future^((alpha-rhon)/(1-alpha))) + lambdan)/((c_n^-rhon)*Pf);
  foc_f = 1 - (beta*EEulerf_future*(Evh_future^((alpha-rhoh)/(1-alpha))))/((c_h^(-rhoh))*dIdKp);

  slack_bn = lambdan*(bn_shr_next - bn_shr_lb);    %mun_lw*bn_shr_next;
  slack_bh = lambdah*(bn_shr_ub - bn_shr_next);    %mun_up*(1-bn_shr_next);

  equations;
    err_bdgt_h;
    err_bdgt_n;
    foc_stock;
    foc_bondh;
    foc_bondn;
    foc_f;
    slack_bn;
    slack_bh;
  end;

end;

simulate;
  num_periods = 10000;
  num_samples = 100;

  initial K Kss;
  initial bn_shr 0.5;
  initial shock 2;

  var_simu Y c_h c_n Inv Ps Div Pf bn_shr_next Knext omega PDratio Rs R_ep SharpeRatio std_ExcessR;

  K' = Knext;
  bn_shr' = bn_shr_next;
end;
```

B.4 Bianchi (2011)

```
% Toolbox options
USE_ASG=1; USE_SPLINE=0;
AsgMaxLevel = 10;
AsgThreshold = 1e-4;

% Parameters
parameters r sigma eta kappaN kappaT omega beta;
r = 0.04;
sigma = 2;
eta = 1/0.83 - 1;
kappaN = 0.32;
kappaT = 0.32;
omega = 0.31;
beta = 0.91;

% States
var_state b;
bPts = 101;
bMin=-0.5;
bMax=0.0;
b=linspace(bMin,bMax,bPts);

% Shocks
var_shock yT yN;
yPts = 4;
shock_num=16;

yTEpsilonVar = 0.00219;
yNEpsilonVar = 0.00167;
rhoYT = 0.901;
rhoYN = 0.225;

[yTTrans,yT] = markovappr(rhoYT,yTEpsilonVar^0.5,1,yPts);
[yNTrans,yN] = markovappr(rhoYN,yNEpsilonVar^0.5,1,yPts);

```

```
shock_trans = kron(yNTrans,yTTrans);
[yT,yN] = ndgrid(yT,yN);
yT = exp(yT(:)');
yN = exp(yN(:)');

% Define the last-period problem
var_policy_init dummy;
inbound_init dummy -1.0 1.0;

var_aux_init c lambda;
model_init;
  cT = yT + b*(1+r);
  cN = yN;
  c = (omega*cT^(-eta) + (1-omega)*cN^(-eta))^(-1/eta);
  partial_c_partial_cT = (omega*cT^(-eta) + (1-omega)*cN^(-eta))^(-1/eta-1) * omega * cT^(-eta-1);
  lambda = c^(-sigma)*partial_c_partial_cT;

  equations;
    0;
  end;
end;

% Implicit state transition functions
var_interp lambda_interp;
initial lambda_interp lambda;
lambda_interp = lambda;

% Endogenous variables, bounds, and initial values
var_policy nbNext mu cT pN;
inbound nbNext 0.0 10.0;
inbound mu 0.0 1.0;
inbound cT 0.0 10.0;
inbound pN 0.0 10.0;

var_aux c lambda bNext;
var_output bNext pN;

model;
  % Non tradable market clear
  cN = yN;

  % Transform variables
  bNext = nbNext - (kappaN*pN*yN + kappaT*yT);
  % Interp future values
  lambdaFuture' = lambda_interp'(bNext);

  % Calculate Euler residuals
  c = (omega*cT^(-eta) + (1-omega)*cN^(-eta))^(-1/eta);
  partial_c_partial_cT = (omega*cT^(-eta) + (1-omega)*cN^(-eta))^(-1/eta-1) * omega * cT^(-eta-1);
  lambda = c^(-sigma)*partial_c_partial_cT;
  euler_residual = 1 - beta*(1+r) * GDSGE_EXPECT{lambdaFuture'}/lambda - mu;

  % Price consistent
  price_consistency = pN - ((1-omega)/omega)*(cT/cN)^(eta+1);

  % budget constraint
  budget_residual = b*(1+r)+yT+pN*yN - (bNext+cT+pN*cN);

  equations;
    euler_residual;
    mu*nbNext;
    price_consistency;
    budget_residual;
  end;
end;

simulate;
  num_periods = 1000;
  num_samples = 100;
  initial b 0.0
  initial shock 1;
  var_simu c pN;
  b' = bNext;
end;
```

B.5 Guerrieri et al. (2022)

```
% Parameters
parameters beta rho sigma phi nbar delta Abar;
beta = 0.99;     % discount factor
rho = 0.75;      % 1/rho intratemporal elasticity
sigma = 0.5;     % 1/sigma intertemporal elasticity
phi = 0.2;       % share of sector 1
nbar = 1;        % normal labor endowment
delta = 0.5;     % fraction of labor endowment during crisis
Abar = 0.3;      % borrowing limit
TolEq = 1e-8;    % Solve with high adccuracy

% Shocks
var_shock n1;
shock_num = 2;
pi2 = 0.5;       % the pandemic lasts for 2 quarters
freq = 0.005;    % frequency of pandemic: 0.5 percent of the time.
pi1 = 1 - (freq/(1-freq))*(1-pi2);
shock_trans = [pi1,1-pi1;
               1-pi2,pi2];
n1 = [nbar,delta*nbar];

% Endogenous States
var_state a1;
Ngrid = 301;
a1_lb = -Abar;
a1_ub = (1-phi)*Abar/phi;
a1 = linspace(a1_lb,a1_ub,Ngrid);

% Last period
var_policy_init c1_shr;
inbound_init c1_shr 0 1;
var_aux_init P1 log_lambda1 log_lambda2;

model_init;
   c1_1 = c1_shr*(phi*n1)/phi;
   c2_1 = (1-c1_shr)*(phi*n1)/(1-phi);
   c1_2 = c1_shr*((1-phi)*nbar)/phi;
   c2_2 = (1-c1_shr)*((1-phi)*nbar)/(1-phi);

   Y = (phi*n1^(1-rho) + (1-phi)*nbar^(1-rho))^(1/(1-rho));
   lambda1 = (c1_shr/phi*Y)^(-sigma)*(Y/nbar)^rho;
   lambda2 = ((1-c1_shr)/(1-phi)*Y)^(-sigma)*(Y/nbar)^rho;
   log_lambda1 = log(lambda1);
   log_lambda2 = log(lambda2);

   % price of good 1
   P1 = ((c1_1/phi)/(c1_2/(1-phi)))^(-rho);
   % wage of sector 1
   W1 = P1;
   budget1_resid = P1*c1_1 + c1_2 - W1*n1 - a1;

  equations;
     budget1_resid;
  end;
end;

var_interp log_lambda1_interp log_lambda2_interp;
initial log_lambda1_interp log_lambda1;
initial log_lambda2_interp log_lambda2;
% Updates
log_lambda1_interp = log_lambda1;
log_lambda2_interp = log_lambda2;

% Endogenous variables, bounds, and initial values
var_policy c1_shr a1n mu1 mu2 r;
inbound c1_shr 0 1;
inbound a1n -Abar (1-phi)*Abar/phi;
inbound mu1 0 1;
inbound mu2 0 1;
inbound r -0.5 0.5;

% Other equilibrium variables
```

```
var_aux a2 P1 log_lambda1 log_lambda2;

model;
   a2 = -a1*phi/(1-phi);
   c1_1 = c1_shr*(phi*n1)/phi;
   c2_1 = (1-c1_shr)*(phi*n1)/(1-phi);
   c1_2 = c1_shr*((1-phi)*nbar)/phi;
   c2_2 = (1-c1_shr)*((1-phi)*nbar)/(1-phi);

   Y = (phi*n1^(1-rho) + (1-phi)*nbar^(1-rho))^(1/(1-rho));
   lambda1 = (c1_shr/phi*Y)^(-sigma)*(Y/nbar)^rho;
   lambda2 = ((1-c1_shr)/(1-phi)*Y)^(-sigma)*(Y/nbar)^rho;
   log_lambda1 = log(lambda1);
   log_lambda2 = log(lambda2);

   % price of good 1
   P1 = ((c1_1/phi)/(c1_2/(1-phi)))^(-rho);
   % wage of sector 1
   W1 = P1;

   log_lambda1Future' = log_lambda1_interp'(a1n);
   log_lambda2Future' = log_lambda2_interp'(a1n);
   lambda1Future' = exp(log_lambda1Future');
   lambda2Future' = exp(log_lambda2Future');

   budget1_resid = P1*c1_1 + c1_2 + a1n/(1+r) - W1*n1 - a1;
   euler_residual = 1 - beta*(1+r) * GDSGE_EXPECT{lambda1Future'}/lambda1 - mu1;
   euler_residua2 = 1 - beta*(1+r) * GDSGE_EXPECT{lambda2Future'}/lambda2 - mu2;

   a2n = -a1n*phi/(1-phi);
   slackness1 = mu1*(a1n + Abar);
   slackness2 = mu2*(a2n + Abar);
  equations;
    budget1_resid;
    euler_residual;
    euler_residua2;
    slackness1;
    slackness2;
  end;
end;

simulate;
  num_periods = 10000;
  num_samples = 20;
  initial a1 0;
  initial shock 1;

  var_simu a2 P1 r c1_shr;

  a1' = a1n;
end;
```

B.6 A multi-country business cycle model solved with adaptive sparse grids

```
% Number of countries
#define N 15

% Use asg
USE_SPLINE = 0; USE_ASG = 1;
AsgMinLevel = 2;
AsgMaxLevel = 2;
AsgThreshold = 1e-2;
AsgOutputMaxLevel = 4;
AsgOutputThreshold = 1e-2;
PrintFreq = 1;
SaveFreq = inf;
TolEq = 1e-5;
```

```
SIMU_RESOLVE = 0; SIMU_INTERP = 1;

% Parameters
parameters beta sigma alpha delta numCountries;
beta  = 0.99;           % discount factor
sigma = 2.0;            % CRRA coefficient
alpha = 0.36;           % capital share
delta = 0.025;          % depreciation rate
numCountries = N;

% Innovations
parameters rho_z sigma_z prob_z;
rho_z = 0.9;        % First-order autocorrelation coefficient
sigma_z = 0.01;     % std of innovation of productivity
prob_z = 1/(2*N);   % Probability for each perturbation; used for integration

% Pareto weights
parameters omega_vec;
omega_vec = ones(1,N);

% Endogenous States
Kss  = (alpha/(1/beta - 1 + delta))^(1/(1-alpha));
KPts = 2;   % placeholder
KMin = Kss*0.9;
KMax = Kss*1.1;
zMin = -0.05;
zMax = 0.05;
zPts = 2;   % placeholder
#for i=1:N
var_state z#i;
z#i = linspace(zMin,zMax,zPts); % placeholder
#end
#for i=1:N
var_state K#i;
K#i = linspace(KMin,KMax,KPts); % placeholder
#end

var_policy_init lambda;
inbound_init lambda 0 1;
model_init;
  % Construct vector from state
  vector z_vec[N];
  vector K_vec[N];
  #for i=1:N
    z_vec(#i) = z#i;
    K_vec(#i) = K#i;
  #end

  % Back out consumption from the marginal utility
  vector c_vec[N];
  for i=1:N
    c_vec(i) = (lambda / omega_vec(i))^(-1/sigma);
  end

  % Calcualte resource residual
  resource_resid = 0;
  for i=1:N
    resource_resid = resource_resid + exp(z_vec(i))*K_vec(i)^alpha + (1-delta)*K_vec(i) - c_vec(i) - K_vec(i);
  end
  equations;
    resource_resid;
  end;
end;

% Interp
var_interp lambda_interp;
initial lambda_interp lambda;
lambda_interp = lambda;

% Endogenous variables as unknowns of equations
var_policy K_next[N]
inbound K_next 25 45;
var_policy lambda;
inbound lambda 0 1;
```

```
model;
  % Construct vector from state
  vector z_vec[N];
  vector K_vec[N];
  #for i=1:N
    z_vec(#i) = z#i;
    K_vec(#i) = K#i;
  #end

  % Prepare future states, under no perturbation
  vector state_next[2*N];
  for i=1:N
    % First N states is future productivity
    state_next(i) = rho_z*z_vec(i);
    % Next N states if future capital
    state_next(N+i) = K_next(i);
  end

  % Calculate marginal product of capital with no perturbation
  vector mpk_next[N];
  for i=1:N
    mpk_next(i) = alpha*exp(state_next(i)) * K_next(i)^(alpha-1);
  end

  % Do a sparse quadrature by perturbing productivity once at a time
  vector euler_residual[N];
  for i=1:N
    euler_residual(i) = 0;
  end
  for i=1:N
    % Modifty future productivity of the perturbed country
    % Positive innovation
    state_next(i) = rho_z*z_vec(i) + sigma_z*sqrt(numCountries);
    mpk_next(i) = alpha*exp(state_next(i)) * K_next(i)^(alpha-1);
    lambda_future = GDSGE_INTERP_ARRAY_adouble(shock,state_next_GDSGE_GRID);
    #for j=1:N
        euler_residual(#j) = euler_residual(#j) + prob_z*(-lambda + beta*lambda_future*(mpk_next(#j)+(1-delta)));
    #end

    % Negative innovation
    state_next(i) = rho_z*z_vec(i) - sigma_z*sqrt(numCountries);
    mpk_next(i) = alpha*exp(state_next(i)) * K_next(i)^(alpha-1);
    lambda_future = GDSGE_INTERP_ARRAY_adouble(shock,state_next_GDSGE_GRID);
    #for j=1:N
        euler_residual(#j) = euler_residual(#j) + prob_z*(-lambda + beta*lambda_future*(mpk_next(#j)+(1-delta)));
    #end

    % Revert the perturbation back
    state_next(i) = rho_z*z_vec(i);
    mpk_next(i) = alpha*exp(state_next(i)) * K_next(i)^(alpha-1);
  end

  % Back out consumption from the marginal utility
  vector c_vec[N];
  for i=1:N
    c_vec(i) = (lambda / omega_vec(i))^(-1/sigma);
  end
  % Calcualte resource residual
  resource_resid = 0;
  for i=1:N
    resource_resid = resource_resid + exp(z_vec(i))*K_vec(i)^alpha + (1-delta)*K_vec(i) - c_vec(i) - K_next(i);
  end

  % Record variables
  #for i=1:N
    K_next_output#i = K_next(#i);
    c#i = c_vec(#i);
  #end

  equations;
    for i=1:N
        euler_residual(i);
    end
```

```
    resource_resid;
  end;
end;

simulate;
  num_samples = 24;
  num_periods = 1000;
  #for i=1:N
    initial shock 1;
    initial z#i 0;
    initial K#i Kss;
    var_simu K_next_output#i c#i;
    z#i' = z_exo(:,GDSGE_t+1,#i);
    K#i' = K_next_output#i;
  #end
end;
```

C User Manual

The user manual, online compiler, gmod files and other examples can be found on the toolbox's website: http://www.gdsge.com.