Stephen Debug Notes

Here, I'm using the files `dd_reiter_jm2.m` and `dd_fsys_jm2.m` as the starting point. The new files are in `dd_fsys_sjt.m` and `dd_reiter_sjt.m`. Below, the red stuff indicates problems (I was writing them as I went), and the blue stuff indicates successes. I got a little bit happy with the colors...

- The file hunt.m, called from the spline interpolation functions, seemed to be missing from the Reiter-implement directory. It's available from any of the Reiter example directories.
- After evaluation of the code, there seem to be issues with the following entries:
  - V^{ND}: The LHS, which is V_{t-1}^{ND}, is fine and matches the SS code's values for V^{ND}. However, comparison of the SS code output and the RHS reveals that there are non-zero entries in the RHS at every second point, while most of those entries are zero on the LHS. The reason is simple. Given the indexng scheme (first z, then b_{-1}, then DS), every second entry is associated with $DS = 1$, i.e. with the "defaulted" state. In equilibrium, you only reach this point and are performing the optimization in V^{ND} when you *previously* defaulted and are still being punished. But then you must have $b_{-1} = 0$. Therefore, the SS code puts zeros for all b_{-1} indexes other than the one for b_{-1}. In the MATLAB file, this is called `b0ind`. I therefore edited line 164 in `dd_fsys_sjt.m` to only compute V^{ND} at the 0 bond holdings point. Not fully fixed, but almost all zeros now.
  - V: The LHS, which is V_{t-1}, is fine and matches the SS code's values for V. However, the RHS has 0's at the `znum` points corresponding to $(z, DS = 1, b_{-1} = 0), \quad \forall z$. However, in equilbrium these points are potentially reached, so we need to compute them. Making a change to line 298 in `dd_fsys_sjt.m` fixed the issue. Fully fixed.
  - *Dist*: There was a type on line 332 of `dd_fsys_jm2.m`, which labelled a variable `bvalt` when the intention was `bval`. This is in the block that linearly interpolates the policies and default thresholds. When I removed the `t`, I got some other errors. These led me to determine that there were problems in lines 340-341 in `dd_fsys_jm2.m`. In particular, the number stored in `bind` should be checked against `bnum`, not `bdensenum`. Ok, this did not fix the issue, suggesting that perhaps there is underlying problem in the policies used for the transitions. Therefore, I went to the block that did the re-optimization of policies (starting at line 155 of `dd_fsys_jm2.m`), and I made the following changes:
    - Line 218: added 1e-4 to nlb, to match Fortran code
    - Line 222: added 1e-4 to nub, to match Fortran code
    - Lines 219, 223, 233: added parentheses around $W_{-1}z$
    - Line 244: determined output of internal bisection on labor to be $\frac{n_a+n_b}{2}$, to match Fortran code

* Lines 277-281: fixed what may have been a meaningful optimization issue. The code in `dd_fsys_jm2.m` didn't order the re-assignment of b and d correctly. Because of this, it would sometimes re-assign b and then assign c to the *re-assigned* value of b. This would lead to a collapse in the interval, so I flipped the ordering. At the end of this, the entries in `bvalmat` in the MATLAB code lined up close (but not perfect, see next bullet for potential cause), with the values in bNDpol.txt from the Fortran code.
* Line 293: Noted that on this line, instead of storing b_d, the Fortran code averages the endpoints and re-evaluates the RHS of the Bellman eqn. Because the Fortran code uses a function for this, it was implementable easily. The MATLAB code stores d and uses the already computed RHS. Since the code converges when the interval is small, this shouldn't be a big issue, but it will lead to numerical differences. I've left it for now.
* Lines 301 and 304: Because I made a change to line 298, I now needed to flip the ordering of lines 301 and 304 and make a tweak to line 304, in order to avoid contaminating the stored `xivalmat` (which is just for non-defaulted states) with the value of xival in a defaulted state. The Fortran code has a slightly different organization of the conditional statements, but they now produce the same values of the policies and thresholds b and ξ^*. First, note that after these changes, the V^{ND} block is fully fixed now, with zeros up to small numerical errors which are likely due to the d vs endpoint averaging issue I highlighted above.
* Line 344: Need to get interpolation weights based on `b0`, not based on `bdense0`. At this point, the dense values of policies and thresholds `bval2mat` and `xival2mat` match the values of their Fortran counterparts.
* Line 365: Only pushforward weight to the defaulted point next period when have zero assets this period (added a condition to the if statement).
* Line 374: changed condition to only push stuff forward if not defaulted. Note that this covers all the cases since there is not any weight on the not-defaulted states with positive assets.
* Lines 374-410: There were several major errors here. Instead of looking like Lines 699-743 in the Fortran code, which compute a probability of default and then push forward defaulted and non-defaulted weight accordingly, this code was treating all the weight as if there is 0 probability of default in the next period. Also, the code was incorrectly interpolating the next period bond policy. Finally, the code was reading in the value of the interpolated policy incorrectly (see `bcountt` rather than `bmin1count` on line 377.) Ok, after these were fixed (see the new codes in lines 356-435 of the file `dd_fsys_sjt.m`, the distributions match up to within rounding error. The rounding errors is likely due to the different algorithm endpiont from the GSS noted above.

– δ_t: This default rate was fixed already when the micro-level stuff was fixed.

– L: Total labor supply. The LHS of this matches the value from the SS code. The RHS did not match. This made me investigate the underlying labor supply functions, and I discovered a big error in the Fortran code which computed ndensepol. I wasn't updating b correctly when re-optimizing dense labor policies. I've fixed it in line 575 of the code. However, this means that we'll need to iterate to find a new SS. After these changes, the underlying aggregated values of borrower labor supply, borrower consumption, borrower

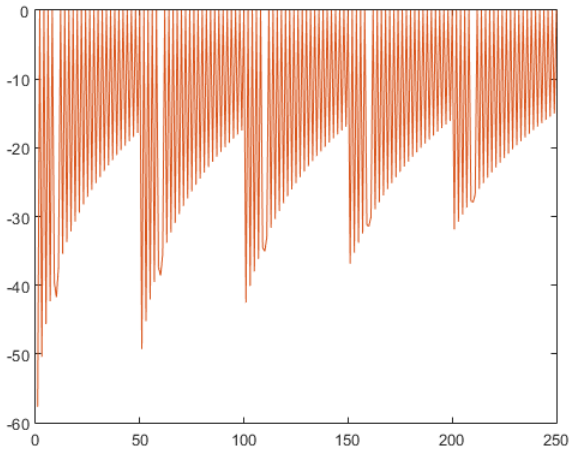

Figure 1: LHS and RHS of V^{ND}

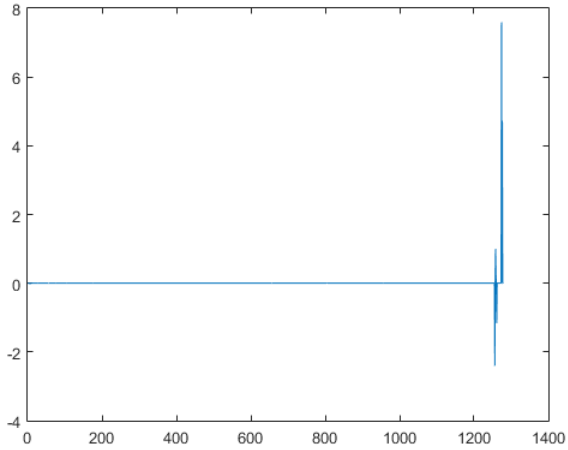

Figure 2: The Entire $F(X_{-1}, X, \varepsilon, \eta)$ Function

debt, and borrower default all match up. We can now be confident in the micro-level optimization section of the MATLAB code.

And at the end, there are some pictures, for your happiness.