Quantitative Macroeconomics

Aiyagari: Computational Details

Tomás R. Martinez

UnB

References

- Heer and Maussner (2009): Ch. 1, 4, and 7.
- Fehr and Kindermann (2019): Ch. 8 and 9.
- There are online notes by some very smart people: Makoto Nakajima, Alisdair McKay, Jesús Fernández-Villaverde, Econ-ark, and many others. A big thanks to all of them.
- I will assume that you know basic Dynamic Programming, Value Function Iteration and Markov chains.
  - If you need a refresh, check Ljunqvist and Sargent: Ch. 2, 3, and 4.
- An advanced reference: Maliar and Maliar (2014, Numerical Methods for Large-Scale Dynamic Economic Models).

Baseline Aiyagari Model

- Aggregate production function: $Y_t = K_t^{\alpha} N_t^{1-\alpha}$
- Household Problem:

$$V(a,s) = \max_{a' \geq -\phi} \{u((1+r)a + w\exp\{s\} - a') + \beta \mathbb{E}[V(a',s')|s]\}$$

$$s_t = \rho s_{t-1} + \sigma \varepsilon_t, \qquad \text{where } \varepsilon \sim N(0,1)$$

- We must solve for the interest rate that clears the asset market:

$$\int_{A \times S} a d\lambda(a,s) = K$$

where λ is the invariant distribution and K is the capital demand from the firm's problem.

Algorithm in a Nutshell

The solution of the problem is characterized by a **fixed point:**

1. Guess an interest rate r_0 (or quantity of capital K_0).
2. Using r_0 in the solution of the firm's problem, recover capital demand and the implied wage.
3. Given w and r_0, solve for policy functions of the household.
4. Given the household policy functions', solve for the invariant distribution.
5. Given the invariant distribution, compute the excess demand function:

$$\Phi(r) = \int_{A\times S} a d\lambda(a, s; r) - K(r),$$

   if $\Phi(r) = 0$, we found equilibrium. Otherwise, update the guess r_0 and go back to step 2.

Discretization

- We have to solve the HH problem using **global methods**.
- The first step is to discretize the state space!
- **Trade-off between speed and accuracy**: with more gridpoints, you have a more accurate solution, but it takes more time to solve the problem.
- You must **always** check that the solution of your model is not affected by the choices you make regarding the gridpoints!
  - Sanity check: increase the number of gridpoints and change the boundary of the state space to check if anything is changing.

Discretization: Assets

- Choose the number of gridpoints n_A, the bounds of the state space (a_1, a_{n_A}) and a discretization method so the grid is:

$$g_A = [a_1, a_2, ..., a_{n_A}]'.$$

- Bounds: $a_1 = -\phi$, the upper bound a_{n_A} must be chosen so it never binds.
  - A simple trick is to use the steady state k of the Neoclassical Growth Model and multiply by a constant.
- **Gridpoints distance**:
  - Simplest way: equal distance between the gridpoints.
  - Alternative: include more points in the area with more curvature/kinks. In Aiyagari, this is closer to the borrowing constraint.

$$a_i = a_1 + (a_{n_A} - a_1)\frac{(1+\nu)^{i-1} - 1}{(1+\nu)^{n_A - 1} - 1} \qquad \text{for } i = 1, 2, ..., n_A.$$

where ν is the growth rate between points.

Discretization: Assets

- Very often people use the same gridpoints to approximate both the policy functions and the invariant distribution, **but it does NOT to be the case**!
- The exact number of gridpoints will depend on the method used to solve for the policy function and the invariant distribution.
  - Less accurate methods require more grid points.
  - If you plan to use interpolation methods, you may also want to be extra careful.
- Another strategy is to use a multigrid method: solve the model with a coarse grid and then use it as a guess to refine the solution.
- **Extra advanced references**: Maliar and Maliar (2014, Handbook of Computational Econ.); Brumm and Scheidegger (2017, ECTA).

Discretization: Markov Chain

- We must discretize: $s_t = \rho s_{t-1} + \sigma \varepsilon_t,$ where $\varepsilon \sim N(0,1)$, in a discrete Markov Chain with n_S points.

- The output will be a grid and a transition matrix:

$$g_S = [s_1, s_2, ..., s_{n_S}]', \qquad \text{and } \Pi = \begin{bmatrix} \pi_{11} & \pi_{12} & \cdots \\ \vdots & \ddots & \\ \pi_{n_S 1} & & \pi_{n_S n_S} \end{bmatrix}$$

  where the elements of $\Pi_{n_S \times n_S}$ are the probabilities of going from state i to j:

$$\pi_{ij} = Prob(s' = s_j | s = s_i).$$

- **Two standard methods to discretize the AR(1):** Tauchen and Rouwenhorst.

Discretization: Tauchen

- **References:** Tauchen (1986) and Flodén (2008, Econ. Letters).
- Start by choosing (symmetrical) end points using the unconditional standard deviation:

$$-s_1 = s_{n_S} = m\left(\frac{\sigma^2}{1-\rho^2}\right)^{\frac{1}{2}},$$

  where $m > 0$ is a constant.
- Tauchen uses $m = 3$, Floden advocates for $m = 1.2\ln(n_S)$.
- Choose equidistant points between s_1 and s_{n_S}.
  - Denote d as the distance between points.
  - Another option is Gaussian nodes (Tauchen and Hussey, 1991).

Discretization: Tauchen

- Compute the transition probabilities between points using the distribution of ε:

$$\pi_{ik} = \begin{cases} \Phi\left(\dfrac{s_1 - \rho s_i + d/2}{\sigma}\right) & \text{if } k = 1 \\ \Phi\left(\dfrac{s_k - \rho s_i + d/2}{\sigma}\right) - \Phi\left(\dfrac{s_k - \rho s_i - d/2}{\sigma}\right) & \text{if } 1 < k < n_S \\ 1 - \Phi\left(\dfrac{s_{n_S} - \rho s_i - d/2}{\sigma}\right) & \text{if } k = n_S \end{cases}$$

  where Φ is the CDF of the $N(0,1)$.

- Intuitively, Tauchen approximates the AR(1) by targeting the conditional distribution of s.
- In general, the method is pretty efficient in matching the AR(1) with a low n_S as long the process is not too close to a unit root.

Discretization: Rouwenhorst

- If the AR(1) is too persistent (e.g., $\rho > 0.9$), you should use **Rouwenhorst**. The reference is Kopecky and Suen (2010, RED).

- The idea is to approximate the process to a Markov Chain that converges to the invariant binomial distribution.

- Grid: equidistant and symmetric, with $-s_1 = s_{n_S} = \psi$.

- Using three parameters of the Markov chain (p, q, ψ), the method can match exactly:
  - Unconditional mean, variance and first-order correlation of s_t;
  - In the particular example we used, it matches the conditional mean and variance as well.

- Rouwenhorst does not use information about the distribution of ε. As long we are mostly interested in the first two moments, the method should work well.

Discretization: Advanced Issues

- In the case of non-stationary processes (e.g., life-cycle models), you should adapt the methods. See Fella, Gallipoli and Pan (2019, RED).
- For more general processes such as non-normal, asymmetric distributions, and correlated shocks you may have to use simulation methods. See De Nardi, Fella and Paz-Pardo (2020, JEEA).
- Multivariate processes are also described in Tauchen.
- There is also discretization based on Gauss-Hermite quadrature (see Maliar and Maliar).
  - They are often useful if you plan to pre-compute the expectation of the VF as in Judd, the Maliars, and Tsener (2017, QE).

Household Problem

- Once we have discretized the state space, we can solve the household problem using a variety of global methods:
  - Value Function Iteration;
  - Policy Function Iteration;
  - Projection Methods.
- Here I will focus in the first two. If we have time, I may discuss the last one.
- Strictly speaking, projection is just a way to approximate the value/policy function, but it is useful to separate it in a different method.

Value Function Iteration

- Once we have the asset and labor grid, we define the discretized value function on the same grid points: $V(a_i, s_j) = V_{ij}$, where V is a $n_A \times n_S$ array:

$$V = \begin{bmatrix} V_{11} & V_{12} & \dots & V_{1n_S} \\ \vdots & \ddots & & \\ V_{n_A 1} & & \dots & V_{n_A n_S} \end{bmatrix}$$

- Our goal is to solve the following Dynamic Programming problem:

$$V_{ij} = \max_{a'_k \geq -\phi} \left\{ u((1+r)a_i + w\exp(s_j) - a'_k) + \beta \sum_{m=1}^{n_S} \pi_{jm} V_{km} \right\},$$

where the conditional expectation $\mathbb{E}[V(a', s')|s] = \sum_{m=1}^{n_S} \pi_{jm} V_{km}$.

Value Function Iteration

- We know that under certain conditions Banach Fixed Point Theorem applies, and we can use the following iterative procedure:

    1. Guess an initial value function V_{ij}^n.
    2. Compute the continuation value using the conditional expectation $\mathbb{E}[V^n(a'_k, s')|s_j]$.
    3. Given the return function and the continuation value, solve the maximization problem to compute the value function V_{ij}^{n+1} for all state space $(i,\, j)$.
    4. Compute the absolute distance: $d = \max_{i,j} |V_{ij}^{n+1} - V_{ij}^n|$. If $d < tol$, we found V. Otherwise, update the guess $V^n = V^{n+1}$ and repeat.

- The slowest part of the procedure is solving the maximization problem.

Value Function Iteration

- Simplest way to solve the maximization problem: brute force using **Grid Search**.
- **Idea:** for all (i, j), compute the value function for all next period asset a'_k and select the one that yields the maximum:

$$
\begin{aligned}
V_{ij,k}^{n+1} &= \begin{cases} u((1+r)a_i + w\exp(s_j) - a'_k) + \beta \mathbb{E}[V^n(a'_k, s')|s_j], & \text{if } c_{ij,k} > 0 \\ -\infty, & \text{if } \; c_{ij,k} \leq 0 \end{cases} \\
V_{ij}^{n+1} &= \max_k \{V_{ij,1}^{n+1}, V_{ij,2}^{n+1}, ..., V_{ij,n_A}^{n+1}\}
\end{aligned}
$$

- **Issues:**
  - Your optimal policy a' will be defined on-grid. You should have a lot of points in the asset grid to have a good approximation.
  - The **Curse of dimensionality** bites hard. The method is robust but can be very slow.

Value Function Iteration

- Another way to solve the maximization problem is to **interpolate** the value function and use a one-dimension optimization algorithm to solve the $\max$.

- **Interpolation**
  - First, interpolate $V^n(a', s')$, then take the conditional expectation.
  - This gives a continuous function on a' (conditional on s): $\mathbb{E}[\hat{V}^n(a'_k, s')|s_j] = \hat{V}^n(a'; s_j)$, where $\hat{V}$ is the interpolated VF.
  - Since $u()$ is continuous on a', we can define the continuous function φ:

$$\varphi(a'; a_i, s_j) = u((1+r)a_i + w\exp(s_j) - a') + \beta\hat{V}^n(a'; s_j).$$

- **Optimization**
  - Then, we can apply standard optimization routines on $\varphi(a')$ to find the optimal a'^*.
  - The value function is $V_{ij}^{n+1} = \varphi(a'^*; a_i, s_j)$.

Value Function Iteration

Issues:

- Many interpolation algorithms:
  - Linear: fast but not differentiable at the nodes.
  - Cubic: a bit slower, but differentiable at all points.
  - Chebyshev.
- Many optimization algorithms:
  - In general, you should use derivative-free methods (Brent's, Golden-search,...).
  - You can try (faster) Newton-style methods. Just recall that since they need derivatives, your interpolation algorithm should give you a differentiable VF.
- In comparison to brute force algorithms, interpolate + optimization is slower but significantly more accurate (for the same n_A).
  - Once you factor that you can get better accuracy with fewer grid points, interpolation can be faster (usually depends on the problem).

Speeding up VFI: Howard's Improvement Algorithm

- Since the maximization step is slowest part, one popular strategy is to "skip" the $\max$ in a couple of iterations.
- Say that you just finish iteration n, and you have computed V_{ij}^n and the policy $a' = g_{ij}^n$.
  - You can do n_H iterations to update V without solving the $\max$ and keeping $a' = g_{ij}^n$ constant instead:

$$V_{ij}^{n_H+1} = u((1+r)a_i + w\exp(s_j) - g_{ij}^n) + \beta \mathbb{E}[V^{n_H}(g_{ij}^n, s')|s_j]$$

  - Once you finish the n_H, you can do one regular iteration where you compute the policy function and check convergence for the VF.
- It should work with all VFI methods, but for me, Howard's tend to perform better with interpolation-types of maximization than with pure brute force.

Speeding up VFI: Exploiting Monotonicity and Concavity

- Under certain conditions, we know that the VF is monotone and/or concave.
- We can use this information to reduce the state space where we look for a solution.
- **Example:** say your VF is monotone in a
  - then, for two grids i and j:

$$a_i \geq aj \Rightarrow a_i' = g_a(a_i) \geq g_a(a_j) = a_j'.$$

  - Once we solve for a_j, we can reduce the search space for the solution of a_i
- Similarly for concavity (see Heer and Maussner, ch. 4).
- To exploit monotonicity, I like to use the divide-and-conquer algorithm by Gordon and Qiu (2018, QE).

Policy Function Methods

- VFI is robust and works under a wide set of conditions: discrete choices, multiple controls, etc.
- But it tends to be very slow and often not very accurate.
- Policy Function Methods (i.e., iteration on the Euler Equation) are fast and accurate, but not as robust.
- Here I will present the **Endogenous Grid Method** which is likely the most used method to solve consumption-savings problem nowadays.

Endogenous Grid Method

- Let $c = g_c(a, s)$ be the consumption policy function. We want to solve the following functional equation:

$$c^{-\gamma} = \beta(1+r)\mathbb{E}\left[g_c(a', s')^{-\gamma}|s\right]$$
$$c^{-\gamma} = \beta(1+r)\mathbb{E}\left[g_c\left((1+r)a + w\exp(s) - c, s'\right)^{-\gamma}|s\right]$$

- Using a guess for g_c on the grid, standard methods involve solving for c using interpolation and root-finding method.
  - Then, we check if g_c^n is close enough to c. If it is not, update the guess and keep going.
- As we know, using a non-linear equation solver is costly.
- Carroll (2005) introduces the **Endogenous Grid Method**, which bypasses the non-linear solver.
  - See Barillas and Fernández-Villaverde (2007) for an extension that combines VFI and EGM.

Endogenous Grid Method

1. Guess a consumption policy on an exogenously defined grid: $c = g_c^n(a_i, s_j)$;
2. Use the guess to compute the RHS of the EE (note we iterate on a_i'!):

$$RHS(a_i', s_j) = \beta(1+r) \sum_{m=1}^{n_s} \pi_{jm} u'(g_c^n(a_i', s_m')).$$

3. Invert the marginal utility to find the consumption decision (in t) associated to the state $(\hat{a}_i, s_j)$ and asset policy $a_i' = g_a^n(\hat{a}_i, s_j)$:

$$\tilde{c}(a_i', s_j) = u'^{-1}(RHS(a_i', s_j)).$$

   This is the "next iteration" consumption policy $\tilde{c}(a_i', s_j) = g_c^{n+1}(\hat{a}_i, s_j)$. But we cannot compare with the previous iteration, since g_c^n is defined on (a_i, s_j), while g_c^{n+1} is defined on $(\hat{a}_i, s_j)$

Endogenous Grid Method

4. To redefine the consumption policy, g_c^{n+1}, in the same exogenous grid, you must recover the **endogenous grid** $\hat{a}_i$ using the budget constraint:

$$\hat{a}_i = \frac{\tilde{c}(a'_i, s_j) + a'_i - w\exp(s_j)}{1+r}$$

5. Now use the pair of points $(g_c^{n+1}, \hat{a}_i)$ to interpolate and find $g_c^{n+1}(a_i, s_j)$ defined on the exogenous grid.

6. Compute the distance $d = \max_{i,j} |g_c^{n+1}(a_i, s_j) - g_c^n(a_i, s_j)|$. If $d < tol$, we stop. Otherwise update the guess and start over.

Endogenous Grid Method: Borrowing Constraint

- To deal with the borrowing constraint, it is often convenient to interpolate the asset policy instead: $a_i' = g_a(\hat{a}_i, s_j)$.

- Then, after you find the interpolated function $g_a(a_i, s_j)$, you check if the borrowing constraint is binding: $g_a(a_i, s_j) < a_1$.
  - If $g_a(a_i, s_j) < a_1$, set $g_a(a_i, s_j) = a_1$;
  - If $g_a(a_i, s_j) \geq a_1$, do nothing.

- After this correction, you can recover g_c^{n+1} using the budget constraint:

$$g_c^{n+1}(a_i, s_j) = (1 + r)a_i + w \exp(s_j) - g_a(a_i, s_j),$$

  and proceed as usual.

Endogenous Grid Method: Extra Issues

- **Endogenous Labor Supply**: As long the labor supply equation has an analytical solution, $n = g_n(c, s_j)$, we just need to substitute it in step 4.
  - You may have to use a non-linear solver for the borrowing constraint, but this is just for a few grids.
- **Discrete choices and non-convexities**: If there is non-convex choice sets the Euler Equation is not sufficient for the solution.
  - You must add a step where you check whether the value function is indeed a maximum. See Fella (2014), Iskhakov et al (2017), Druedahl (2020).
- **Multiple control variables**: See Ludwig and Schön (2018).
- Another fast method but not as popular is the **Envelope Condition Method**. See Maliar and Maliar (2013).

Euler Equations Errors

- If you want to compare the accuracy of different solution methods, you can compute the Euler Equation errors.
- Define the approximation error, ε, using:

$$
\begin{aligned}
& u'(c_t(1-\varepsilon)) = \beta(1+r)\mathbb{E}_t[u'(c_{t+1})] \\
\iff \quad & \varepsilon = 1 - \frac{u'^{-1}(\beta(1+r)\mathbb{E}_t[u'(c_{t+1})])}{c_t}
\end{aligned}
$$

- One can compute ε using a different grid than the one used to solve the model, or simulate the decisions using a long history of shocks.
- See Aruoba, Fernandez-Villaverde and Rubio-Ramirez (2006, JEDC) for an application of the Euler Errors.

Invariant Distribution

- Once we have the policy functions of the HH problem, we can compute the invariant distribution, $\lambda(a, s)$

- A couple of methods:
  1. Monte-Carlo simulation;
  2. Non-stochastic simulation;
  3. Parameterized distributions.

- I will focus on method 2. Method 1 is usually too slow, method 3 is useful with aggregate uncertainty but the implementation is bit cumbersome (see Algan, Allais, and Haan (2008) to learn about it).

Invariant Distribution

- Approximate the density by a histogram over a fixed grid (Young (2010, JEDC)).
- The asset grid has to be sufficiently fine. It does NOT need to be the same as the one used for the VF/policy function.
- The distribution will be stored in an array with $n_A * n_S$ entries: $\lambda(a_i, s_j)$.
  - It can be a matrix $n_A \times n_S$ or a vector $n_A * n_S \times 1$.
- Then, we build a transition function that gives the probability an agent move from state (a, s) to state (a', s'):

$$\begin{aligned}\mathcal{P}(a', s', a, s) &= Prob[(a_{t+1} = a', s_{t+1} = s')|a_t = a, s_t = s] \\ &= Prob[a_{t+1} = a'|a_t = a, s_t = s] * Prob[s_{t+1} = s'|s_t = s],\end{aligned}$$

Invariant Distribution

- Note that the transition function together with the distribution array defines a Markov chain.

- From a distribution $\lambda_t(a, s)$, we can compute the mass at node (a_k, s_m) in the next period:

$$\lambda_{t+1}(a_k, s_m) = \sum_{i=1}^{n_A} \sum_{j=1}^{n_S} \lambda_t(a_i, s_j) \mathcal{P}(a_k, s_m, a_i, s_j)$$

- Computing $\mathcal{P}(a_k, s_m, a_i, s_j)$ requires the transition matrix for s and the policy function $g_a(a, s)$.

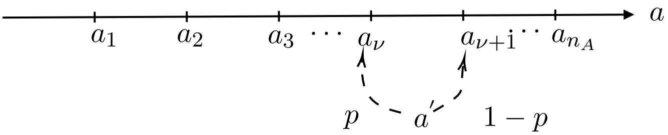
$$\mathcal{P}(a_k, s_m, a_i, s_j) = Prob[a_{t+1} = a_k | a_t = a_i, s_t = s_j] * Prob[s_{t+1} = s_m | s_t = s_j]$$

Invariant Distribution

- To get $Prob[a_{t+1} = a_k | a_t = a_i, s_t = s_j]$, note that the policy function gives $a' = g(a_i, s_j)$.
- The problem is that a' usually lies off-grid. The trick is to allocate some households in the grid below, and some in the grid above in a way to preserve the aggregate mass of assets:

$$a' = pa_\nu + (1-p)a_{\nu+1}$$

where a_ν is the grid point just below a'.



- $Prob[a_{t+1} = a_\nu] = p$, $Prob[a_{t+1} = a_{\nu+1}] = 1-p$, and $Prob[a_{t+1} = a_i] = 0$ for all other i.

Invariant Distribution

- Multiplying $Prob[a_{t+1} = a_k | a_t = a_i, s_t = s_j]$ with Π and we have the transition function $\mathcal{P}$.

- Since this is just a Markov chain, you can find the stationary distribution by solving a linear system using standard methods: $\lambda\mathcal{P} = \lambda$.
  - Note that $\mathcal{P}$ is often a sparse matrix!

- Iteration methods tend to be robust:
  1. Guess $\lambda_n(a, s)$ (initial guess can be a uniform mass or anything that sums to one).
  2. Compute the next period distribution, $\lambda_{n+1}(a, s)$, applying the transition function.
  3. Check convergence: $d = \max_{i,j} |\lambda_{n+1}(a_i, s_j) - \lambda_n(a_i, s_j)|$. If $d < tol$ finish the iteration; otherwise try again using λ_{n+1} as a guess.

Invariant Distribution

- Once we have the invariant distribution $\lambda(a, s)$, we can compute the aggregate asset supply:

$$\mathbb{E}a = \sum_{i=1}^{n_A} a_i \sum_{j=1}^{n_S} \lambda(a_i, s_i)$$

- Which together with the capital demand $K(r)$ defines the equilibrium condition.

- We are now ready to define an iterative procedure to find the steady state equilibrium.

Finding the Equilibrium: Algorithm

1. Guess an interest rate, r_n.

2. Use the interest rate to compute the capital demand and wage using the firm's optimality condition:

$$K(r) = \left(\frac{\alpha}{r+\delta}\right)^{\frac{1}{1-\alpha}} N \qquad \text{and ,} \qquad w(r) = (1-\alpha)\left(\frac{K(r)}{N}\right)^{\alpha}.$$

   Note that the aggregate labor supply, N, is time invariant and can be computed using the invariant distribution of the labor endowment, μ_j:

$$N = \sum_{j=1}^{n_S} s_j \mu_j$$

Finding the Equilibrium: Algorithm

3. Given r_n and $w(r_n)$, solve the household problem. Denote the policy function as $g_a(a, s; r_n)$.

4. Using the policy function $g_a(a, s; r_n)$ and the law of motion of the shock s, find the stationary distribution $\lambda(a, s; r_n)$ and the aggregate asset supply:

$$\mathbb{E}a(r_n) = \sum_{i=1}^{n_A} a_i \sum_{j=1}^{n_S} \lambda(a_i, s_i; r_n)$$

5. Compute the excess demand function:

$$\Phi(r) = \mathbb{E}a(r) - K(r),$$

6. if $|\Phi(r_n)| < tol$, we found an equilibria. Otherwise, update r and try again.
   - If $\Phi(r) < 0$, capital demand is too low. Decrease r.
   - If $\Phi(r) > 0$, capital demand is too high. Increase r.

Finding the Equilibrium

- Finding an equilibrium boils down to finding a root of the excess demand function.
  - Usually bracketing methods work well: bisection, Brent's, etc.

- **Initial guess:** theoretically, good bounds for r are:
  - Upper bound: $r_u = 1/\beta - 1$.
  - Lower bound: $r_l = -\delta$.

- Alternatively, we can iterate on K (which defines r and w using the firm's foc).
  - In this case, we can update the capital guess, K^n, using the following strategy:

$$K^{n+1} = d\mathbb{E}a + (1-d)K^n \tag{1}$$

where $d \in (0,1]$ is a dampening parameter.

Finding the Equilibrium

- **Endogenous labor supply**: In case of endogenous labor supply, aggregate labor supply, N_t, will change with prices.
  - Iterate on capital-labor ratio instead: $k = K/L$.

- **Fiscal policy and tax instruments**: it involves add an extra condition, the **government budget constraint**.
  - In some cases, it is possible to include inside the loop.
  - In more complicated cases, one must include as an extra condition together with the asset market excess demand.

- **Other market clearing conditions**:
  - Must be added in the excess demand loop. One can use multivariate optimization/root-finding algorithms (e.g., simplex), or solve one condition at a time.