# Online Appendix (User's Guide) for *FiPIt*: A Simple, Fast Global Method for Solving Models with Two Endogenous States & Occasionally Binding Constraints

Enrique G. Mendoza
University of Pennsylvania, NBER & PIER

Sergio Villalvazo
University of Pennsylvania

# 1 Introduction

This Appendix provides a user's guide for the Matlab codes that implement the *FiPIt* algorithm. It describes how the various steps of the algorithm presented in Section 3 of the paper are undertaken in the computer programs. The programs are distributed in a zip file labeled *MendozaVillalvazoFiPItCode* available online. The main directory of this file has the same name, and it contains two folders named *FiPIt* and *Mfiles*. The main Matlab script is named mainFiPIt.m and is located in the *FiPIt* folder. This folder also includes the output files as well as script files used to generate various output components (moments, charts, etc.). The mainFiPIt.m program calls several function scripts that are stored in the *MFiles* folder. Table A1 provides a list of all the files, their location and contents.

Table A1: Files Included in the *MendozaVillalvazoFiPItCode* Directory

| *Name* | *Folder Location* | *Description* |
|---|---|---|
| mainFiPIt.m | FiPIt/ | Main script |
| script1_Moments.m | FiPIt/ | Script to obtain moments in Table 2 and 3 |
| script2_PolicyPlot.m | FiPIt/ | Script to produce Figure 3 |
| script3_Simulation.m | FiPIt/ | Script to produce Figure 4 |
| script4_TableDiffAmpl.m | FiPIt/ | Script to obtain Table 5 and the probability of Sudden Stops |
| fBiLinearInterpolation.m | MFiles/ | Function for bi-linear interpolation |
| fFiPIt_Cons.m | MFiles/ | Function to find consumption rule from the Euler Equation for bonds using FiPIt |
| fFiPIt_EulerError.m | MFiles/ | Function to compute Euler Errors |
| fFiPIt_MuBond.m | MFiles/ | Function to compute Lagrange multiplier of ad-hoc debt limit in the RBC model |
| fFiPIt_MuHat.m | MFiles/ | Function that solves for credit constraint mutiplier ratio using a non-linear solver |
| fFiPIt_PriceK.m | MFiles/ | Function to compute price of capital from the capital Euler Equation using FiPIt |
| fMarkov.m | MFiles/ | Function to generate Markov chain simulation |
| BondPolicyBoth.pdf | FiPIt/ | Figure 3.a |
| CapitalPolicyBoth.pdf | FiPIt/ | Figure 3.b |
| ErgoBondDist_RBC.pdf | FiPIt/ | Figure 1.a |
| ErgoBondDist_SS.pdf | FiPIt/ | Figure 2.a |
| ErgoCapitalDist_RBC.pdf | FiPIt/ | Figure 1.b |
| ErgoCapitalDist_SS.pdf | FiPIt/ | Figure 2.b |
| ErgoDist_RBC.pdf | FiPIt/ | Figure 1.c |
| ErgoDist_SS.pdf | FiPIt/ | Figure 2.c |
| EventStudySS_mean.pdf | FiPIt/ | Figure 4 |
| MuBoth.pdf | FiPIt/ | Figure 3.d |
| PriceKPolicyBoth.pdf | FiPIt/ | Figure 3.c |

The output of mainFiPIt.m is stored in a .mat file. To solve the variant of the model in which the credit constraint never binds (denoted the RBC model), set the valuef of $\kappa$ high enough so as to ensure that this is the case. Under our calibration, $\kappa > 1$ is sufficient. The .mat file with the RBC solution is named solFiPIt_RBC.mat. To solve the Sudden Stops (SS) model, set $\kappa < 1$. The .mat file with this solution is named solFiPIt_SS.mat. The long-run moments of these two models reported in Tables 3 and 4 of the paper are computed using script1_Moments.m, choosing to comment in or out either line 14 or 15 to load the corresponding .mat file with the RBC or SS solution. Similarly, to produce the policy function plots run script2_PolicyPlot.m. The event study

plots are produced by running script3_Simulation.m, and both the probability of Sudden Stops and the data for Table 5 are produced by running script4_TableDiffAmpl.m.

## 2 Contents of the mainFiPIt.m program

The mainFiPIt.m file is divided into 5 cells, each one including comments describing how the contents of each cell relate to each of the seven algorithm steps described in subsection 3.2 of the paper. The itemized step numbers labeled in bold typeface below match the step numbers in the paper description, with the line in the Matlab code in which the step is executed indicated in parenthesis.

**Cell 1. Parameterization & State Space:** Sets the model's parameter values, creates the discrete grids of bonds and capital, defines the Markov processes of shocks, and sets the values of program parameters that define the method to solve for capital price, the convergence criteria, the maximum number of iterations and the updating coefficients for decision rule conjectures between one iteration and the next. The endogenous states are foreign bonds $b$ and domestic capital $k$. The exogenous states are included in $s$, which denotes a triple of shocks $s \equiv (A, R, p)$ that includes TFP ($A$), the world interest rate ($R$) and the price of imported inputs ($p$). The realization set for shock triples $s \in \mathbf{S}$ comes from the discretization of the stochastic processes of the shocks, which is typically done using Tauchen's quadrature method. Here, we take $\mathbf{S}$ and the associated Markov transition probability matrix from Mendoza (2010), where each shock has two realizations and hence $\mathbf{S}$ has eight triples. For the endogenous states, the algorithm defines grids with a total of *nBondGrid* nodes for bonds and *nCapitalGrid* nodes for capital. The state space has $nBondGrid \times nCapitalGrid \times 8$ elements and is defined by all $(b, k, s) \in \mathbf{B} \otimes \mathbf{K} \otimes \mathbf{S}$. The conditional statement starting in Line 82 adjusts the bonds grid when the SS model is being solved to make sure the collateral constraint binds before the lower bound of the grid. The recursive equilibrium is defined by a set of recursive functions for allocations $[b'(b,k,s), k'(b,k,s), c(b,k,s), L(b,k,s), v(b,k,s)]$, prices $[w(b,k,s), q(b,k,s), d(b,k,s)]$ and the multipliers $[\lambda(b,k,s), \mu(b,k,s)]$. The model and program parameters are listed in Table A2.

Table A2: Parameter Values

| | | |
|---|---|---|
| *Calibrated parameters* | | |
| $\sigma$ | coefficient of relative risk aversion | 2.0 |
| $\omega$ | labor elasticity coefficient | 1.8461 |
| $\beta$ | discount factor | 0.92 |
| $a$ | capital adjustment costs coefficient | 2.75 |
| $\phi$ | fraction of input costs requiring working capital | 0.2579 |
| $\delta$ | depreciation rate | 0.088 |
| $\alpha$ | labor share in gross output | 0.59 |
| $\eta$ | imported inputs share in gross output | 0.10 |
| $\gamma$ | capital share in gross output | 0.31 |
| $\tau$ | tax on consumption | 0.17 |
| $A$ | average TFP | 6.982 |
| | | |
| *Algorithm parameters* | | |
| $\rho^b$ | Updating weight for bonds decision rule | 1.00 |
| $\rho^\mu$ | Updating weight for multiplier ratio | 1.00 |
| $\rho^q$ | Updating weight for price of capital | 0.30 |
| $\varepsilon^f$ | Function convergence criterion | 10e-4 |

**Cell 2. Initial Conjectures, Array Definitions & Non-linear Solver Options:** This cell defines the initial conjectures for the equilibrium recursive functions. Following the notation in the paper, at any iteration $j$ the initial conjectured functions are denoted $\hat{q}_j(b,k,s)$ for the price of capital, $\hat{B}_j(b,k,s) \equiv \hat{b}'_j(b,k,s)$ for the decision rule for bonds, and $\hat{\tilde{\mu}}_j(b,k,s) \equiv \mu_j(b,k,s)/\lambda_j(b,k,s)$ for the multiplier ratio. This cell also initializes the arrays for other variables and constructs a function that sets the optimization options for the non-linear solver used later in the program to solve for allocations when the credit constraint binds.

**Step 1.** (Line 106) Sets the first-iteration recursive function conjectures to $\hat{B}_0(b,k,s) = b$, $\hat{q}_0(b,k,s) = 1$ and $\hat{\tilde{\mu}}_0(b,k,s)) = 0$ for all $(b,k,s) \in \mathbf{B} \otimes \mathbf{K} \otimes \mathbf{S}$. The instructions after those initialize the arrays for other variables, the first-iteration value of the convergence metric for the recursive functions (*nMaxDif*) and the iterations counter (*nIter*), and they also define the function *pSolverOpt* to set the options for Matlab's *fsolve* non-linear solver used later in the code when solving for allocations in states in which $\tilde{\mu} > 0$.

**Cell 3. Main Loop Executing Iterations on Equilibrium Recursive Functions:** The *While* loop starting in line 143 executes the successive iterations on the equilibrium recursive functions for bonds, price of capital and multiplier ratio. The current iteration number ($j$) is stored in

the integer *nIter*, and the value of the convergence metric attained in iteration *nIter* is stored in *nMaxDif*.

**Step 2.** (Line 146) Generates decision rules for capital, investment, factor allocations, gross output and consumption in iteration $j$ implied by the conjectures $\hat{q}_j(b,k,s), \hat{B}_j(b,k,s), \hat{\tilde{\mu}}_j(b,k,s)$:

$$\begin{aligned}
K_j(b,k,s) =& \frac{k}{a}\left[\hat{q}_j(b,k,s) - 1 + a\right] \\
\tilde{i}_j(b,k,s) =& (K_j(b,k,s) - k)\left[1 + \frac{a}{2}\left(\frac{K_j(b,k,s)-k}{k}\right)\right] - \delta k \\
v_j(b,k,s) =& \left\{\frac{Ak^{\gamma}\eta^{\frac{\omega-\alpha}{\omega}}\frac{\alpha}{1+\tau}^{\frac{\alpha}{\omega}}}{p^{\frac{\omega-\alpha}{\omega}}\left[1+\phi(R-1)+\hat{\tilde{\mu}}_j(b,k,s)\phi R\right]}\right\}^{\frac{\omega}{\omega(1-\eta)-\alpha}} \\
L_j(b,k,s) =& \left\{\frac{\alpha}{\eta(1+\tau)}pv_j((b,k,s)\right\}^{\frac{1}{\omega}} \\
y_j(b,k,s) =& Ak^{\gamma}L_j(b,k,s)^{\alpha}v_j(b,k,s)^{\eta}
\end{aligned}$$

$$\begin{aligned}
(1+\tau)c_j(b,k,s) = & \, y_j(b,k,s) - pv_j(b,k,s) - \phi(R-1)\left[(1+\tau)L_j(b,k,s)^{\omega} + pv_j(b,k,s)\right] \\
& - \tilde{i}_j(b,k,s) - \frac{\hat{B}_j(b,k,s)}{R} + b
\end{aligned}$$

The code uses here the same set of expressions for the RBC and SS solutions. For the latter, the values of factor allocations, gross output and consumption vary with $\tilde{\mu}(\cdot)$, whereas in the RBC solution they do not because $\tilde{\mu}(\cdot) = 0$ always. Note also that since $\tilde{\mu}(\cdot)$ is always set to zero in the first iteration, the first-iteration results of this step are identical when solving either the RBC or SS models. When solving the RBC model, $\tilde{\mu}(\cdot)$ remains zero in all iterations, but when solving the SS model, $\tilde{\mu}(\cdot) > 0$ in states in which the credit constraint binds.

**Step 3.1** (Line 166) Assume the collateral constraint does not bind. Solve for new decision rules (indexed $j+1$) for labor, intermediate goods and output. Since the constraint is assumed to be non-binding, these decision rules are the same in RBC and SS solutions:

$$\begin{aligned}
v_{j+1}(b,k,s) =& \left\{\frac{Ak^{\gamma}\eta^{\frac{\omega-\alpha}{\omega}}\frac{\alpha}{1+\tau}^{\frac{\alpha}{\omega}}}{p^{\frac{\omega-\alpha}{\omega}}\left[1+\phi(R-1)\right]}\right\}^{\frac{\omega}{\omega(1-\eta)-\alpha}} \\
L_{j+1}(b,k,s) =& \left\{\frac{\alpha}{\eta(1+\tau)}pv_{j+1}(b,k,s)\right\}^{\frac{1}{\omega}} \\
y_{j+1}(b,k,s) =& Ak^{\gamma}L_{j+1}(b,k,s)^{\alpha}v_{j+1}(b,k,s)^{\eta}
\end{aligned}$$

**Steps 3.2 & 3.3.** (Line 181) Solve for the $j+1$ consumption and bonds decision rules using the bonds' Euler Equation and the resource constraint. For each $(b,k,s)$ in the state space,

consumption is solved for using the fFiPIt_Cons.m function located in the *Mfiles* folder. This function finds the new consumption rule by solving "directly" from the Euler equation, as explained in Step 3.2 of the algorithm description in the paper:

$$c_{j+1}(b,k,s) = \left\{ \beta RE \left[ \left( c_j(\hat{B}_j(b,k,s), K_j(b,k,s), s') - \frac{L_j(\hat{B}_j(b,k,s), K_j(b,k,s), s')^{\omega}}{\omega} \right)^{-\sigma} \right] \right\}^{-\frac{1}{\sigma}} + \frac{L_{j+1}(b,k,s)^{\omega}}{\omega}$$

fFiPIt_Cons.m calls the function fBiLinearInterpolation.m, also in the *Mfiles* folder, in order to find the values of $c_j(\hat{B}_j(b,k,s), K_j(b,k,s), s')$, $L_j(\hat{B}_j(b,k,s), K_j(b,k,s), s')$, which are determined using bi-linear interpolation because $\hat{B}_j(b,k,s)$ and $K_j(b,k,s)$ are not on the nodes of the bonds and capital grids in general. Once $c_{j+1}(b,k,s)$ is determined, the new bonds policy function $B_{j+1}(b,k,s)$ is solved for using the resource constraint, and the implied leverage ratio is computed (i.e. the value of $\frac{-q_t^b b_{t+1} + \phi R_t(w_t L_t + p_t v_t)}{\kappa q_t k_{t+1}}$).

**Step 3.4.** (Line 192) Check if collateral constraint binds using the new decision (j+1-indexed) decision rules:

$$\frac{B_{j+1}(b,k,s)}{R} - \phi R\left[(1+\tau)L_{j+1}(b,k,s)^{\omega} + pv_{j+1}(b,k,s)\right] + \kappa \hat{q}_j(b,k,s) K_j(b,k,s) \geq 0$$

Line 202 evaluates if there are $(b,k,s)$ states for which the new bonds decision rule is below the lower bound of the bonds grid. In these cases, the lower bound is a binding ad-hoc debt limit. The bonds decision rule is re-set equal to this debt limit, the consumption decision rule is re-set to the value implied by the resource constraint, and we also compute the associated Lagrange multiplier for the binding ad-hoc debt limit.

**Step 4.** (Line 217) This step is only executed when solving the SS model and only for states $(b,k,s)$ in which the constraint was found to be binding in Step 3.4, because these are the only states in which the decision rules depend on $\tilde{\mu}$. This step solves for $\tilde{\mu}_{j+1}(b,k,s)$ by applying Matlab's *fsolve* root finder to a function formed using the fFiPIt_MuHat.m script located in the *Mfiles* folder. fFiPIt_MuHat.m forms equation (48) in the paper. It uses the $j$-indexed functions for consumption and labor to form the expected value in the right-hand-side of eq. (48), which requires the same bi-linear interpolation method used to solve for $c_{j+1}$ in step 3.2. The solver uses the optimization options set in *pSolverOpt* as defined in Cell 1 and returns the value

of $\tilde{\mu}_{j+1}(b,k,s)$. The solver uses these options: optimoptions('fsolve','Display','off','TolFun',1e-18). A small tolerance convergence criterion is needed in order to attain convergence of the recursive functions and small Euler errors. We use as initial condition (*vInitX*) the current iteration's initial conjecture $\tilde{\mu}_j(b,k,s)$. After $\tilde{\mu}_{j+1}(b,k,s)$ is solved for, we compute the associated j+1 values of the decision rules using eqs. (43)-(46) in the paper. Keep in mind that there are many variations of occasionally binding constraints for which the constrained allocations and the multiplier of the binding constraint can be solved separately, in which case there is no need to use a non-linear solver in this step. Two cases explored in the paper are one in which working capital is removed from the collateral constraint and one in which the credit constraint is set to a constant value instead of the value of collateral (see p. 16 and p. 34 of the paper). This makes the *FiPIt* algorithm significantly faster.

**Step 5.** (Line 250) This step is just a comment noting that at this point in the code we have solved the new ($j+1$-indexed) optimal decision rules for all $(b,k,s)$ in the state space conditional on the conjectured $\hat{q}_j(b,k,s)$ function.

**Step 6.** (Line 252) Compute the new pricing function. This step is coded so as to allow the user to choose one of the two alternatives to compute the pricing function described in Steps 6.1 and 6.2 of the paper. The former uses fixed-point iteration, the latter finds $q$ as the forward solution of the capital Euler equation. The fixed-point iteration (forward) solution is chosen by setting $pFixPointPriceK == 1$ ($pFixPointPriceK == 0$) in the algorithm parameters of Cell 1. In both cases, we solve for $q_{j+1}(b,k,s)$ using the fFiPIt_PriceK.m script located in the *Mfiles* folder. This script solves the following equation, which is eq. (51) in the paper (we use $(\cdot)$ to denote $(b,k,s)$ so as to shorten the notation):

$$q_{j+1}(b,k,s) = \frac{\beta E_t\left[\left(c_{j+1}\left(B_{j+1}(\cdot),K_j(\cdot),s'\right)-\frac{L_{j+1}(B_{j+1}(\cdot),K_j(\cdot),s')^{\omega}}{\omega}\right)^{-\sigma}\left[d'\left(B_{j+1}(\cdot),K_j(\cdot),s'\right)+\hat{q}_j\left(B_{j+1}(\cdot),K_j(\cdot),s'\right)\right]\right]}{\left(c_{j+1}(\cdot)-\frac{L_{j+1}(\cdot)^{\omega}}{\omega}\right)^{-\sigma}\left(1-\kappa\tilde{\mu}_{j+1}(\cdot)\right)}$$

where

$$d'\left(B_{j+1}(\cdot),K_j(\cdot),s'\right)=\gamma A K_j(\cdot)^{\gamma-1}L_{j+1}\left(B_{j+1}(\cdot),K_j(\cdot),s'\right)^{\alpha}v_{j+1}\left(B_{j+1}(\cdot),K_j(\cdot),s'\right)^{\eta} - \delta+\frac{a}{2}\frac{\left(K_j(B_{j+1}(\cdot),K_j(\cdot),s')-K_j(\cdot)\right)^2}{K_j(\cdot)^2}$$

When solving by fixed-point iteration, the above Euler equation solves directly for $q_{j+1}(\cdot)$, since all the terms in the right-hand-side of the expression are known at this point in the code. The

equation is solved once and the solution passed on as the new pricing function. Note that in forming the conditional expectation, we use $j$-indexed conjectures of the price of capital and the capital decision rule (since their $j+1$ values are not known), but the rest of the relevant recursive functions are indexed $j+1$ (since they have been solved for in the previous steps of the algorithm). As before, bi-linear interpolation is used to determine the values of all the functions that have $(B_{j+1}(\cdot), K_j(\cdot))$ as arguments (the $t+1$ variables in the conditional expectation of the Euler equation), since those functions are only known at grid nodes.[1] When solving by forward solution, fFiPIt_PriceK.m is used repeatedly to iterate on the above capital Euler equation until $q_{j+1}(\cdot)$ and $\hat{q}_j(\cdot)$ converge, but keeping all the other functions unchanged. For these iterations, the iteration counter is the integer *nIterInner*, and the value of the convergence metric at iteration *nIterInner* is denoted *nMaxDifInner*. The convergence criterion is the value assigned to the parameter *nTolInner* in Cell 1.

**Step 6.1.** (Line 271) If $pFixPointPriceK = 1$, then the first solution for $q_{j+1}(b,k,s)$ generated for each $(b,k,s)$ using fFiPIt_PriceK.m is retained as the new pricing function.

**Step 6.2.** (Line 274) If $pFixPointPriceK = 0$ (which is executed by the *else* instruction when pFixPointPriceK = 1 is not valid), then fFiPIt_PriceK.m is used to generate the new values of $q_{j+1}(b,k,s)$ for each $(b,k,s)$ as we iterate to convergence on the capital pricing function.

**Step 7.** (Line 281) Check convergence and update conjectures. The convergence criterion is given by $nMaxDif \leq \varepsilon^f$, where $nMaxDif$ is the following convergence metric:

$$nMaxDif = \max\left\{|q_{j+1}(b,k,s) - \hat{q}_j(b,k,s)|, |B_{j+1}(b,k,s) - \hat{B}_j(b,k,s)|, |\tilde{\mu}_{j+1}(b,k,s) - \hat{\tilde{\mu}}_j(b,k,s)|\right\}$$

$\forall (b,k,s) \in \mathbf{B} \otimes \mathbf{K} \otimes \mathbf{S}$. The value of $\varepsilon^f$ is defined by setting the program parameter *nTol* in Cell 1. If convergence is attained, the recursive equilibrium has been solved and the results are stored in either the solFiPIt_SS.mat file for the SS model or the solFiPIt_RBC.mat file for the RBC model. If convergence is not attained, then generate new conjectures as follows:

$$\hat{x}_{j+1}(b,k,s) = (1-\rho^x)\hat{x}_j(b,k,s) + \rho^x x_{j+1}(b,k,s)$$

[1] For evaluating dividends, we found that the algorithm performs better if we interpolate the functions that enter in the definition of dividends individually and then generate the value of dividends, instead of first defining dividends and then interpolating the dividends function.

for $x = [q, B, \tilde{\mu}]$ and some $0 \leq \rho^x$. $\hat{x}_j(b,k,s)$ in the right-hand-side of this expression represents the initial conjectures used in the current iteration, while $\hat{x}_{+1}j(b,k,s)$ in the left-hand-side denotes the new conjectures for the next iteration. Use $0 < \rho^x < 1$ ($\rho^x > 1$) for the particular function $x(\cdot)$ that is not converging (converging too slowly). The values of the $\rho^x$ coefficients are set with the parameters *nUpdateGuessB* and *nUpdateGuessPK* in Cell 1.[2] Return to Step 2 (Line 146) using the new conjectures for the next iteration.

**Cell 4.** Compute Euler Equation Errors.

The solution of the recursive equilibrium is completed when the program exits Cell 3. The next two cells generate two important objects based on the model solution. First, Cell 4 computes the errors of the Euler equations of bonds and capital using the fFiPIt_EulerError.m function located in the *Mfiles* folder. Then Cell 5 computes the ergodic distribution of bonds, capital and shocks. To compute the Euler errors, the Euler equations are evaluated at the equilibrium solutions rather than used to solve for the equilibrium. Hence, fFiPIt_EulerError.m uses the equilibrium functions (the last solutions generated by the functions that converged according to the tolerance criterion) in all the relevant terms of the Euler equations.

**Cell 5.** Compute the Ergodic Distribution.

We compute the ergodic distribution of $(b,k,s)$ by iterating to convergence on the law of motion of the conditional transition probabilities from $(b,k,s)$ (denoted $M_j(b,k,s)$) to $(b',k',s')$ (denoted $M_{j+1}(b',k',s')$) $\forall (b,k,s),(b',k',s') \in \mathbf{B}\otimes\mathbf{K}\otimes\mathbf{S}$. The initial guess (called *mErgDistGuess* in line 404) is a uniform distribution. The law of motion is formed using the decision rules for capital and bonds and the exogenous Markov process of the shocks. Since we have solved for approximately continuous decision rules using bi-linear interpolation, we use a standard modification of this law of motion adjusted for the fact that decision rules do not yield values on the nodes of the bonds and capital grids in general. For every $(b,k,s)$ we find $b_L \leq B(b,k,s) \leq b_U$ and $k_L \leq K(b,k,s) \leq k_U$, where $b_L, b_U, k_L, k_U$ are the grid points closest to $B(\cdot)$ and $K(\cdot)$ . Then we iterate on the conditional distributions as follows:

$$M_{j+1}(b_L,k_L,s') = \sum_s Pr[s'|s] M_j(b,k,s) \left(\frac{b_U - B(b,k,s)}{b_U - b_L}\right)\left(\frac{k_U - K(b,k,s)}{k_U - k_L}\right)$$

[2]We set $\rho^B = \rho^{\tilde{\mu}} = 1$ and $\rho^q = 0.3$ because this produced the best convergence performance, but this can change with other parameterizations or in other applications of the algorithm.

$$M_{j+1}(b_L, k_U, s') = \sum_s Pr[s'|s] M_j(b,k,s) \left(\frac{b_U - B(b,k,s)}{b_U - b_L}\right)\left(\frac{K(b,k,s) - k_L}{k_U - k_L}\right)$$

$$M_{j+1}(b_U, k_L, s') = \sum_s Pr[s'|s] M_j(b,k,s) \left(\frac{B(b,k,s) - b_L}{b_U - b_L}\right)\left(\frac{k_U - K(b,k,s)}{k_U - k_L}\right)$$

$$M_{j+1}(b_U, k_U, s') = \sum_s Pr[s'|s] M_j(b,k,s) \left(\frac{B(b,k,s) - b_L}{b_U - b_L}\right)\left(\frac{K(b,k,s) - k_L}{k_U - k_L}\right)$$

The convergence criterion is $\max|M_{j+1}(b,k,s) - M_j(b,k,s)| < \epsilon_{Dist} \quad \forall (b,k,s) \in \mathbf{B} \otimes \mathbf{K} \otimes \mathbf{S}$, with the value of $\epsilon_{Dist}$ set by the parameter *nTolDist* in Cell 1.

# 3 Auxiliary Notes

- **Interpolation:** Bi-linear interpolation can be done using the "interp2" Matlab function, but we found that programming the interpolation directly improved the performance of the code. We determine first the interpolation nodes, and then apply the standard bi-linear interpolation rule. The scripts that implement the functions interpolations determine the relevant interpolation nodes and then perform the bi-linear interpolation. To determine the interpolation nodes, for each $(b,k,s)$, create first vectors with the differences $h^b(b,k,s) = \hat{B}_j(b,k,s) - b$ and $h^k(b,k,s) = K_j(b,k,s) - k$, then find the location of the smallest positive difference and smallest negative difference (i.e. the difference closest to zero from below) in these vectors. For example, for the interpolation nodes over the $b$ dimension $(b_n, b_{n+1})$, find the locations of argmin$h^b(b,k,s)$ for $h^b(b,k,s) \geq 0$ and argmax$h^b(b,k,s)$ for $h^b(b,k,s) \leq 0$. $b_n$ is the location of the argmin and $b_{n+1}$ is the location of the argmax. Once the interpolation nodes are found, the interpolation is executed by calling the fBiLinearInterpolation.m function located in the *Mfiles* folder. The scripts also make these adjustments when the interpolated functions return decision rule values outside the state space: Use extrapolation if $K_j(b,k,s)$ returns a value below (above) the first node $k^1$ (last node $k^{NCapitalGrid}$) and also if $\hat{B}_j(b,k,s)$ returns a value above the last node $b^{NBondGrid}$, but for $\hat{B}_j(b,k,s) < b^1$ evaluate the functions at $b^1$, because the lower bound on bonds represents an ad-hoc debt limit used for calibration.

- **Parallelization:** There are several loops that run faster in parallel, using *parfor* instead of *for*. This can be done with all loops that do not need to run sequentially. The outmost loop controlling the iterations of the policy and pricing functions needs to be executed sequentially, but several others can be parallelized. *Parfor* can be used in Step 2, 3, 4, and 6. For

the *FiPIt* variant of Step 6 a sequential sum is needed to attain convergence. We included comments in the code indicating specific loops where *parfor* was used. Using *parfor* requires Matlab's Parallel Computing Toolbox. Note also that setting the number of workers to the largest feasible (i.e. the number of processors) does not necessarily minimize execution time, particularly in machines with several processors. In various computers with more than 16 processors, we found that using 6 or 7 workers produced the fastest execution times.

- **Invalid allocations:** Rule out allocations with non-positive arguments in the utility function. These are cases such that, at any iteration and for a given triple $(b, k, s)$ the conjectured functions (indexed by $j$) or the unconstrained or constrained new decision rules (indexed by $j+1$) yield $c - L^{\omega}/\omega \leq 0$. In these cases, the solution of consumption when the constraint does not bind and/or of the multiplier $\tilde{\mu}$ when it binds cannot be obtained because they involve the fractional exponent $1/\sigma$ (for $\sigma > 1$), which requires a positive base. Note that this requirement is stricter than feasibility, because it is not just that the allocations are technologically feasible, they also need to avoid hitting the Inada condition of the CRRA utility function. In the *mainFiPIt.m* program, this causes an error that stops execution at the point in which the first attempt to solve for a state of nature with $c - L^{\omega}/\omega \leq 0$ is encountered. As explained in the paper (see p. 14 and p. 16), however, the *FiPIt* algorithm has the advantage that starting from initial conjectures $\hat{B}_0(b,k,s)$, $\hat{q}_0(b,k,s)$, $\hat{\tilde{\mu}}_0(b,k,s))$ such that the implied labor and consumption decision rules satisfy $c_0(\cdot) - \frac{L_0(\cdot)^{\omega}}{\omega} > 0$, implies that $c_j(\cdot) - \frac{L_j(\cdot)^{\omega}}{\omega} > 0$ for any iteration $j > 0$ . For the baseline calibration and all six variations we solved for, the initial conditions $\hat{B}_0(b,k,s) = b$, $\hat{q}_0(b,k,s) = 1$ and $\hat{\tilde{\mu}}_0(b,k,s)) = 0$ satisfied this condition.