A more technical introduction to the "fake news algorithm" for sequence-space Jacobians

In class, I worked through the "fake news algorithm" for efficiently computing sequence-space Jacobians, emphasizing intuition (with "insights" 1–3) and practical implementation in code rather than technical derivations.

Here, I show how we can obtain the algorithm in a more technical way, which may be a useful complement to the intuition and code in class. This material closely follows the derivation in section 3.2 of the paper "Using the Sequence-Space Jacobian to Solve and Estimate Heterogeneous-Agent Models" (Auclert Bardoczy Rognlie Straub, Econometrica 2021), which you can also consult as a reference, but here I try to focus on the main thread of the derivation and add a bit of class-relevant context.

To be clear, reading this note is not necessary for the class. I described the full algorithm because we need it to calculate Jacobians, and I didn't want that to be a total black box. The concept of the fake news matrix will also be relevant later when we deviate from full information and rational expectations, and you might be able to use some of the insights from this algorithm (especially "insight 1") on your own even if you don't use the full algorithm. But compared to the other material we've covered in class, precisely understanding the full algorithm is not essential for the class. This note is mainly intended for the subset of you (maybe 10–20%) who have expressed interest in understanding the algorithm in more detail.

Initial setup. Suppose that there is some outcome of interest—say, the consumption or asset policy—that we are interested in. At time t, we denote by $\mathbf{y}_t$ the vector giving this outcome across all idiosyncratic states. I'll assume for now that this is some kind of standard, forward-looking policy function.

Further, we'll suppose that there is a potentially time-varying transition matrix Λ_t across states. We multiply by the transpose of this matrix to update the distribution of agents across states: $\mathbf{D}_{t+1} = \Lambda_t' \mathbf{D}_t$.

Finally, suppose that we're interested in the aggregate value of the outcome across the distribution at each t. To get that, we need to take the dot product of $\mathbf{y}_t$ and $\mathbf{D}_t$, which (since these are column vectors) we denote by $Y_t \equiv \mathbf{y}_t' \mathbf{D}_t$.

Mapping to what we've done. In the code, we compute the asset and consumption policy functions $a'(e, a)$ and $c(e, a)$ over some discrete grid of states (e, a). Usually we keep these as two-dimensional arrays, but above I'm assuming that they've been flattened to be $(n_e n_a) \times 1$ dimensional column vectors $\mathbf{y}_t$.

Similarly, we compute the approximate distribution $D(e, a)$ over the same discrete grid. Again, above, I'm assuming that we rewrite this as a $(n_e n_a) \times 1$ dimensional vector $\mathbf{D}_t$. Then, aggregating the asset or consumption policy on the distribution, which we normally do by taking element-by-element dot products of the two-dimensional arrays, now amounts to just taking the ordinary vector dot product of $\mathbf{y}_t$ and $\mathbf{D}_t$.

The transition matrix between states, Λ_t, is in principle an $(n_e n_a) \times (n_e n_a)$ matrix giving the transition probability between *any* two idiosyncratic states.[1] This matrix reflects both the exogenous probability of moving between states e and the endogenous probability of moving between states a (which we model as a lottery between neighboring gridpoints). This is not a matrix that we ever actually form, since it would be impractically large. But conceptually it exists, and when we iterate forward on the distribution from $\mathbf{D}_t$ to $\mathbf{D}_{t+1}$, effectively we're performing (in a more efficient way that never involves actually forming the matrix!) $\mathbf{D}_{t+1} = \Lambda_t' \mathbf{D}_t$.

Results with this setup. Now imagine that there is some shock dx to an input that matters for the household (e.g. the real interest rate) at date s, and that at every other date, all inputs to the household problem are unchanged. We denote the resulting paths of $\mathbf{y}_t$, $\mathbf{D}_t$, and Y_t with s superscripts: e.g. Y_t^s is the aggregate Y_t when the household is hit with dx at date s.

[1] Note that this is *not* just the $n_e \times n_e$ matrix giving the transition probability between exogenous states, which we called Π. It's much bigger!

What we want to find is the Jacobian matrix

$$\mathcal{J}_{t,s} \equiv \frac{dY_t^s}{dx}$$

How do we obtain this? Totally differentiating $Y_t = \mathbf{y}_t'\mathbf{D}_t$ around the steady state, the product rule implies:

$$dY_t^s = d\mathbf{y}_t^{s\prime}\mathbf{D}_{ss} + \mathbf{y}_{ss}'d\mathbf{D}_t^s \tag{1}$$

This says that the aggregate change in Y_t, to first order, is the individual-level policy shock $d\mathbf{y}_t^{s\prime}$ aggregated across the steady-state distribution $\mathbf{D}_{ss}$, plus the steady-state policy $\mathbf{y}_{ss}'$ aggregated across the perturbation to the distribution.[2]

Our **insight 1** is that $\mathbf{y}_t^s$ only depends on $s-t$: the policy function at the individual level at t only depends on the time $s-t$ until the shock at s. (It is zero if $s < t$, since your policy doesn't change in response to past shocks.) This can be proven directly using the Bellman equation, although we won't do it here. Similarly, Λ_t^s only depends on $s-t$; since, for instance, in our example the transition probability between states just depends on the asset policy, which is forward-looking and depends only on $s-t$.

This insight makes it easy to calculate all the $\mathbf{y}_t^s$ and Λ_t^s that we need in practice. We just start with $s = T-1$ for some large T and iterate backward to obtain $\mathbf{y}_t^{T-1}$ and Λ_t^{T-1} for all $0 \leq t \leq T-1$. For arbitrary s, we then use $\mathbf{y}_t^s = \mathbf{y}_{T-1-(s-t)}^{T-1}$ (for $s \geq t$; 0 for $s < t$) and similar for Λ_t^s.

This gives us the first term of (1), but what about the $d\mathbf{D}_t^s$? Let's first totally differentiate $\mathbf{D}_t = \Lambda_{t-1}'\mathbf{D}_{t-1}$ to obtain

$$d\mathbf{D}_t^s = d\Lambda_{t-1}^{s\prime}\mathbf{D}_{ss} + \Lambda_{ss}'d\mathbf{D}_{t-1}^s \tag{2}$$

We can recursively substitute (2) into itself once to obtain

$$d\mathbf{D}_t^s = d\Lambda_{t-1}^{s\prime}\mathbf{D}_{ss} + \Lambda_{ss}'d\Lambda_{t-2}^{s\prime}\mathbf{D}_{ss} + (\Lambda_{ss}')^2 d\mathbf{D}_{t-2}^s$$

and, continuing the process, obtain[3]

$$d\mathbf{D}_t^s = d\Lambda_{t-1}^{s\prime}\mathbf{D}_{ss} + \Lambda_{ss}'d\Lambda_{t-2}^{s\prime}\mathbf{D}_{ss} + \ldots + (\Lambda_{ss}')^{t-1}d\Lambda_0^{s\prime}\mathbf{D}_{ss} \tag{3}$$

This may look formidable, but it's fairly intuitive: it expresses the perturbation to the distribution at date t as the sum of perturbations to the transition matrix at prior dates, which (given our assumption that the initial distribution at 0 is fixed and unaffected by the shock) are the ultimate causes of the distribution being different today. For instance, in (3) the term $d\Lambda_{t-1}^{s\prime}\mathbf{D}_{ss}$ represents the transition matrix between $t-1$ and t changing, then the term $\Lambda_{ss}'d\Lambda_{t-2}^{s\prime}\mathbf{D}_{ss}$ represents the transition matrix between $t-2$ and $t-1$ changing (which causes a perturbation $d\Lambda_{t-2}^{s\prime}\mathbf{D}_{ss}$ to the distribution at $t-1$, which we iterate forward to t using Λ_{ss}'), and so on.

For $t, s > 0$, we can condense the messiness of (3) to something much nicer by writing

$$\begin{aligned} d\mathbf{D}_t^s &= \left(d\Lambda_{t-1}^{s\prime}\mathbf{D}_{ss} + \Lambda_{ss}'d\Lambda_{t-2}^{s\prime}\mathbf{D}_{ss} + \ldots + (\Lambda_{ss}')^{t-2}d\Lambda_1^{s\prime}\mathbf{D}_{ss}\right) + (\Lambda_{ss}')^{t-1}d\Lambda_0^{s\prime}\mathbf{D}_{ss} \\ &= \left(d\Lambda_{t-2}^{s-1\prime}\mathbf{D}_{ss} + \Lambda_{ss}'d\Lambda_{t-3}^{s-1\prime}\mathbf{D}_{ss} + \ldots + (\Lambda_{ss}')^{t-2}d\Lambda_0^{s-1\prime}\mathbf{D}_{ss}\right) + (\Lambda_{ss}')^{t-1}d\Lambda_0^{s\prime}\mathbf{D}_{ss} \\ &= d\mathbf{D}_{t-1}^{s-1} + (\Lambda_{ss}')^{t-1}d\Lambda_0^{s\prime}\mathbf{D}_{ss} \end{aligned} \tag{4}$$

Here, in the first line, we use insight 1 to replace $d\Lambda_{t-1}^{s\prime}$ with $d\Lambda_{t-2}^{s-1\prime}$, $\Lambda_{ss}'d\Lambda_{t-2}^{s\prime}$ with $\Lambda_{ss}'d\Lambda_{t-3}^{s-1\prime}$, and so on, for all terms except the last. We then realize that the grouping in parentheses (all terms except the last) is just $d\mathbf{D}_{t-1}^{s-1}$, according to (3).

What is the meaning of (4)? It says that the perturbation to the distribution at date t, in response to a shock at date s, is *almost* the same as the perturbation at date $t-1$ in response to a shock at date $s-1$. The only difference is the term $(\Lambda_{ss}')^{t-1}d\Lambda_0^{s\prime}\mathbf{D}_{ss}$, reflecting the fact that in the former case, at date 0 we anticipated the shock s periods ahead of time (an "extra period of anticipation"). This led to a perturbation $d\mathbf{D}_1^s = d\Lambda_0^{s\prime}\mathbf{D}_{ss}$, which at date t has the persistent effect $(\Lambda_{ss}')^{t-1}d\Lambda_0^{s\prime}\mathbf{D}_{ss}$.

[2] Terms like $d\mathbf{y}_t^{s\prime}d\mathbf{D}_t^s$ only show up when we are doing a second- or higher-order approximation.

[3] Note, annoyingly, that superscripts have two different meanings here: e.g. $d\Lambda_{t-1}^s$ is the change in the transition matrix at date $t-1$ caused by the shock at date s, while $(\Lambda_{ss}')^{t-1}$ is the transpose of the steady-state transition matrix to the $t-1$ power.

If we multiply both sides of (4) by $\mathbf{y}'_{ss}$ and then add $d\mathbf{y}_t^{s\prime}\mathbf{D}_{ss} = d\mathbf{y}_{t-1}^{s-1\prime}\mathbf{D}_{ss}$ to both sides, we get

$$\underbrace{d\mathbf{y}_t^{s\prime}\mathbf{D}_{ss} + \mathbf{y}'_{ss}d\mathbf{D}_t^s}_{=dY_t^s} = \underbrace{d\mathbf{y}_{t-1}^{s-1\prime}\mathbf{D}_{ss} + \mathbf{y}'_{ss}d\mathbf{D}_{t-1}^{s-1}}_{=dY_{t-1}^{s-1}} + \mathbf{y}'_{ss}(\Lambda'_{ss})^{t-1}\underbrace{d\Lambda_0^{s\prime}\mathbf{D}_{ss}}_{=d\mathbf{D}_1^s} \tag{5}$$

where equality to dY_t^s and dY_{t-1}^{s-1} follows from (1). If we define the "fake news matrix" for $t, s > 0$ by $\mathcal{F}_{t,s} \equiv \mathcal{J}_{t,s} - \mathcal{J}_{t-1,s-1}$, then the above says that, for $t, s > 0$,

$$\mathcal{F}_{t,s}dx = dY_t^s - dY_{t-1}^{s-1} = \mathbf{y}'_{ss}(\Lambda'_{ss})^{t-1}d\mathbf{D}_1^s \tag{6}$$

This is our **insight 2**. It says that the difference between the aggregate response at date t to a date-s shock and the aggregate response at date $t-1$ to a date-$s-1$ shock is given by a single term, $\mathbf{y}'_{ss}(\Lambda'_{ss})^{t-1}d\mathbf{D}_1^s$. We can interpret this term as the effect of anticipating, at date 0, that there will be a shock at date s. This anticipation causes a change of $d\mathbf{D}_1^s = d\Lambda_0^{s\prime}\mathbf{D}_{ss}$ in the distribution at date 1, and this makes a contribution to the distribution at date t of $(\Lambda'_{ss})^{t-1}d\mathbf{D}_1$, which changes Y_t by $\mathbf{y}'_{ss}(\Lambda'_{ss})^{t-1}d\mathbf{D}_1^s$, the term on the right in (6).

We sometimes think of this anticipation effect as the impulse response to a "fake news shock": a shock where at date 0, the change at date s is announced, and then at date 1 the change is retracted.

On its own, (6) is nice but not that useful, because it still seems costly to evaluate. For each s, it looks like we need to calculate $d\mathbf{D}_1^s = d\Lambda_0^{s\prime}\mathbf{D}_{ss}$ and then iterate forward on the distribution, for each t, to get all the $(\Lambda'_{ss})^{t-1}d\mathbf{D}_1^s$.

But with a slight change in perspective, we can do much better. That's because the first two factors in (6), $\mathbf{y}'_{ss}(\Lambda'_{ss})^{t-1}$, *don't depend on the date of the shock* s. This means we can calculate them once and reuse them for all s!

We'll define the tth *expectation function* to be[4]

$$\mathcal{E}_t \equiv \Lambda_{ss}^t\mathbf{y}_{ss} \tag{7}$$

The expectation function $\mathcal{E}_t$ gives, for every state today, the expected value of $\mathbf{y}$ in t periods, assuming that we follow the steady-state policy (embedded in steady-state Λ_{ss}). For instance, if $\mathbf{y}$ is consumption, then $\mathcal{E}_t$ will be, for each person today, how much on average they'll consume in t periods. This reflects the general point that multiplying by a Markov matrix takes expectations (whereas multiplying by the Markov matrix's transpose iterates forward on a distribution). Given this definition, (6) reduces to just

$$\mathcal{F}_{t,s}dx = \mathcal{E}'_{t-1}d\mathbf{D}_1^s \tag{8}$$

The beauty of (7) and (8) is that the $\mathcal{E}_t$ can be calculated recursively using $\mathcal{E}_t = \Lambda_{ss}\mathcal{E}_{t-1}$. This is our **insight 3**. Like with the distribution, we don't explicitly form Λ_{ss}, which is far too large at $(n_e n_a) \times (n_e n_a)$, but instead we write code that directly applies Λ_{ss} to a vector, taking advantage of its special structure. This code is very similar to our code for updating the distribution, which applies Λ'_{ss}.

Putting it all together to obtain an algorithm. Originally, we defined the fake news matrix $\mathcal{F}_{t,s}$ for $t, s > 0$ as $\mathcal{F}_{t,s} \equiv \mathcal{J}_{t,s} - \mathcal{J}_{t-1,s-1}$, and we eventually found that it could be evaluated by (8).

When t or s is 0, we'll adopt the convention that $\mathcal{F}_{t,s} = \mathcal{J}_{t,s}$. For $t > 0$ and $s = 0$, we can calculate this with the same formula (8); this is the effect at later dates t, persisting through the distribution, of a shock that *actually happens* at date 0. For $t = 0$, we write $\mathcal{F}_{0,s} = \mathcal{J}_{0,s} = (d\mathbf{y}_0^s)'\mathbf{D}_{ss}/dx$; this is the anticipatory effect at 0 of a shock that is expected to happen at s.

Note that if we write $\mathcal{J}_{t,s} = \mathcal{F}_{t,s} + \mathcal{J}_{t-1,s-1}$ for $t, s > 0$ with the base case $\mathcal{J}_{t,s} = \mathcal{F}_{t,s}$ when t or s is 0, we can recursively calculate $\mathcal{J}$ from $\mathcal{F}$.

Combining all our results thus far, we have an effective algorithm (the "fake news algorithm") to obtain the Jacobian $\mathcal{J}_{t,s}$:

[4]This is called an "expectation vector" in the sequence-space Jacobian paper, but I think "expectation function" is more evocative, and is parallel with the idea of a "policy function" over the state space.

1. First, given some small shock dx at $T-1$, we iterate backward to find $\mathcal{Y}_s \equiv (d\mathbf{y}^{T-1}_{T-1-s})\mathbf{D}_{ss}/dx$ and $\mathcal{D}_s \equiv (d\Lambda^{T-1}_{T-1-s})'\mathbf{D}_{ss}/dx$ for each s. $\mathcal{Y}_s$ and $\mathcal{D}_s$ give the response of aggregate output at date 0 and the distribution at date 1 to a shock at date s, respectively.

2. Second, starting with $\mathcal{E}_0 \equiv \mathbf{y}_{ss}$, repeatedly take expectations using $\mathcal{E}_t = \Lambda_{ss}\mathcal{E}_{t-1}$ to obtain the expectation functions $\mathcal{E}_t = \Lambda^t_{ss}\mathbf{y}_{ss}$ for all $t = 0, \ldots, T-2$.

3. Construct the fake news matrix $\mathcal{F}$. For row $t = 0$, write $\mathcal{F}_{0,s} = \mathcal{Y}_s$. For rows $t \geq 1$, calculate $\mathcal{F}_{t,s} = \mathcal{E}'_{t-1}\mathcal{D}_s$.

4. Construct the Jacobian $\mathcal{J}$, writing $\mathcal{J}_{t,s} = \mathcal{F}_{t,s}$ whenever t or s is 0, and calculating all other entries with the recursion $\mathcal{J}_{t,s} = \mathcal{J}_{t-1,s-1} + \mathcal{F}_{t,s}$.

Why is this so fast? A direct or "brute-force" approach to calculating $\mathcal{J}_{t,s}$ would separately calculate each column s by finding the impulse response to a small shock dx at date s. Naively, for each s this would involve iterating backward and forward T times to calculate the policy and then the distribution. Since we need to do this for T values of s, the total cost would be T^2 backward and forward iterations.

With insight 1, we see how we can avoid reduce this to only T backward iterations total, but it still seems as if the process might require T^2 forward iterations on the distribution. Insights 2 and 3, however, show that we only need to do around T "expectation" iterations[5] to get the expectation function, which we perform in step 2, and that this circumvents the need for forward iterations.

The combined effect of these insights is that in the first two steps of the algorithm, we do work that is proportional to T rather than T^2. Since these steps require working with the full heterogeneous-agent state space, this is a very important improvement in efficiency, especially when T is large.

In step 3, we have to form approximately T^2 dot products $\mathcal{F}_{t,s} = \mathcal{E}'_{t-1}\mathcal{D}_s$, so we're still doing T^2 work somewhere here. But dot products are *extremely* efficiently implemented on a computer, so this is very cheap compared to almost anything else we do with a heterogeneous-agent model. Indeed, if we stack the $\mathcal{E}_t$ and $\mathcal{D}_s$ in matrices, then we can calculate all the dot products at once in a single matrix multiplication, which is very, very fast—so this step is almost never a bottleneck at all.

This is even more true for step 4, where we also have to do roughly T^2 calculations to fill out the matrix—but these are trivial scalar additions $\mathcal{J}_{t,s} = \mathcal{J}_{t-1,s-1} + \mathcal{F}_{t,s}$, and very cheap.

So, if we wanted to sum up why this algorithm is so much more efficient, we'd say this: in a brute-force approach, we need to do approximately T^2 each of *hard* steps, the backward and forward iterations of the heterogeneous-agent model. Now we need to do only do approximately T each of the same hard steps (replacing forward iterations by the very similar expectation iterations), a factor-of-T improvement! There's still T^2 work being done in the final two steps, but this is very easy work that can be implemented on a computer.

[5] To be precise, as we can see in step 2, we actually do $T-2$ expectation iterations