Solve the Aiyagari (1994) model for

1. Stationary distributions: Huggett (1993); Aiyagari (1994); Castañeda et al. (2003).
2. Steady-state to steady-state transitions
3. Aggregate shocks: Krusell and Smith (1998); Den Haan and Rendahl (2010)

We will quickly review 1. and 2., which you should have already dealt with in Sumudu's class.

Recommendations:

- A good general textbook for this is Heer and Maussner (2009)
- A good paper summary is Ríos-Rull (1999)
- Once you know how to solve these models other models (OLG models, human capital models, endogenous incomplete markets, labor-leisure models) are all direct extensions, you can look up and read papers with such applications on your own
- I wouldn't recommend solving a full-blown model just for practice unless you want to write a paper with similar methods yourself

For illustration purposes we remain as simple as we can. Agents have the value

$$V(a,e) = \max_{c,a'} \left\{ u(c) + \beta \mathbb{E}\left[V(a',e')|e\right] \right\} \tag{1a}$$

$$\text{s.t.} \quad c + a' = we + (1+r)a \tag{1b}$$

where $e \in \{0,1\}$ follows a simple, $n \times n$ transition matrix:

$$\Pi = \left[\pi_{ij}\right], \quad i,j = 1,\ldots,n, \quad \sum_i \pi_{ij} = 1.$$

In this simple model the natural borrowing constraint is simply 0. The aggregate technology is standard neoclassical:

$$F(K,N) = AK^{\alpha}N^{1-\alpha}.$$

In most models that focus on households rather than firms, you will see this a lot! The only difference, perhaps, will be how they define aggregate labor N, but that should be straightforward from the model.

1. Stationary Distribution

For now we assume that A is constant, so there is no aggregate uncertainty. In a stationary equilibrium, the market clearing conditions for capital and labor are simply

$$K = \int a dF(a,e) = K' = \int a^*(a,e) dF(a,e),$$
$$N = \int e dF(a,e) = \int e G(e),$$

where F is the stationary distribution you will have to find, and G is the invariant distribution of Π. Two things to note right away:

1. Since we seek a stationary distribution, $K = K'$.

2. Since Π is exogenous, we don't need F to obtain N.

DEFINITION 1 *A **stationary equilibrium** is defined as*

1. *Agents and firms optimize,*
2. *Markets clear—note the capital and labor market clearing conditions are sufficient by Walras' Law.*
3. *The distribution of individual states is constant, i.e.*

$$F = \Phi(F)$$

where Φ is the aggregate law of motion induced by a^* and Π.

We can solve the model numerically as follows:

EXERCISE 1 *Solve stationary distribution for endogenous $R = r + \delta$:*

1. *Guess an equilibrium R. Note that w is then fixed by the representative firm's f.o.c.:*

$$w = A(1-\alpha)\left(\frac{\alpha}{R}\right)^{\frac{\alpha}{1-\alpha}}.$$

2. *Solve the individual value/policy functions for all states, given $r_S = R - \delta$*
3. *Compute the stationary distribution*
4. *Compute aggregate supply of capital (AS) by integrating over the stationary distribution*
5. *Plug K_S in the AD curve to obtain r_D*
6. *Terminate if $r_D \approx r_S$. Otherwise update r_S and repeat from 1.*

Intuitively (but proving this is harder), we can guess that the AS curve is bounded above by $1/\beta - 1$, and crosses 0 at a positive level of K_S. So basically all we need to know is how to solve for the policy function, and how to obtain the stationary distribution F. Before that, a comment on steps 1 and 2.

Modeling AD Huggett (1993) analyzed an endowment economy while Aiyagari (1994) analyzed a production economy. If you believe what I said about the AS-curve, it should be straightforward why this procedure should work—in a Huggett economy, $K_D = \bar{K}$ is fixed vertically, and in an Aiyagari economy the AD curve is simply the representative firm's f.o.c.

$$r_D = A\alpha\left(\frac{N}{K}\right)^{1-\alpha} - \delta.$$

In other applications, r is fixed, implying either a small open economy or a short-run partial equilibrium. In any case, the existence of a market clearing r (or when r is fixed, market clearing K_D) is guaranteed.

Equilibrium dynamics or calibration? In many applications, we want to match a specific target interest rate, e.g. $r = 4\%$. If that is the case, we have to keep computing the equilibrium until the r we computed somehow ends up being 4%. Instead of doing that, it is faster to fix $r = 4\%$, and iterate on a parameter that you know controls individual behavior in a similar way as the interest rate, such as β or risk aversion (or whatever your model has).

EXERCISE 2 *Solve stationary distribution for $r = 4\%$:*

1. *Fix $r = 4\%$, and solve for w and K_D as well. The firm side now remains constant throughout the problem.*

2. *Guess a β (or some other parameter).*
3. *Solve the individual value functions for all states*
4. *Compute the stationary distribution*
5. *Compute aggregate supply of capital (AS) by integrating over the stationary distribution (K_S)*
6. *Terminate if $K_S \approx K_D$. Otherwise update β and repeat from 2.*

Why this would save time should be straightforward.

2. Solution in Practice

Now we turn to solving the model in practice. In Fortran, the backbone of the entire code will be in the main ***PROGRAM*** file. In C/C++, this is the int or void main(). This file will just keep track of things, irrelevantly of what the particular model you are trying to solve. For example, this is where you will invoke an optimization routine over a ***FUNCTION*** or ***SUBROUTINE*** that solves the entire model, taking parameter guesses as inputs and a distance estimate (or whatever estimation/calibration criterion) as an output.

2.1 Set parameters

The first thing to decide is whether to fix a grid over assets *a* outside the model or not; in this case it is better to fix it. If you let the grids vary with different parameter guesses, the solution may never converge. However, in other cases, you may be able to allow the grid to vary systematically given a guess—e.g., you may not care about savings levels above 50 times the level of average earnings (but in this case, you would have to also normalize average earnings in the model equilibrium. We'll talk more about this later).

Then fix or guess a vector for the model's parameters. Later, these would be calibrated or estimated by setting a big outer-loop over the vector of parameter guesses.

Coding practicalities: For parameters of the model, including functional forms for say, the utility function, make a separate file/module that includes all of them so that they can be easily invoked from any routine. For example, make ***USE*** of ***MODULE*** in Fortran.

Modules and I/O Files

1. You want to know how many model parameters are going to be calibrated in advance, so you can make a parameter vector that will be input into an optimization routine. This will also make it easier to run robustness/counterfactual exercises with different parameter values.
2. Another trick is to save parameter values in a separate text file so that you don't need to recompile every time you change the parameters.
3. You also need to input parameter targets. If targets are few, you can declare them as parameters from the beginning. If it comes from manipulating a large dataset, these can also come from a separate text file.
4. In any language but particularly in Fortran, exploit modules (or similar constructs in other languages) as much as you can. Think in advance, in terms of a flow chart or whatever other logic, how you want to build the code. My practice is to just write a bunch of empty modules from the very beginning, then build up step-by-step.
5. By default, all variables declared in a module are global. To prevent this, make variables you want to remain local to the module as ***PRIVATE*** (as opposed to ***PUBLIC***).

The advantage of modules is that they are declared ***EXTERNAL*** by default with respect to the main ***PROGRAM*** file.

2.2 Exogenous process

Create a grid for e's; in this case it is $(0, 1)$. For an AR(1) process, we can use either ***Tauchen (1986)*** or better, the ***Rouwenhorst method*** in Kopecky and Suen (2010). In any case, we assume that the e process is already stationary. For other generic processes, we also fix the process here, before solving any of the optimization problems.

* **IF** you later plan to use Monte Carlo simulation to solve for the stationary distribution, simulate the earnings process before solving the rest of the problem.

1. Aiyagari (1994) simulated one individual for $\alpha + T$ periods for large T, and assumed that the distribution of T is the stationary distribution. This works because of ergodicity of an AR(1) process.

2. But no one does this any more. Rather, simulate N individuals for T periods, for large N and smaller T (a rule of thumb is 200-300 periods), as in Ríos-Rull (1999). The distribution over N at time T is taken as the stationary distribution. (N, T) should be declared as parameters.

Again, both the grids, and the fixed stationary process and Monte Carlo simulation can be saved in their own modules. Make sure that any cumulative variables are initialized.

If you plan to directly compute the distribution, ignore this step.

2.3 Solve the savings problem

Guess an interest rate r, or β, in a routine that will solve the model, given a guess for model parameters. In models with more markets, you will have to guess more prices. Then given the guess of a price vector, solve for the value/policy functions.

Equilibrium Code Construct a routine that takes the prices as inputs, and admits some distance metric (distance between K_S and K_D, for example). Then in the model code, iterate over the prices (or β, etc.) to solve for the equilibrium. When computing the steady state of the incomplete markets model, this would contain

1. a code or module that solves for the value/policy functions

2. a code or module that solves for the stationary distribution.

In the standard incomplete markets model, all we really care about is the policy function, of course. Again, there are several ways to solve this. In any case, the policy function has to be found on every possible point on the (a, e) grid that you have fixed above:

1. Discrete state space optimization: find next periods best a' that lies on the a-grid. This is easy, but not exact and also not very fast. You will also need a large number of grid points. If you insist, use fast-sorting algorithms like ***QUICKSORT*** or ***HEAPSORT***.

2. Value function iteration: just solve (1) directly. Given a guess of V^n, use any optimization routine to solve the RHS of (1). Save the resulting solution as V_{n+1}. Iterate until $V_{n+1} \approx V_n$

   (a) The fastest method is usually ***Newton-Raphson***. In more complex models, you may have more than one choice variable; as long as the value function is smooth, you can use a ***conjugate gradient method*** (basically a multidimensional Newton method) or a ***quasi-Newton/variable-metric method (e.g. BFGS)***.

(b) There are also derivative-free methods (***simplex, NEWUOA*** etc.), but since the value function is in the most inner-loop of the problem, getting an exact solution is important and such methods are less exact—except when there is only one choice variable. In this case, I recommend ***Brent's method***.

It remains what the initial guess, V_0, should be. You may have learned to set it to zero, but that wastes a step, since we know in that case the optimal policy would be zero (since tomorrow doesn't matter):

$$\begin{aligned} V_1(a,e) &= \max_{c,a'} \{u(c) + \beta \mathbb{E}[0|e]\} \quad \text{s.t.} \quad c + a' = we + (1+r)a \\ &= u(we + (1+r)a) \end{aligned}$$

So you might as well set $V_0(a,e) = u(we + (1+r)a)$ to begin with.

3. Policy function iteration: instead of solving the value function directly, iterate on the policy function. In this simple case, we can take advantage of the Euler equation. Given a guess of a_n, find a' that solves, for each (a,e):

$$u'(we + (1+r)a - a') = \beta(1+r)\mathbb{E}\left[u'\left(we' + (1+r)a' - a_n(a',e')\right)|e\right]$$

and set $a_{n+1}(a,e) = a'$ for each (a,e).

By the envelope theorem, in practice it can be easier to iterate on the derivative of the value function:

$$V'_{n+1}(a,e) = (1+r) \cdot u'(we + (1+r)a - a') = (1+r) \cdot \beta \mathbb{E}\left[V'_n(a',e')|e\right],$$

where V' is the derivative of V w.r.t. the first argument a.

(a) Whether you are optimizing the value function or solving an equation like here, the same rule of thumb applies: Newton will usually be the fastest. Especially if you have more than one choice variable, and hence multiple Euler equations.

(b) In the single choice variable case, again you can use Brent's equation solver (as opposed to optimizer).

Again, what about the initial guess? Whether you guess the policy function or derivative of the value function, you might as well guess 0, since this would be equivalent to my proposed guess for V_0.

However, if you in fact do have more than one choice variable, it may not be wise to try to solve a system of Euler equations: non-linear equation solvers that use Newton or secant methods (e.g. ***Broyden***) are prone to error with more than one equation. Many practitioners optimize over several variables if they have to, while setting a nested loop to solve for the Euler equation only for the savings choice (which is the easiest to characterize).

How would you deal with a corner solution? This is when using Brent's method to solve for an Euler equation comes in handy, since it always first checks the two boundaries. If the inequality holds in the wrong direction at the borrowing constraint, that must be the solution.

For 2. and 3., you will also need to interpolate over the a-grid, since a' is not forced to lie on a grid: ***linear interpolation*** is always the most robust, but ***cubic splines*** and ***Hermite/Chebyshev polynomials*** are arguably more popular. Sometimes a ***shape-preserving spline*** is used, but this is known to be unstable. All such interpolation schemes are easily available as packages in most languages.

In all cases, there are small tricks to be more exact and speed things up:

1. Since the curvature of the policy function is more extreme at lower states and close to linear for larger states, you want to have more grid points at the bottom than at the top to be as close as possible to the true solution

2. Since we know the policy function is monotone in (a, e), once we know $a^*(a_i, e_j)$, you need not search below this value when searching for the solution at all states $(a_i, e_{j'})$ s.t. $(i' \geq i, j' \geq j)$ (***Gauss-Seidel***)

2.4 Solving for the Stationary Distribution

So now we have $a^*(a, e)$, the optimal policy function, for every point on our fixed grids (a, e). If we don't care about an equilibrium, we're done (this is what a partial equilibrium model could stop; most micro-papers do this). But we want to get the equilibrium stationary distribution.

A note of caution: the equilibrium and stationary distribution are separate objects. Neither implies the other. A lot of people seem to confuse this. Equilibrium in our context means market clearing in every possible time and for any distribution. Stationary distribution is just a distribution that does not change over time. With ergodicity, a stationary distribution exists uniquely.

Coding Revisited So a model routine should take prices (or β) as given, and first call the value/policy function routine, get the policy functions, and second call the stationary distribution routine. Then the model routine uses a wrapper to solve for the equilibrium price. This only works because we *impose* stationarity from the start.

2.4.1 Monte Carlo Methods

Aiyagari's Monte Carlo Some earlier papers used correct but less accurate methods. It is not recommended to use these anymore, since it was mainly used due to limited computation power. But it is still useful to know for its intuition.

1. Start from any initial condition for one individual
2. Simulate they behavior of this individual for T periods, say 105,000.
3. Aggregate over this one guy's policy function from $t = 5001$ to T. This should approximate the stationary distribution, call it $\tilde{F}$.
4. If you want to check whether this works, simulate the same guy for T more periods. Check if the to distributions ($\tilde{F}_1$ and $\tilde{F}_2$) are the same. If not, take a longer T for your stationary distribution.

That's it. Basically, all it's doing is taking advantage of ergodic properties of a Markov transition. As mentioned earlier, if implementing this method you must fix the exogenously random process for *e before* solving for the equilibrium. Otherwise you may not find an equilibrium (because the exact distribution of e will vary ever so slightly at each iteration).

But as said above, this method is not recommended.

Regular Monte Carlo A more direct approach, but the same idea, is to just do a straightforward Monte Carlo:

1. Start from any distribution of initial conditions over N individuals, say 100,000.
2. Simulate the behavior of *all* these guys until $\tilde{F}_t \approx \tilde{F}_{t+1}$.
3. In practice, let them run for T periods, say 200. Check if the distributions are identical from $t = 199$ to $t = 200$. If not, increase T.

Compared to approximating the density function, this is cumbersome but easier to code. Depending on the model, it can take both more or less time to run than other methods. Again, if implementing this method you must fix the exogenously random process for *e before* solving for the equilibrium, now for N individuals though. In the code, this means this should be *outside* the equilibrium routine, and either in the program or model routine.

2.4.2 Approximating the density function

The most accurate method is to directly solve for the density function by approximation. There are many ways to do this, I summarize three.

Inverting the policy function If you have a continuous solution to the savings policy function (either the analytical solution, or some method using a smooth interpolation scheme), we could do the following.

1. Set a 2-dimensional c.m.f. over (a, e), where the points for a can potentially take on an infinite number of values. The abscissas of this c.m.f. does *not* have to be identical to the grid you used to solve the value functions (but can be).
2. Guess $\tilde{F}_0$ over (a, e), for all we care it could be uniform.
3. For each $\tilde{F}_t$, obtain $\tilde{F}_{t+1}$ as follows. For every point (a', e') on your abscissas (which can be continuous, if you assumed $\tilde{F}$ to be a function), solve

$$\tilde{F}_{t+1}(a', e') = \sum_e \pi(e'|e) \tilde{F}_t \left(a^{*-1}(a', e,), e \right),$$

   where $a^{*-1}\left(a^*(a, e), e\right) = a$.
4. Terminate if $\tilde{F}_t \approx \tilde{F}_{t+1}$.
5. In practice, as in the Monte Carlo case, simulate for say T periods and check the above condition once. If it doesn't hold, increase T.

While this is theoretically the most sound way to get the distribution function, in practice it will be difficult to directly obtain a^{*-1}, or computationally too costly to iterate on the distribution function as above.

Forward iterating on a p.m.f. Usually instead of fixing (a', e) and finding the implied a, we search over all (a, e)—which again, does not have to be the same as the grid you used to solve for the value function, but can be—to determine the next period density for a'. The simplest way is to approximate a p.m.f. using linear interpolation—i.e., whenever $a^*(a, e)$ is off a grid, assign probabilities to the neighboring bins according to its interpolation weights.

1. Set a 2-dimensional **p.m.f.** over (a, e). Let $\mathcal{A}$ denote the grid for a:

$$\left[a_1, a_2, \cdots, a_{j-1}, a_j, \cdots, a_M\right].$$

   Recall that in this model, $a_1 = 0$, and that we know with certainty that there exists an $a_M < \infty$ in equilibrium.
2. Guess $\tilde{f}_0$ over (a, e), for all we care it could be uniform.
3. For all $\tilde{f}_t(a, e)$, obtain $\tilde{f}_{t+1}$ by

$$\begin{aligned}\tilde{f}_{t+1}(a_j, e') = &\sum_{e \in \mathcal{E}} \sum_{a \in \{\mathcal{A} | a'(a,e) \in [a_{j-1}, a_j)\}} \pi(e'|e) \cdot \left[\frac{a^*(a, e) - a_{j-1}}{a_j - a_{j-1}} \right] \cdot \tilde{f}_t(a, e) \\ &+ \sum_{e \in \mathcal{E}} \sum_{a \in \{\mathcal{A} | a'(a,e) \in [a_j, a_{j+1})\}} \pi(e'|e) \cdot \left[\frac{a_{j+1} - a^*(a, e)}{a_{j+1} - a_j} \right] \cdot \tilde{f}_t(a, e)\end{aligned}$$

   Of course, this gives a new p.m.f. over $\mathcal{A}$. You could assign probabilities differently using a different interpolation scheme.

4. Terminate if $\tilde{f}_t \approx \tilde{f}_{t+1}$.

5. In practice, as in the Monte Carlo case, simulate for say T periods and check the above condition once. If it doesn't hold, increase T.

This will usually be the easiest and fail-safe way to approximate the density function.

Eigenvalue method Note that all the above algorithms rely on the Markov theorem, that the limiting distribution of any ergodic Markov chain converges to the stationary distribution. Instead we can solve for the exact stationary distribution without relying on limiting assumptions. In the following we again assume linear interpolation.

1. As above, set the a-grid $\mathcal{A}$ and e-grid $\mathcal{E}$. This means the state space is $\mathcal{A} \times \mathcal{E}$. For example, if $\mathcal{A}$ has M elements and $\mathcal{E}$ has N, the size of the state space is $M \times N$. In order to solve the transition matrix, we "reshape" this matrix into a vector

   $$\mathcal{S}' = [s_1, \ldots, s_M, s_{M+1}, \ldots, s_{MN}]$$

   such that

   $$\begin{aligned} s_1 &= (a_1, e_1), \quad \ldots, \quad s_M &&= (a_M, e_1) \\ s_{M+1} &= (a_1, e_2), \quad \ldots, \quad s_{MN} &&= (a_M, e_N) \end{aligned}$$

   where $(e_1, e_2) = (0, 1)$.

2. Solve for the transition matrix Φ, which is $MN \times MN$ and each column will add up to 1, i.e.

   $$\Phi = \begin{bmatrix} \phi_{1,1} & \cdots & \phi_{M,1} & \phi_{M+1,1} & \cdots & \phi_{MN,1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_{1,M} & \cdots & \phi_{M,M} & \phi_{M+1,M} & \cdots & \phi_{MN,M} \\ \phi_{1,M+1} & \cdots & \phi_{M,M+1} & \phi_{M+1,M+1} & \cdots & \phi_{MN,M+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_{1,MN} & \cdots & \phi_{M,MN} & \phi_{M+1,MN} & \cdots & \phi_{MN,MN} \end{bmatrix},$$
   $$\phi_{i,j} \geq 0 \qquad \forall i, j, \qquad \sum_j \phi_{i,j} = 1 \qquad \forall i,$$

   where $\phi_{i,j}$ is the probability of transitioning from state s_i to s_j. For each (i, j),

   $$\begin{aligned} \phi_{i,j} &= \pi_{k,l} \cdot \frac{a_{n+1} - a^*(a_m, e_k)}{a_{n+1} - a_n} \\ \phi_{i,j+1} &= \pi_{k,l} \cdot \frac{a^*(a_m, e_k) - a_n}{a_{n+1} - a_n} \end{aligned}$$

   where $a^*(a_m, e_k) \in [a_n, a_{n+1})$ and

   $$\begin{aligned} m &= i - M(k-1), \quad k = \quad \text{mod } (i, M) + 1, \\ n &= j - M(l-1), \quad l = \quad \text{mod } (j, M) + 1 \end{aligned}$$

   The expressions for (m, n) and (k, l) look weird, but just looking for the corresponding indices of (a, e) on the grid $\mathcal{A} \times \mathcal{E}$ given $\mathcal{S}$. Coding is very easy.

3. Obtain the eigenvalues and eigenvectors of Φ, i.e. the Λ and X, both $MN \times MN$ s.t.

   $$\Phi X = \Lambda X.$$

   This is easily done with pre-existing numerical routines. Clearly the eigenvector associated with $\lambda = 1$ is the invariant distribution.

Note that there can be cases where there is no $\lambda = 1$ in the real domain. Most likely this will mean that the particular set of parameters does not admit a stationary distribution. Unlikely but possibly, it could simply be due to numerical approximation errors. While this method is recommended, it is infeasible for large state spaces where the Markov transition matrix is too large (which will make numerical computation of the eigenvector extremely costly or error-prone).

2.5 Obtain Equilibrium Variables

In the Aiyagari (1994) model, all you need to get is the equilibrium interest rate r, or the discount factor β. Although the nested problem is large (i.e., solving the value function then finding the stationary distribution), mathematically speaking it is still one equation, one unknown (either $r_S = r_D$ in Exercise 1 or $K_S = K_D$ in Exercise 2). So although Aiyagari (1994) used a ***bisection*** method to find r^*, it would be faster to use any equation solver in one dimension, like Newton or Brent.

However, for more complicated problems, there is usually more than one equilibrium variable. Since these equilibrium loops are costly, you always want to avoid nested loops whenever possible; but how do we solve a system of equations with multiple unknowns? As mentioned earlier, equation solvers for non-linear systems of equations are best avoided, but at the equilibrium stage we are no longer (depending on the problem) looking for as exact a solution as in the inner value function problem.

Besides, though hard to prove, the demand-supply relationships in general equilibrium are quite standard, so it is likely the equations are well-behaved. So maybe it is safe to use a Newton, Broyden, or even simplex routine. Also, while not recommended by applied mathematicians, for economics practitioners, it may also make sense to just minimize the objective function at the equilibrium stage (e.g., minimize $r_S - r_D$ rather than solve for the zero), which will transform the equilibrium problem into an optimization problem rather than an equation solver.

In such cases, that is where you have multiple equilibrium variables and are not sure how to solve out for them other than to bring the sum of squared distances as close to zero possible, you might as well include the equilibrium in the most outerloop—that is, calibrate the equilibrium variables along with the model parameters.

References

Aiyagari, S. Rao, "Uninsured Idiosyncratic Risk and Aggregate Saving," *Quarterly Journal of Economics*, August 1994, *109* (3), 659–684.

Castañeda, Ana, Javier Díaz-Giménez, and José-Víctor Ríos-Rull, "Accounting for the U.S. Earnings and Wealth Inequality," *Journal of Political Economy*, 2003, *111*, 818–857.

Haan, Wouter J. Den and Pontus Rendahl, "Solving the incomplete markets model with aggregate uncertainty using explicit aggregation," *Journal of Economic Dynamics and Control*, January 2010, *34* (1), 69–78.

Heer, Burkhard and Alfred Maussner, *Dynamic General Equilibrium Modeling*, 2 ed., Springer, 2009.

Huggett, Mark, "The Risk Free Rate in Heterogeneous-Agent, Incomplete-Insurance Economies," *Journal of Economic Dynamics and Control*, 1993, *17*, 953–969.

Kopecky, Karen and Richard Suen, "Finite State Markov-chain Approximations to Highly Persistent Processes," *Review of Economic Dynamics*, July 2010, *13* (3), 701–714.

Krusell, Per and Anthony Smith, "Income and Wealth Heterogeneity in the Macroeconomy," *Journal of Political Economy*, 1998, *106*, 867–896.

Ríos-Rull, José-Víctor, "Computation of Equilibria in Heterogeneous-Agent Models," in Ramon Marimon and Andrew Scott, eds., *Ramon Marimon and Andrew Scott, eds.*, Oxford University Press, 1999.

Tauchen, George, "Finite State Markov-Chain Approximations to Univariate and Vector Autoregressions," *Economic Letters*, 1986, *20*, 177–181.