ONLINE APPENDIX

A Machine Learning Projection Method for Macro-Finance Models

VYTAUTAS VALAITIS AND ALESSANDRO T. VILLA

A Optimal Fiscal Policy with Epstein-Zin Preferences

This appendix presents the general model with N bonds and Epstein-Zin preferences, describes the computation algorithm, and presents the equilibrium dynamics of the Epstein-Zin model with two bonds.

The Household's Problem Households have Epstein-Zin preferences where the instantaneous utility comes from consumption (c_t) and leisure (l_t). The parameter ρ controls the substitutability in time and the parameter γ controls the attitude towards risk.[25] When $\rho = \gamma$, preferences collapse to CRRA, where consumption and leisure are additively separable. Preferences are

$$V_t = [(1-\beta)U(c_t, l_t)^{1-\rho} + \beta(\mathbb{E}_t V_{t+1}^{1-\gamma})^{\frac{1-\rho}{1-\gamma}}]^{\frac{1}{1-\rho}}.$$

The time endowment is equal to 1; therefore, hours worked are $h_t = 1 - l_t$. The household cash-on-hand consists of (i) the after-tax labor income and (ii) the current bond holdings. This can be either consumed or spent to purchase new bonds. The budget constraint (BC) is

$$c_t + p_t b_{t+1} = b_t + (1-\tau_t)h_t.$$

Defining $W_t = b_t$ and $R_{t+1} = W_{t+1}/p_t b_{t+1} = 1/p_t$, the BC can be rewritten as

$$\begin{aligned} c_t + p_t W_{t+1} &= W_t + (1-\tau_t)h_t \\ \implies W_{t+1} &= R_{t+1}(W_t - c_t + (1-\tau_t)h_t). \end{aligned}$$

[25] When the instantaneous utility includes leisure, the relative risk aversion is not γ but $1-(1-\gamma)(1-\rho)$, see Swanson (2012) or Swanson and Rudebusch (2012) for a detailed explanation.

Given the redefinition of the BC, the household's problem can be rewritten as

$$V_t(W_t) = \max_{c_t, h_t}[(1-\beta)U(c_t, 1-h_t)^{1-\rho} + \beta(\mathbb{E}_t V_{t+1}(W_{t+1})^{1-\gamma})^{\frac{1-\rho}{1-\gamma}}]^{\frac{1}{1-\rho}},$$
$$W_{t+1} = R_{t+1}(W_t - c_t + (1-\tau_t)h_t).$$

Define the certainty equivalent (CE) as $\mathcal{R}_t(V_{t+1}) \equiv (\mathbb{E}_t V_{t+1}^{1-\gamma})^{\frac{1}{1-\gamma}}$. The optimality condition for consumption (FOC_c) is

$$V_t^\rho \left((1-\beta)(1-\rho)U_t^{-\rho}U_{c,t} - \beta(1-\rho)(\mathbb{E}_t V_{t+1}^{1-\gamma})^{\frac{\gamma-\rho}{1-\gamma}}\mathbb{E}_t\left[V_{t+1}^{-\gamma}R_{t+1}V_{W,t+1}\right]\right) = 0$$
$$\implies (1-\beta)U_t^{-\rho}U_{c,t} = \beta\mathcal{R}_t^{\gamma-\rho}\mathbb{E}_t\left[V_{t+1}^{-\gamma}R_{t+1}V_{W,t+1}\right].$$

The optimality condition for labor supply (FOC_h) is

$$V_t^\rho \left(-(1-\beta)(1-\rho)U_t^{-\rho}U_{l,t} + \beta(1-\rho)(\mathbb{E}_t V_{t+1}^{1-\gamma})^{\frac{\gamma-\rho}{1-\gamma}}\mathbb{E}_t\left[V_{t+1}^{-\gamma}(1-\tau_t)R_{t+1}V_{W,t+1}\right]\right) = 0$$
$$\implies (1-\beta)U_t^{-\rho}U_{l,t} = (1-\tau_t)\beta\mathcal{R}_t^{\gamma-\rho}\mathbb{E}_t\left[V_{t+1}^{-\gamma}R_{t+1}V_{W,t+1}\right].$$

The envelope condition is

$$V_{W,t} = V_t^\rho \beta\mathcal{R}_t^{\gamma-\rho}\mathbb{E}_t V_{t+1}^{-\gamma}R_{t+1}V_{W,t+1}.$$

Combine FOC_c with FOC_h to get

$$\frac{U_{l,t}}{U_{c,t}} = 1-\tau_t.$$

Combine FOC_c with the envelope condition to get

$$V_{W,t} = V_t^\rho(1-\beta)U_t^{-\rho}U_{c,t} \implies V_{W,t+1} = V_{t+1}^\rho(1-\beta)U_{t+1}^{-\rho}U_{c,t+1},$$

which implies

$$(1-\beta)U_t^{-\rho}U_{c,t} = \beta\mathcal{R}_t^{\gamma-\rho}\mathbb{E}_t\left[V_{t+1}^{-\gamma}R_{t+1}V_{t+1}^\rho(1-\beta)U_{t+1}^{-\rho}U_{c,t+1}\right].$$

Plugging this back in the FOC_c, rearranging and simplifying leads to the following inter-temporal Euler equation

$$1 = \beta \mathbb{E}_t \left[\mathcal{M}_t(V_{t+1}) \left(\frac{U_{t+1}}{U_t} \right)^{-\rho} \frac{U_{c,t+1}}{U_{c,t}} R_{t+1} \right],$$

where $\mathcal{M}_t(V_{t+1}) \equiv \left(\frac{V_{t+1}}{\mathcal{R}_t(V_{t+1})} \right)^{\rho-\gamma}$. The bond's price p_t is the expected value of the stochastic discount factor

$$p_t = \beta \mathbb{E}_t \left[\mathcal{M}_t(V_{t+1}) \left(\frac{U_{t+1}}{U_t} \right)^{-\rho} \frac{U_{c,t+1}}{U_{c,t}} \right].$$

This problem can be generalized to N maturities in a similar fashion to section 3.2. This yields one Euler equation for each maturity $i = 1, \cdots, N$ that reads:

$$p_t^i = \beta \mathbb{E}_t \left[\mathcal{M}_t(V_{t+i}) \left(\frac{U_{t+i}}{U_t} \right)^{-\rho} \frac{U_{c,t+i}}{U_{c,t}} \right].$$

Ramsey Problem

The Ramsey problem consists of finding $\{c_t, \{b_{t+1}^i\}_{i=1}^N, \mu_t, V_t\}_{t=0}^\infty$ in order to maximize the household's welfare taking household intra- and inter-temporal first order conditions as constraints. Hence, the Lagrangian of the problem is

$$\mathcal{L} = V_0 + \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \Bigg\{ \mu_t \Big(U_t^{-\rho} U_{c,t} s_t + \sum_{i=1}^{N} \mathbb{E}_t \beta^i b_{t+1}^i \mathcal{M}_t(V_{t+i}) U_{t+i}^{-\rho} U_{c,t+i} -$$
$$\sum_{i=1}^{N} \mathbb{E}_t \beta^{i-1} b_t^i U_{t+i-1}^{-\rho} U_{c,t+i-1} \mathcal{M}_t(V_{t+i-1}) \Big) + \sum_{i=1}^{N} \xi_{U,t}^i (B^U - b_{t+1}^i) + \sum_{i=1}^{N} \xi_{L,t}^i (b_{t+1}^i - B^L) \Bigg\}.$$

In addition, the Ramsey planner needs to respect the recursivity constraint for V_t

$$V_t = [(1-\beta) U(c_t, 1 - c_t - g_t)^{1-\rho} + \beta (\mathbb{E}_t V_{t+1}^{1-\gamma})^{\frac{1-\rho}{1-\gamma}}]^{\frac{1}{1-\rho}}.$$

Optimality Conditions

In order to calculate the first order condition with respect to c_t, it is necessary to calculate an expression for the derivative of welfare V_0 with respect to c_t. Note that V_0 contains all the consumption path from 0 throughout ∞, which we derive in subsection A. Knowing $\frac{\partial V_0}{\partial c_t(g^t)}$ the first order

condition with respect to consumption is[26]

$$V_0^{\rho}(1-\beta)\mathcal{X}_{0,t}U_t^{-\rho}\frac{\partial U_t}{\partial c_t(g^t)}+\mu_t\left(\frac{\partial U_t^{-\rho}U_{c,t}}{\partial c_t(g^t)}s_t+\frac{\partial s_t}{\partial c_t}U_t^{-\rho}U_{c,t}\right)+$$
$$\frac{\partial U_t^{-\rho}U_{c,t}}{\partial c_t(g^t)}\sum_{i=1}^{N}\left(\mu_{t-i}\mathcal{M}_{t-i}(V_t)-\mu_{t-i+1}\mathcal{M}_{t-i+1}(V_t)\right)b_{t-i+1}^i+\lambda_t^V V_t^{-\rho}(1-\beta)U_t^{-\rho}\frac{\partial U_t}{c_t(g^t)}=0,$$

where λ_t^V is the time-t Lagrange multiplier associated with the recursive constraint and $\mathcal{X}_{t_1,t_2}\equiv\prod_{k=1}^{t_2-t_1}\mathcal{M}_{t_1+k-1}(V_{t_1+k})$ with $\mathcal{X}_{t_1,t_2}\equiv 1,\forall t_2\leq t_1$. Note that $\mathcal{X}$ admits a recursive representation.[27] The first order condition with respect to b_{t+1}^i yields the following inter-temporal expression for the promise keeping Lagrange multiplier μ

$$\mu_t=\left[\mathbb{E}_t\mathcal{M}_t(V_{t+i})U_{t+i}^{-\rho}U_{c,t+i}\right]^{-1}\left[\mathbb{E}_t\mu_{t+1}\mathcal{M}_{t+1}(V_{t+i})U_{t+i}^{-\rho}U_{c,t+i}+\frac{\xi_t^U}{\beta^i}-\frac{\xi_t^L}{\beta^i}\right].$$

The first order condition with respect to V_t is

$$\beta^{t-i}\sum_{i=1}^{N}\pi(g^{t-i}|g^0)\beta^i\mu_{t-i}\pi(g^t|g^{t-i})b_{t-i+1}^iU_{c,t}U_t^{-\rho}\frac{\partial\mathcal{M}_{t-i}(V_t)}{\partial V_t(g^t)}-$$
$$\beta^{t-i+1}\sum_{i=1}^{N}\pi(g^{t-i+1}|g^0)\beta^{i-1}\mu_{t-i+1}\pi(g^t|g^{t-i+1})b_{t-i+1}^iU_{c,t}U_t^{-\rho}\frac{\partial\mathcal{M}_{t-i+1}(V_t)}{\partial V_t(g^t)}-$$
$$\lambda_t^V\beta^t\pi(g^t|g^0)+\beta^{t-1}\pi(g_{t-1}|g^0)\lambda_{t-1}^V\beta V_{t-1}^{\rho}\mathcal{R}_{t-1}(V_t)^{-\rho}\mathcal{M}_{t-1}(V_t)^{\frac{-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-1})=0,$$

which, after rearranging, yields the following recursion for λ_t^V[28]

$$\lambda_t^V=\sum_{i=1}^{N}\left(\mu_{t-i}\frac{\partial\mathcal{M}_{t-i}(V_t)}{\partial V_t(g^t)}-\mu_{t-i+1}\frac{\partial\mathcal{M}_{t-i+1}(V_t)}{\partial V_t(g^t)}\right)b_{t-i+1}^iU_{c,t}U_t^{-\rho}+\lambda_{t-1}^V\left(\frac{V_{t-1}}{V_t}\right)^{\rho}\mathcal{M}_{t-1}(V_t).$$

The remaining first order condition with respect to μ_t just gives back the inter-temporal government implementability constraint.

[26] With $\frac{\partial V_0}{\partial c_t(g^t)}=V_0^{\rho}\beta^t(1-\beta)\mathcal{X}_{0,t}\pi(g^t|g^0)U_t^{-\rho}\frac{\partial U_t}{\partial c_t(g^t)}$.

[27] $\mathcal{X}_{t_1,t_2}\equiv\prod_{k=1}^{t_2-t_1}\mathcal{M}_{t_1+k-1}(V_{t_1+k})=\mathcal{M}_{t_2-1}(V_{t_2})\prod_{k=1}^{t_2-t_1-1}\mathcal{M}_{t_1+k-1}(V_{t_1+k})=\mathcal{M}_{t_2-1}(V_{t_2})\mathcal{X}_{t_1,t_2-1}$.

[28] Where: $\frac{\partial\mathcal{M}_{t-i}(V_t)}{\partial V_t}=(\rho-\gamma)\frac{\mathcal{M}_{t-i}(V_t)}{V_t}\left[1-\mathcal{M}_{t-i}(V_t)^{\frac{1-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-i})\right]$.

1. Derivation of $\partial V_t/\partial c_{t+j}$

If $j < 0$:

$$\partial V_t/\partial c_{t+j} = 0.$$

If $j = 0$:

$$\frac{\partial V_t}{\partial c_t} = (1-\beta)V_t^{\rho}U_t^{-\rho}\frac{\partial U_t}{\partial c_t}.$$

If $j = 1$:

$$\begin{aligned}\frac{\partial V_t}{\partial c_{t+1}(g^{t+1})} &= V_t^{\rho}\beta\mathcal{R}_t(V_{t+1})^{\gamma-\rho}\pi(g_{t+1}|g^t)V_{t+1}^{-\gamma}\frac{\partial V_{t+1}}{\partial c_{t+1}(g^{t+1})}\\ &= V_t^{\rho}\beta\mathcal{R}_t(V_{t+1})^{\gamma-\rho}\pi(g_{t+1}|g^t)V_{t+1}^{-\gamma}\left((1-\beta)V_{t+1}^{\rho}U_{t+1}^{-\rho}\frac{\partial U_{t+1}}{\partial c_{t+1}(g^{t+1})}\right)\\ &= V_t^{\rho}\beta(1-\beta)\mathcal{M}_t(V_{t+1})\pi(g_{t+1}|g^t)U_{t+1}^{-\rho}\frac{\partial U_{t+1}}{\partial c_{t+1}(g^{t+1})}.\end{aligned}$$

If $j = 2$:

$$\begin{aligned}\frac{\partial V_t}{\partial c_{t+2}(g^{t+2})} &= V_t^{\rho}\beta\mathcal{R}_t(V_{t+1})^{\gamma-\rho}\pi(g_{t+1}|g^t)V_{t+1}^{-\gamma}\frac{\partial V_{t+1}}{\partial c_{t+2}}\\ &= V_t^{\rho}\beta\mathcal{R}_t(V_{t+1})^{\gamma-\rho}\pi(g_{t+1}|g^t)V_{t+1}^{-\gamma}\left(V_{t+1}^{\rho}\beta(1-\beta)\mathcal{R}_{t+1}(V_{t+2})^{\gamma-\rho}\pi(g_{t+2}|g^{t+1})V_{t+2}^{\rho-\gamma}U_{t+2}^{-\rho}\frac{\partial U_{t+2}}{\partial c_{t+2}}\right)\\ &= V_t^{\rho}\beta^2(1-\beta)\prod_{k=1}^{2}\mathcal{M}_{t+k-1}(V_{t+k})\prod_{k=1}^{2}\pi(g_{t+k}|g^{t+k-1})U_{t+2}^{-\rho}\frac{\partial U_{t+2}}{\partial c_{t+2}(g^{t+2})}.\end{aligned}$$

For a generic $j \geq 0$:

$$\frac{\partial V_t}{\partial c_{t+j}(g^{t+j})} = V_t^{\rho}\beta^j(1-\beta)\mathcal{X}_{t,t+j}\pi(g^{t+j}|g^t)U_{t+j}^{-\rho}\frac{\partial U_{t+j}}{\partial c_{t+j}(g^{t+j})}.$$

2. Derivation of $\frac{\partial \mathcal{M}_{t-1}(V_t)}{\partial V_t}$

$$
\begin{aligned}
\frac{\partial \mathcal{M}_{t-1}(V_t)}{\partial V_t} &= (\rho-\gamma)\frac{\mathcal{M}_{t-1}(V_t)^{\frac{\rho-\gamma-1}{\rho-\gamma}}}{\mathcal{R}_{t-1}(V_t)^2}\left[\mathcal{R}_{t-1}(V_t) - V_t \underbrace{\mathcal{M}_{t-1}(V_t)^{\frac{-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-1})}_{\frac{\partial \mathcal{R}_{t-1}(V_t)}{\partial V_t}}\right] \\
&= (\rho-\gamma)\frac{\mathcal{M}_{t-1}(V_t)^{\frac{\rho-\gamma-1}{\rho-\gamma}}}{\left(\frac{V_t}{\mathcal{M}_{t-1}(V_t)^{\frac{1}{\rho-\gamma}}}\right)^2}\left[\mathcal{R}_{t-1}(V_t) - V_t\mathcal{M}_{t-1}(V_t)^{\frac{-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-1})\right] \\
&= (\rho-\gamma)\frac{\mathcal{M}_{t-1}(V_t)^{\frac{\rho-\gamma+1}{\rho-\gamma}}}{V_t^2}\left[\mathcal{M}_{t-1}(V_t)^{\frac{-1}{\rho-\gamma}}V_t - V_t\mathcal{M}_{t-1}(V_t)^{\frac{-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-1})\right] \\
&= (\rho-\gamma)\frac{\mathcal{M}_{t-1}(V_t)^{\frac{\rho-\gamma+1}{\rho-\gamma}}}{V_t}\left[\mathcal{M}_{t-1}(V_t)^{\frac{-1}{\rho-\gamma}} - \mathcal{M}_{t-1}(V_t)^{\frac{-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-1})\right] \\
&= (\rho-\gamma)\frac{\mathcal{M}_{t-1}(V_t)}{V_t}\left[1 - \mathcal{M}_{t-1}(V_t)^{\frac{1-\gamma}{\rho-\gamma}}\pi(g_t|g^{t-1})\right].
\end{aligned}
$$

Algorithm to Solve the Model with Epstein-Zin Preferences

Here we describe an algorithm to solve the model with Epstein-Zin preferences. Generally, it is very similar to the one used to solve the model with CRRA preferences, with the exception that there are additional state variables and additional terms that the neural network has to approximate. At every instant t, the information set is $\mathcal{I}_t = \{g_t, \{\{b^i_{t-k}\}_{k=0}^{N-1}\}_{i=1}^{N}, \{\mu_{t-k}\}_{k=1}^{N}, \{\lambda^V_{t-k}\}_{k=1}^{N}\}$. Consider projections of $\mathcal{R}_{t-i}(V_t)$, $\mathbb{E}_t\mathcal{M}_t(V_{t+i})U^{-\rho}_{t+i}U_{c,t+i}$, $\mathbb{E}_t\mu_{t+i}\mathcal{M}_{t+1}(V_{t+i})U^{-\rho}_{t+i}U_{c,t+i}$ and $\mathbb{E}_t\mathcal{M}_t(V_{t+i-1})U^{-\rho}_{t+i-1}U_{c,t+i-1}$ on $\mathcal{I}_t$. We model these relationships using one single-layer artificial neural network $\mathcal{ANN}(\mathcal{I}_t)$. For example, with two bonds we would have $4N+1$ inputs and 8 outputs.[29] In particular, use the following notations for each output:

$$
\begin{aligned}
\mathcal{ANN}^i_1 &= \mathcal{R}_{t-i}(V_t) \quad \text{for } i = \{1, N-1, N\}, \\
\mathcal{ANN}^i_2 &= \mathbb{E}_t\mathcal{M}_t(V_{t+i})U^{-\rho}_{t+i}U_{c,t+i} \quad \text{for } i = \{1, N\}, \\
\mathcal{ANN}^i_3 &= \mathbb{E}_t\mu_{t+i}\mathcal{M}_{t+1}(V_{t+i})U^{-\rho}_{t+i}U_{c,t+i} \quad \text{for } i = \{1, N\}, \\
\mathcal{ANN}^i_4 &= \mathbb{E}_t\mathcal{M}_t(V_{t+i-1})U^{-\rho}_{t+i-1}U_{c,t+i-1} \quad \text{for } i = \{N\}.
\end{aligned}
$$

Given starting values $\mu_{t-1} = \lambda^V_{-1} = 0$ and initial weights for $\mathcal{ANN}$, simulate a sequence of $\{c_t\}$, $\{\lambda^V_t\}$, $\{\mu_t\}$ as follows:

[29] One with maturity 1 and the other with maturity N.

1. Impose the Maliar moving bounds on all debts instruments, see Maliar and Maliar (2003). These bounds are particularly important and need to be tight and open slowly, since the neural network at the beginning can only make accurate predictions around zero debt - that is our initialization point. Proper penalty functions are used instead of the ξ terms to avoid out of bound solutions, see Faraglia et al. (2014) for more details.

2. Use forward-states on the following i equations

$$\forall i: \quad \mu_t = \left[\mathbb{E}_t \mathcal{ANN}_1^i(\mathcal{I}_{t+1})\right]^{-1} \left[\mathbb{E}_t \mathcal{ANN}_2^i(\mathcal{I}_{t+1}) + \frac{\xi_{U,t}^i}{\beta^i} - \frac{\xi_{L,t}^i}{\beta^i}\right].$$

3. Find λ_t^V, μ_t, c_t and $\{b_{t+1}^i\}_{i=1}^N$ that solve the following system of equations:

i. $$\lambda_t^V = \sum_{i=1}^N \left(\mu_{t-i}\frac{\partial \mathcal{M}_{t-i}(V_t)}{\partial V_t(g^t)} - \mu_{t-i+1}\frac{\partial \mathcal{M}_{t-i+1}(V_t)}{\partial V_t(g^t)}\right) b_{t-i+1}^i U_{c,t} U_t^{-\rho} + \lambda_{t-1}^V \left(\frac{V_{t-1}}{V_t}\right)^\rho \left(\frac{V_t}{\mathcal{ANN}_1^1(\mathcal{I}_{t+1})}\right)^{\rho-\gamma},$$

ii. $$V_0^\rho (1-\beta)\mathcal{X}_{0,t} U_t^{-\rho}\frac{\partial U_t}{\partial c_t(g^t)} + \mu_t\left(\frac{\partial U_t^{-\rho}U_{c,t}}{\partial c_t(g^t)}s_t + \frac{\partial s_t}{\partial c_t}U_t^{-\rho}U_{c,t}\right) +$$

$$\frac{\partial U_t^{-\rho}U_{c,t}}{\partial c_t(g^t)}\sum_{i=1}^N \left(\mu_{t-i}\left(\frac{V_t}{\mathcal{ANN}_1^i(\mathcal{I}_{t+1})}\right)^{\rho-\gamma} - \mu_{t-i+1}\left(\frac{V_t}{\mathcal{ANN}_1^{i-1}(\mathcal{I}_{t+1})}\right)^{\rho-\gamma}\right) b_{t-i+1}^i + \lambda_t^V V_t^{-\rho}(1-\beta)U_t^{-\rho}\frac{\partial U_t}{c_t(g^t)} = 0,$$

iii. $$\sum_{i=1}^N \beta^{i-1} b_t^i \mathcal{ANN}_4^i(\mathcal{I}_{t+1}) = s_t U_c^{-\rho} U_{c,t} + \sum_{i=1}^N \beta^i b_{t+1}^i \mathcal{ANN}_2^i(\mathcal{I}_{t+1}),$$

iv. $$\forall i: \quad \mu_t = \left[\mathbb{E}_t \mathcal{ANN}_1^i(\mathcal{I}_{t+1})\right]^{-1} \left[\mathbb{E}_t \mathcal{ANN}_2^i(\mathcal{I}_{t+1}) + \frac{\xi_{U,t}^i}{\beta^i} - \frac{\xi_{L,t}^i}{\beta^i}\right],$$

where

$$\frac{\partial \mathcal{M}_{t-i}(V_t)}{\partial V_t} = (\rho-\gamma)\frac{\left(\frac{V_t}{\mathcal{ANN}_1^i}\right)^{\rho-\gamma}}{V_t}\left[1 - \left(\frac{V_t}{\mathcal{ANN}_1^i}\right)^{1-\gamma} f_{g_t}(g_t|g_{t-i})\right],$$

$$V_t = [(1-\beta)U(c_t, 1-c_t-g_t)^{1-\rho} + \beta \mathcal{ANN}_1^1(\mathcal{I}_{t+1})^{1-\rho}]^{\frac{1}{1-\rho}},$$

and

$$\frac{\partial U_t}{\partial c_t} = U_{c,t} - U_{l,t}.$$

Note that $f_{g_t}(g_t|g_{t-1})$ is the conditional probability density of the exogenous g process.

4. Use the simulated sequence to train the $\mathcal{ANN}$ and re-start from point 1 till convergence of the predicted sequence over the realized one.

Numerical Results with Two Bonds and Epstein-Zin Preferences

Bhandari et al. (2017*b*) and Karantounias (2018) convincingly demonstrate that implications for optimal portfolios changes once the model contains preferences that match the asset prices. In this section, we explore the implications for optimal debt management once we change preferences to Epstein-Zin and add a shock that is orthogonal to the government expenditure process. In particular, we add an endowment shock z_t, such that $l_t = z_t - h_t$.[30] Figure 4 shows that in this setting the allocation shares are around equal among different maturities, and little portfolio re-balancing happens in response to government shocks. The intuition is that the presence of TFP shocks, which are orthogonal to government expenditure shocks, makes it risky to hold a highly leveraged position. This risk is magnified by Epstein-Zin preferences. Bhandari et al. (2017*b*) solve a similar model using a perturbation method around the current level of government debt. Our results using our methodology are consistent with their intuition.

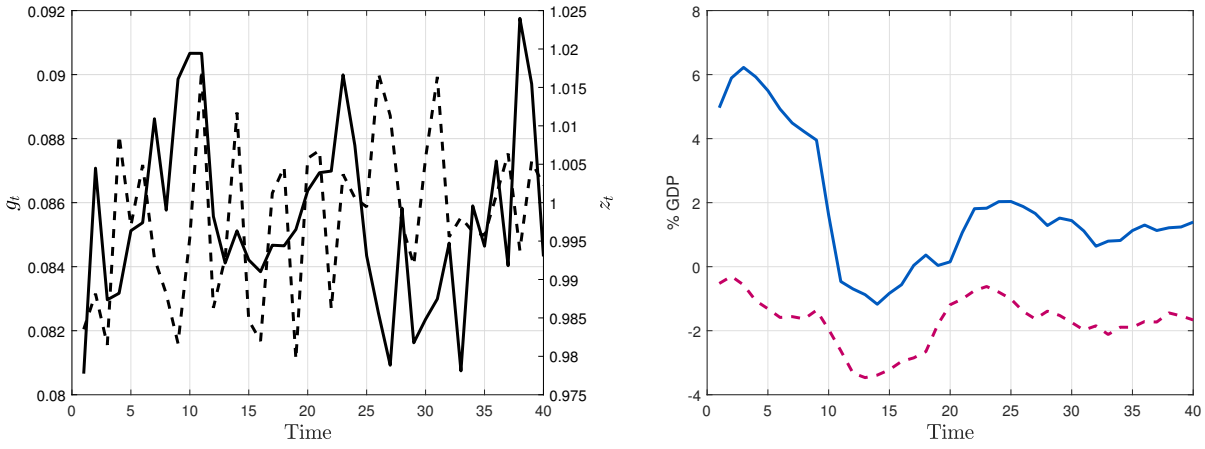

Figure 4: Simulated series with 2 bonds and Epstein-Zin preferences

Notes: The figure shows the equilibrium dynamics of the two-bond model with Epstein-Zin preferences. The left panel plots a sequence of exogenous shocks. Solid black - government expenditure, dashed black - productivity. The right panel plots the sequence of bonds. Solid blue - long bond with N=10. Dashed purple line - short bond with N=1.

[30] z_t is independent of g_t and follows an AR(1) process $z_t = \mu_z + \rho_z z_{t-1} + \epsilon_t^z$ with $\mu_z = 0.9$, $\rho_z = 0.1$, $\epsilon \sim N(0, 0.0001)$.

Parameter	Value
Discount factor	$\beta = 0.96$
RRA	$\gamma = 1.5$
1/EIS	$\rho = 1.6$
Leisure utility parameter	$\eta = 1.8$
AR(1) parameter in g_t	$\phi_1 = 0.95$
constant in AR(1) process of in g_t	$c = 0.00416$
Variance of the disturbances to g_t	$\sigma^2_\epsilon = 0.00001$
AR(1) parameter in z_t	$\phi_z = 0.1$
constant in AR(1) process of in z_t	$c = 0.9$
Variance of the disturbances to z_t	$\sigma^2_{\epsilon_z} = 0.0001$
Borrowing limits	$\bar{M}^N, \bar{M}^S = 100\%$ of GDP $\underline{M}^N, \underline{M}^S = -100\%$ of GDP

Table 7: Parameter Values used in the model with Epstein-Zin preferences

B Implementation Details

This appendix includes three subsections: (i) additional algorithm details, (ii) an in-depth discussion about the relationship with GSSA and (iii) additional figures.

B.1 Algorithm Details

In this section we describe the practical details of the algorithm used to solve the model described in section 3.3. For explanation purposes, and as an example, we use the case with three bonds. In particular, we use three maturities $S = 1$, $M = 5$, and $L = 10$.

Lagrange multipliers The system in step 2 contains multiple constraints, which poses a significant computational challenge. Ideally one would numerically solve the unconstrained model and then verify that the constraints do not bind and if, for example, M^N binds, set $b^N_{t+1} = \bar{M}^N$ and find the associated values for consumption and leisure. In a multiple-bond model this is challenging because after setting $b^N_{t+1} = \bar{M}^N$, one needs to check if the other constraints do not bind in the recomputed solution and, if they do, enforce them and recalculate the solution again and so on and on. To overcome this challenge we approximate Lagrange multipliers with the following function: $\xi^i_{L,t} = \phi(\underline{M}^i - b^i_{t+1}) + \log(1 + \phi(\bar{M}^i - b^i_{t+1}))$ if $b^i_{t+1} < \underline{M}^i$ and $\xi^i_{U,t} = \phi(b^i_{t+1} - \bar{M}^i) + \log(1 + \phi(b^i_{t+1} - \bar{M}^i))$ if $b^i_{t+1} > \bar{M}^i$, where ϕ controls the relative importance of the constraint. In our implementation we set $\phi = 90$. We also find that including these multipliers in the training set allows for different bond dynamics close and away from the constraints and improves prediction accuracy. As a result,

in our implementation

$$\mathcal{I}_t = \{g_t, \{b^S_{t-k}\}_{k=0}^{S-1}, \{b^M_{t-k}\}_{k=0}^{M-1}, \{b^L_{t-k}\}_{k=0}^{L-1}, \{\mu_{t-k}\}_{k=1}^{L}, \{\{\xi^S_{i,t-k}\}_{k=0}^{S-1}\}_{i=L,U}, \{\{\xi^M_{i,t-k}\}_{k=0}^{M-1}\}_{i=L,U}, \{\{\xi^L_{i,t-k}\}_{k=0}^{L-1}\}_{i=L,U}, \{\xi^{\text{Total}}_{i,t-1}\}_{i=L,U}\}.$$

With $S = 1$, $M = 5$, and $L = 10$ the state space includes 61 variables.

Forward-States PEA When the model contains more than one maturity, μ_t is over-identified. This is because optimality conditions for every maturity identify μ_t, as the information set I_t contains variables that are pre-determined at time t.

$$\forall i: \quad \mu_t = \mathcal{ANN}^i_1(\mathcal{I}_t)^{-1}\left[\mathcal{ANN}^i_2(\mathcal{I}_t) + \frac{\xi^i_{U,t}}{\beta^i} - \frac{\xi^i_{L,t}}{\beta^i} + \frac{\xi^{\text{Total}}_{U,t}}{\beta^i} - \frac{\xi^{\text{Total}}_{L,t}}{\beta^i}\right]. \tag{9}$$

We tackle this problem by using a Forward States PEA, introduced in Faraglia et al. (2019). It uses the current values of the state variables I_{t+1} combined with the law of iterated expectations. This is done in two steps. First, we replace the $\mathcal{ANN}^i(\mathcal{I}_t)$ terms in the optimality conditions with $\mathbb{E}_t\mathcal{ANN}^i(\mathcal{I}_{t+1})$ and, instead of approximating $\mathbb{E}_t(u_c(c_{t+i}))$, $\mathbb{E}_t(u_c(c_{t+i-1}))$ and $\mathbb{E}_t(u_c(c_{t+i}\mu_{t+1}))$, we use the information set $\mathcal{I}_{t+1}$ to approximate $\mathbb{E}_{t+1}(u_c(c_{t+i}))$, $\mathbb{E}_{t+1}(u_c(c_{t+i-1}))$ and $\mathbb{E}_{t+1}(u_c(c_{t+i}\mu_{t+1}))$ for $i = S, M, L$. Then, we use Gaussian quadrature to calculate the conditional expectation of the neural network evaluated at $\mathcal{I}_{t+1}$.

Neural Network Initialization In order to initialize the neural network weights, we need to make a guess for bond sequences. We make an educated guess that $b^L_{t+1} = g_t/2 - \mathbb{E}(g_t/2); b^S_{t+1} = -b^L_t$ and $b^M_{t+1} = \sqrt{g_t/10} - \mathbb{E}(g_t/10)$. Given these sequences, we use the government budget constraint and the first order condition for c_t to find the sequence for μ_t. Given the guess for bonds, we can calculate the initial multipliers $\xi^i_{L,t}$, $\xi^i_{U,t}$, $\xi^{\text{Total}}_{L,t}$, and $\xi^{\text{Total}}_{U,t}$. We calculate them setting the initial bond constraints equal to $\underline{M}, \bar{M} = \pm.005$. We then use these sequences to initialize the neural network. Generally, the initial guess can be any real sequences as long as it is not constant. Nevertheless, having a good guess helps the algorithm to converge faster.

Stochastic Simulation We run the stochastic simulation from the starting point where $b^S_1 = b^M_1 = b^N_1 = 0$ and $\mu_0 = \mathbb{E}(\mu)$. To solve the system of first order conditions at every period t of the simulation, we use the Levenberg-Marquardt algorithm and stop the solver when the first order condition errors are less than 10^{-12}. We set T=1000 and drop the first 150 periods in training

the neural network. The algorithm converges when the sequences for bonds and neural network weights do not change between two consecutive stochastic simulations.

Solving the System At every period t of the stochastic simulation we need to solve the system of first order conditions to get values for c_t, b^L_{t+1}, b^M_{t+1}, and b^S_{t+1}. We solve it using the Levenberg-Marquardt method. Since this is a local solver, there is no guarantee that the system is solved given a particular initial guess. In our implementation we attempt to solve the system for at most *maxrep* number of different starting points. If the solution errors are below our specified threshold, the algorithm proceeds with the solution and moves to the next period. If the solution errors are not below our specified threshold, we pick the solution with the lowest error. In practice we use 10^{-13} as the solution criteria and set *maxrep* to 50 and verify that solution errors are below it for every t in the stochastic simulation.

Neural Network Hyperparameters We set most of the hyperparameters to standard values used in the literature. We use one hidden layer to leverage the trade-off between approximation accuracy and training time. According to the universal approximation theorem, single hidden layer networks are able to approximate every continuous bounded function with an arbitrarily small error Cybenko (1988) and have smaller training times than multiple hidden layer networks. We then choose the number of neurons according to procedure described in section 2.3. We train the network using gradient descent with an adaptive learning rate. Table 8 summarizes the remaining hyperparameters.

Parameter	Value
Activation function	Hyperbolic tangent sigmoid
Training algorithm	Gradient descent with an adaptive learning rate backpropagation
Number of Hidden Layers	1
Number of Neurons	10
Learning Rate	0.0001
Learning Increase Factor	1.01
Learning Decrease Factor	0.9
Maximum Number of Epochs	1000
Performance Goal	0
Maximum Validation Checks	6
Maximum Performance Increase	1.04

Table 8: NN hyperparameters

Notes: The table summarizes the hyperparameters of the neural network used to solve the model in section 4.

B.2 Relation to Condensed PEA

Recall that S also corresponds to the number of neurons in the input layer, M is the number of neurons in the hidden layer, E is the number of expectations to approximate (which also corresponds to the number of neurons in the output layer), and T is the number of training examples.

Time Complexity of Condensed PEA In a least squares regression, the matrix multiplication ($X^T X$) dominates asymptotically the Cholesky factorization of $X^T X$.[31] Hence, the time complexity to approximate E expectations terms with OLS is $\mathcal{O}(E \cdot S^2 \cdot T)$. In the condensed PEA, this operation is repeated for an unknown number of iterations n_{CPEA}. We estimate the time complexity of the Condensed PEA as

$$\mathcal{O}(n_{\text{CPEA}} \cdot n_{\text{InternalLoop}} \cdot (\kappa(E, S, T) + E \cdot S^2 \cdot T)),$$

where $\kappa(E, S, T)$ indicates the time complexity to compute lines 3-9 and $n_{\text{InternalLoop}}$ is the unbounded number of iterations required for model convergence (while-loop in line 2).

Time Complexity of NN-based Expectations algorithm The time complexity of a single iteration of back-propagation to train a neural network that has 3 layers (with S, M, and E nodes) is $\mathcal{O}(T \cdot (S \cdot M + M \cdot E))$. Assuming there are more neurons in the hidden layer than the number of expectations to approximate ($M > E$) and n_{NN} is the number of epochs (which is in principle unbounded), we estimate the time complexity to train a neural network as $\mathcal{O}(n_{NN} \cdot T \cdot S \cdot M)$. We estimate the time complexity of the NN-based Expectations algorithm as

$$\mathcal{O}(n_{\text{InternalLoop}} \cdot (\kappa(E, S, T) + n_{NN} \cdot T \cdot S \cdot M)).$$

Given our estimates, our algorithm has a better time complexity when[32]

$$n_{\text{InternalLoop}} \cdot (\kappa(E, S, T) + n_{NN} \cdot T \cdot S \cdot M) < n_{\text{CPEA}} \cdot n_{\text{InternalLoop}} \cdot (\kappa(E, S, T) + E \cdot S^2 \cdot T).$$

After rearranging, we get a simplified expression

$$\underbrace{n_{NN} \cdot T \cdot S \cdot M}_{\text{NN training}} < (n_{\text{CPEA}} - 1) \cdot \kappa(E, S, T) + \underbrace{n_{CPEA} \cdot E \cdot S^2 \cdot T}_{\text{CPEA regression}}. \tag{10}$$

[31] We assume that the number of features (state variables) is smaller than the number of samples. That is, $T > S$.
[32] We assume that $n_{\text{InternalLoop}}$ and $\kappa(E, S, T)$ are the same in both methods.

The left hand side captures the time complexity of training the neural network. The right hand side contains the term that captures the time complexity of the regressions involved in the Condensed PEA and an additional term $(n_{\text{CPEA}} - 1) \cdot \kappa(E, S, T)$ that captures the difference between the complexity of the stochastic simulation as computed with Condensed PEA and NN-based Expectations algorithm. While $\kappa(E, S, T)$ is unknown, we can still compare the $n_{NN} \cdot T \cdot S \cdot M$ and $n_{CPEA} \cdot E \cdot S^2 \cdot T$ terms, which further reduces to comparing $n_{NN} \cdot M$ with $n_{CPEA} \cdot E \cdot S$. In our application $S = 27$, $E = 8$, $M = 10$, and $n_{NN} = 20$ on average. Given these numbers, the two algorithms have comparable complexity when $n_{CPEA} = 1$.[33] When $n_{CPEA} > 1$, the inequality in 10 clearly holds since $(n_{\text{CPEA}} - 1) \cdot \kappa(E, S, T) > 0$.

B.3 Relation to GSSA

Judd et al. (2011) propose a related method called generalized stochastic simulation algorithm (GSSA) to deliver high accuracy predictions as well as resolving the multicollinearity problem. Judd et al. (2011) resolve the multicollinearity problem using standard econometric techniques, such as single value decomposition (SVD), principal components or ridge regression. At the same time, high accuracy is achieved by approximating the policy functions and integrating them using Gauss-Hermite quadrature, instead of approximating the whole expectation terms in the optimality conditions. While the GSSA method has multiple advantages and has been successfully applied in different contexts, we find that its application to the Ramsey problem analyzed in this paper poses challenges. Judd et al. (2011) propose a related method called generalized stochastic simulation algorithm (GSSA) to deliver high accuracy predictions as well as resolving the multicollinearity problem. Judd et al. (2011) resolve the multicollinearity problem using standard econometric techniques, such as single value decomposition (SVD), principal components or ridge regression. At the same time, high accuracy is achieved by approximating the policy functions and integrating them using Gauss-Hermite quadrature, instead of approximating the whole expectation terms in the optimality conditions. While the GSSA method has multiple advantages and has been successfully applied in different contexts, we find that its application to the Ramsey problem analyzed in this paper poses challenges.

First, in this application it is particularly challenging to approximate the policy functions directly, because the expectations are over N periods ahead. In the context of the model presented

[33] Note that when $n_{CPEA} = 1$, the Condensed PEA is essentially a standard PEA algorithm. It is possible that the Condensed PEA converges in one iteration but that requires a good guess of the initial set of core state variables, which needs to be found through trial and error.

in section 2 of this paper, the evaluation of $\mathbb{E}[u'(c_{t+1})]$ requires knowing K_{t+2}, which is a function of K_{t+1}, which itself is a function of K_t and z_t, $K_{t+1} = \phi(K_t, z_t)$. That is, $c_{t+1} = z_{t+1}\phi(K_t, z_t)^{\alpha} + (1-\delta)\phi(K_t, z_t) - \phi(\phi(K_t, z_t), z_{t+1})$. When the expectation is N periods ahead, the evaluation of $\mathbb{E}[u'(c_{t+N})]$ using the GSSA approach would require iterating on the approximated policy functions and the budget constraint N times to have the value for K_{t+N+1}, which would result in an imprecise evaluation of the expectations and lower stability of the algorithm compared to an application where the forecast is one period ahead.

Second, methods such as ridge regression impose a penalty on the size of the coefficients, providing stability but causing them to be downward biased. The choice of this penalty parameter is the source of instability. It is particularly hard to choose the penalty parameter in the context of solving the model, since the regression is performed on simulated data which are not fixed and not exogenous from the choice of the penalty. Simulated data depend on the coefficients and penalties obtained in the previous iteration of the algorithm. Typically the choice of penalty parameter requires adding an additional loop and solving the model with multiple values. However, it is not obvious which penalty parameter is optimal because the optimal penalty is different at every step of the PEA iteration as the Maliar bounds become increasingly open.

In order to compare our method with GSSA and other standard econometric techniques, in this section we solve the one-bond model from section 3, with short-term debt $N = 1$. We solve it with standard PEA, except that we use ridge regression. Hence, we do not use Montecarlo integration as in GSSA because that would require to integrate N-periods ahead. Instead, standard PEA does not require to perform any integration since it approximates the expectation terms directly.[34] In order to pick the penalty parameter we use a cross-validation approach, where we change the penalty dynamically at each step of the fixed point iteration in the regression stage. We select the penalty parameter that minimizes the mean squared prediction error between predicted and simulated sequences.[35] Equation 11 illustrates the penalty selection procedure.

$$\begin{aligned} &\min_{\kappa} ||Y - X\hat{\boldsymbol{\beta}}||_2^2 \quad \text{s. t.} \\ &\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} ||Y - X\boldsymbol{\beta}||_2^2 + \kappa||\boldsymbol{\beta}||_2^2 \end{aligned} \tag{11}$$

[34] Note that when we use more than one maturity, we use Forward-States PEA and perform one step of integration using Gaussian quadrature as explained in section 3.3.

[35] Similarly, Judd et al. (2011) choose the smallest penalty that ensures numerical stability of the fixed point iteration, which also provides a high accuracy solution.

We use ridge regression as opposed to SVD or a principal component analysis for two reasons. First, ridge regression is supposed to work better than SVD when the multicollinearity is severe, as is the case in the model of section 4. Second, we do not use principal component analysis, since the Condensed PEA effectively does the same extraction of orthogonal components, just iteratively.

From our numerical experiments, we discover that PEA combined with ridge regression converges only under specific conditions. Note that throughout the paper we have been using debt limits. This effectively introduces an occasionally binding constraint, which makes the multicollinearity problem even more severe if the debt sequence visits the constrained region frequently. Besides, the algorithm requires using Maliar bounds, which potentially cause even more instability since the borrowing constraint changes as the bounds progressively open. We find this to be crucially important. For illustration we consider the one-bond model with tight ($\bar{M}, \underline{M}$ at $\pm$100% of GDP) and loose ($\bar{M}, \underline{M}$ at $\pm$200%) borrowing constraints. At every iteration we compute the maximum difference between the ridge regression coefficients at the current and the previous iteration. When the borrowing constraint is loose, the regression coefficients stabilize after the Maliar bounds are wide enough and the borrowing constraint stops binding. In contrast, in the specification with tight constraints, the constraint binds more often, requiring the use of a large penalty parameter, which prevents the algorithm from converging.

	Tight Constraint	Loose Constraint
$\mathbb{E}(\Delta\beta)$	3.81	.08
% constraint binds	38.2%	0

Table 9: Solution using ridge regression

Notes: Table shows selected statistics from model specifications with tight and loose borrowing constraints when solving the model using PEA with Ridge regression. Tight refers to the case when $\bar{M}, \underline{M}$ at $\pm$100% and loose refers to the case when $\bar{M}, \underline{M}$ at $\pm$200%. First row shows the average change in the Ridge coefficients in the last five iterations. Second row shows the percentage of time the bond hits the constraint. The specification with loose constraints converges in 198 iterations. The specification with tight constraints never converges, therefore, we stop the code at iteration 198.

Table 9 illustrates this point. The first row shows the average change in ridge coefficients across consecutive PEA iterations for the last 50 iterations. The second row shows the percentage of time that debt visits the constraint in the last iteration. In the specification with tight borrowing constraint, the bond stays around 43% of the time close to the constraint and ridge coefficients never stabilize. Alternatively, this can be seen in figure 5, which plots the total model prediction error across the PEA iterations as the Maliar bound is being opened. In the specification with loose

constraints, the forecast errors begin to stabilize when the Maliar bound stops changing and the algorithm slowly converges. In contrast, when constraints are tight, the ridge penalty parameter keeps changing and the forecast errors never stabilize and remain large (see table 10).

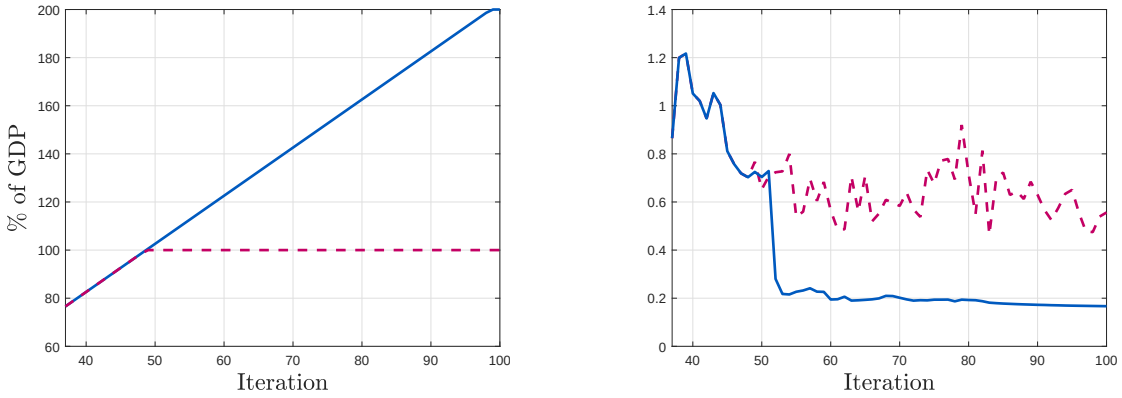

Figure 5: Convergence of model forecast errors using ridge regression

Notes: The figure shows the convergence of the model using ridge regression. Left panel shows the value of Maliar bound in function of the algorithm iteration. Right panel shows the total model forecast error in function of the algorithm iteration. Solid blue line - loose borrowing constraint. Dashed purple line - tight borrowing constraint. Tight constraint refers to the case when $\bar{M}, \underline{M}$ at $\pm 100\%$ and loose constraint refers to the case when $\bar{M}, \underline{M}$ at $\pm 200\%$.

	$\mathbb{E}_t(u_{c,t+N}\mu_{t+1})$	$\mathbb{E}_t(u_{c,t+N})$	$\mathbb{E}_t(u_{c,t+N-1})$
Tight Constraint	0.1634	0.2786	0.2664
Loose Constraint	0.0117	0.0679	0.0668

Table 10: Prediction accuracy using ridge regression

Notes: The table shows the average absolute forecast errors from model specifications tight and loose borrowing constraint when solving them using PEA with Ridge regression. Tight refers to the case when $\bar{M}, \underline{M}$ at $\pm 100\%$ and loose refers to the case when $\bar{M}, \underline{M}$ at $\pm 200\%$. First row shows the average change in the Ridge coefficients in the last five iterations. The specification with loose constraints converges in 198 iterations. The specification with tight constraints never converges, therefore, we stop the code at iteration 198. We define the average absolute forecast error as $\frac{1}{T}\sum \left|Y_{t+N} - \hat{\mathbb{E}}_t(Y_{t+N})\right|$.

We have also attempted to solve the two-bond model using ridge regression but in this case, the problem of instability becomes even more severe as the long and the short bonds are negatively correlated and highly volatile. As a result, bonds hit the constraint very frequently and the algorithm fails to converge even before the Maliar bounds are open.[36]

[36] In the one bond code ridge penalty is a scalar. When solving the 2 bonds model we allow coefficients to have different penalties, without any noticeable improvement.

B.4 Additional Figures

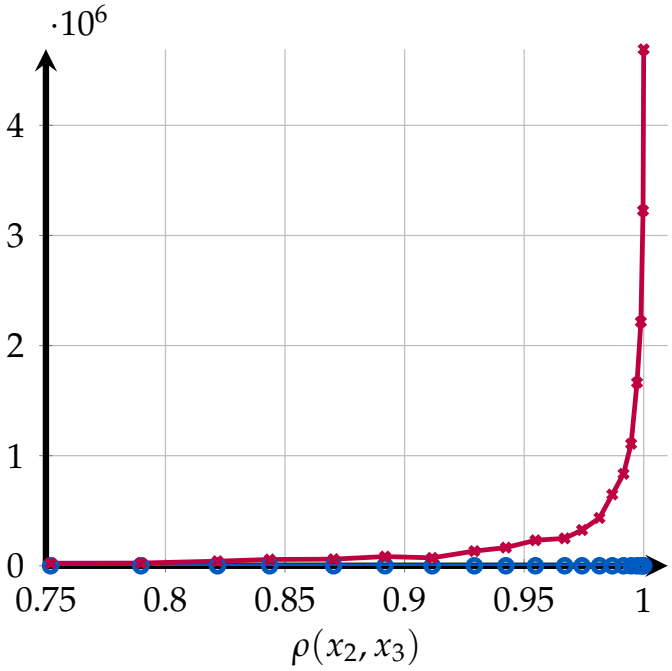

Figure 6: Bias squared of the mean squared prediction error with a neural network and a polynomial

Notes: The figure shows the bias squared of the mean squared prediction error $1/n \sum_{t=1}^{n} [y_t - \mathbb{E}(\hat{y}_t)]^2$ in function of the correlation between x_2 and x_3. Blue line with circles - NN, purple line with crosses - polynomial regression.

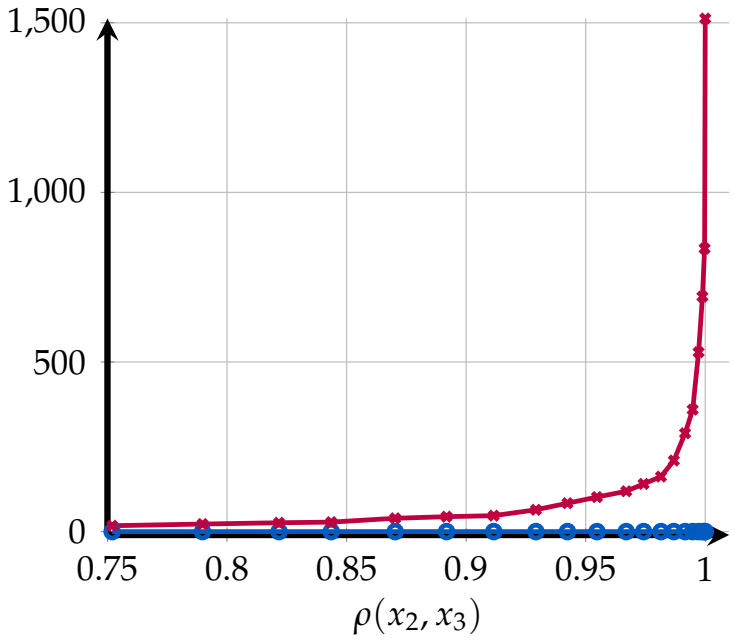

Figure 7: Variance term of the mean squared prediction error with a neural network and a polynomial

Notes: The figure shows the variance component of the mean squared prediction error $1/n \sum_{t=1}^{n} [y_t - \mathbb{E}(\hat{y}_t)]^2$ in function of the correlation between x_2 and x_3. Blue line with circles - NN, purple line with crosses - polynomial regression.

C Neural Network Details

This appendix includes further details about the type of neural network used in section 2.

C.1 Artificial Neural Networks

Neural networks can be used efficiently for both regression and classification purposes (in a supervised machine learning fashion) and clustering (in an unsupervised machine learning fashion). In the context of our application, we focus on the former. Neural networks are typically composed of three types of layers: (i) input, (ii) hidden, and (iii) output. They can contain multiple hidden layers but, for regression purposes, typically one or two hidden layers are sufficient.[37] In our application, the input layer takes as input the state space $X_t \in \mathbb{R}^S$. The hidden layer performs an intermediate transformation of the state space. The output layer predicts the expectation terms contained in the model optimality conditions $\mathbb{E}\left[g_e(c_{t+1}, X_{t+1})|X_t\right] \simeq \mathcal{F}_e(X_t; w, \beta)$. See figure 8 for a graphical illustration.

[37] In our application, we consider a neural network with one hidden layer. The reader can refer to Goodfellow et al. (2016) for a general introduction to machine learning and deep learning in particular.

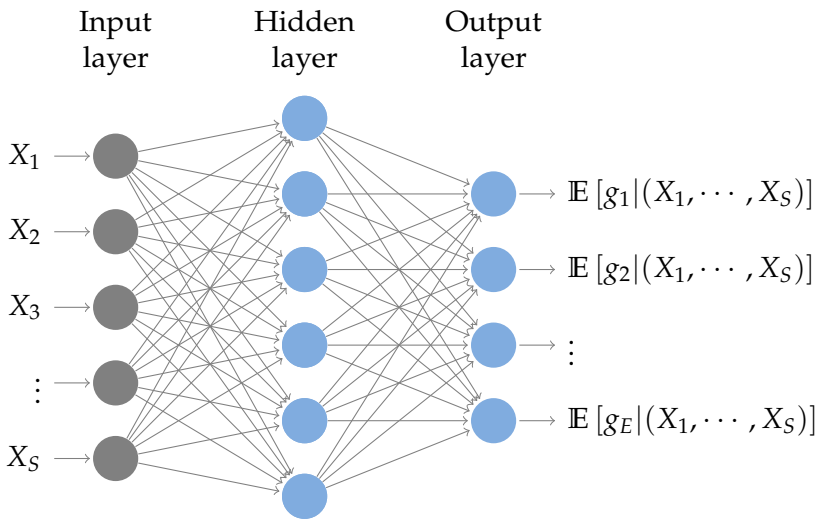

Figure 8: Artificial Neural Network Structure

Notes: The figure presents the structure of a single hidden layer artificial neural network. Each circle in the picture represents an artificial neuron, and the arrows point in the direction of the information flow in the prediction process. Neurons in the hidden layer perform the non-linear activation of inputs, which are combined linearly in the output layer.

If the problem requires the approximation of E expectations terms, a neural network with one hidden layer has the following functional form

$$\tilde{X}_m = \mathcal{H}\left(\sum_{s=0}^{S} w_{m,s} \cdot X_{s,t}\right), \quad m = 1, \cdots, M,$$
$$\mathcal{F}_e(X_t; w, \psi) = \psi_{0,e} + \sum_{m=1}^{M} \psi_{m,e} \cdot \tilde{X}_m, \quad e = 1, \cdots, E.$$

Note that the *universal approximation theorem* (see Cybenko, 1988 and Hornik et al., 1989) ensures that every bounded continuous function can be approximated with arbitrarily small error, by a network with one hidden layer. The hidden layer transforms the state space $X_t \in \mathbb{R}^S$ through M linear combinations of the state variables, further transformed through an activation function $\mathcal{H}(x)$. The activation function is typically a sigmoid

$$\mathcal{H}(x) = \frac{1}{1 + \exp(-x)}.$$

In order to gain intuition about the behavior of this function, consider a more generic function $\tilde{\mathcal{H}}(x; \alpha) = \frac{1}{1+\exp(-\alpha \cdot x)}$, where α is a parameter that regulates the activation rate. Intuitively, the

larger is α, the more $\tilde{\mathcal{H}}(x;\alpha)$ resembles to a step function as shown in figure 9.[38] Note that neural networks are equivalent to linear regression if the activation function $\tilde{\mathcal{H}}$ is linear.[39]

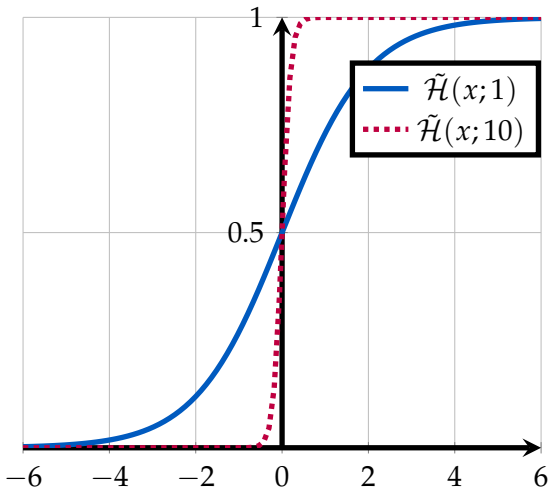

Figure 9: Sigmoid Function

Notes: The plot of the function $\tilde{\mathcal{H}}(x;\alpha)$. Solid blue line: Plot of the sigmoid function (when $\alpha = 1$), typically used in the hidden layer of a neural network. Dashed purple line: Plot of the $\tilde{\mathcal{H}}(x;\alpha)$, when $\alpha = 10$. The higher is α the more the $\tilde{\mathcal{H}}$ function acquires the shape of a step function.

[38] In the context of deep learning (in contrast to shallow neural networks), the de-facto standard is rather the rectifier (or ReLU) or swish activation functions.

[39] Setting $\tilde{\mathcal{H}}$ to a linear function would be equivalent to approximating the expectations with linear polynomials in the traditional PEA algorithm.