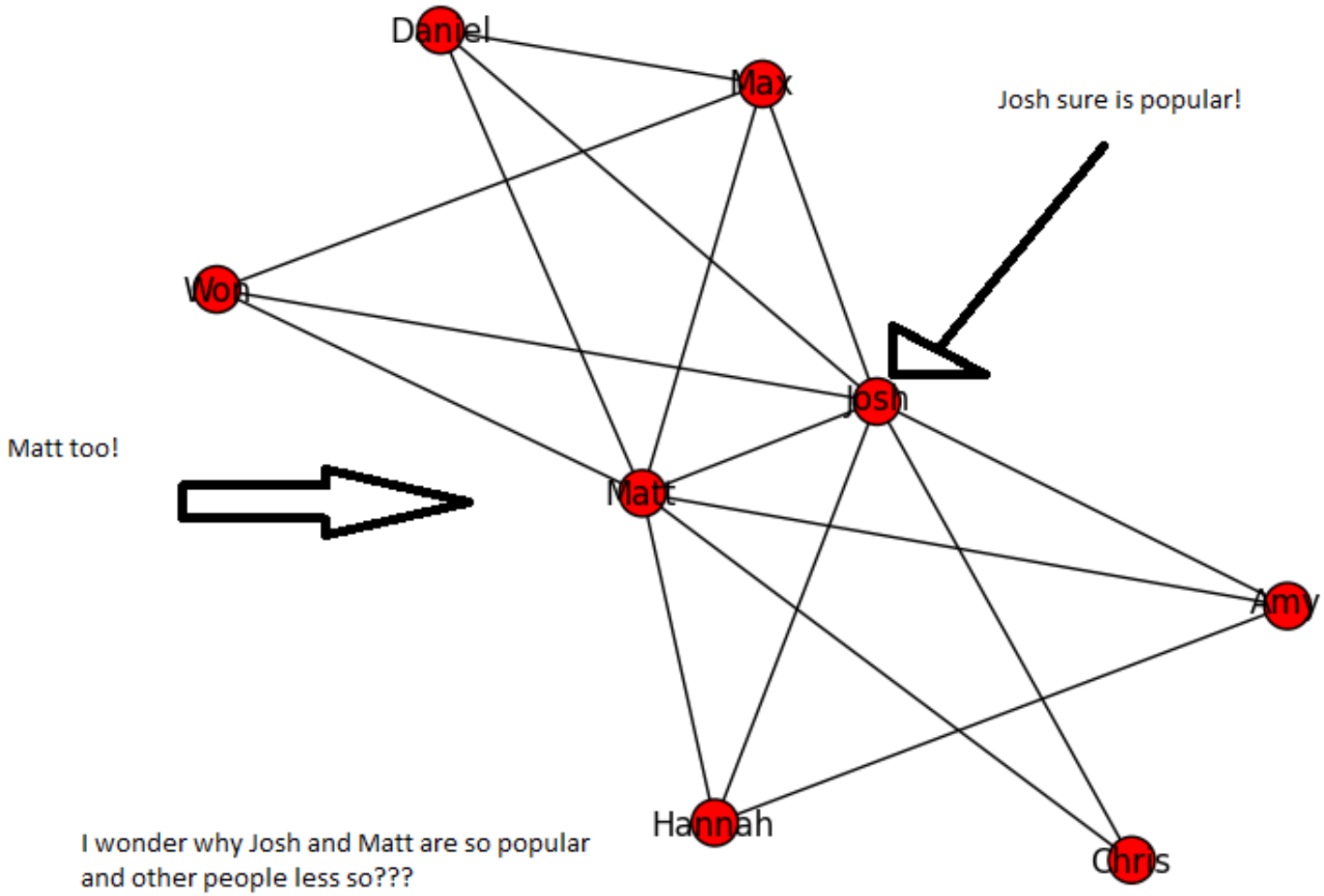
# PS 632: Solving Endogeneous Networks Computationally

Max Gallop

Department of Political Science
Duke University

Version 1.0

# Approximately 70% of Social Network Presentations[1]: Featuring this Class



# A Particular Issue with SNA

- A bunch of networks we care about capture processes which are purposive/strategic/of interest to us. So ideally we'd like to be able to explain how these networks come about.
- Do this by providing a model explaining how forming or not forming links in a network effects an actor's utility, and show what networks are equilibria for a given utility function.
- HOWEVER: Finding an equilibrium, for a network larger than like 4 actors is REALLY hard. So we want to write Python code to do it for us.

# Some Functions We Need to Do Stuff

- To generate networks use NetworkX python package: in particular, the shortest path algorithm.
- Wrote function to import list of nodes with relevant characteristics from CSV.
- NetworkX has functions that generate random graphs, but doesn't have one to add nodes in some random way to existing graph. Wrote one that mkaes a Bernoulli graph (p probability of any link) olut of a given list of nodes.

# Utility

- Function with graph as input to calculate utilities. Nodes gain utility based on the characteristics of neighbors discounted by the *delta* times shortest path length.
- ISSUE: Needs to rewrite function to change utility function.
- Current solution to issue: Use global values for the discount rate and cost of forming a link, to make changing the utility function somewhat simpler.
- Ideal Solution: A function that prompts the user to answer a number of questions and then based on those answers gives the utility function.

# Pairwise Stability

First requirement for an equilibrium network: no two actors want to jointly add a link.

- Loop through nodes in network, then loop again. (Can limit to upper triangle for speed: h = graph nodes. Do for i in h: h = h[1:], for j in h and).
- Make $G^{'}$: G but with a link between i and j. If $U(G^{'})[i], U(G^{'})[j] > U(G)[i], U(G)[j]$, then add the link in G.

# Nash Equilibrium

No actor wants to unilaterally subtract any number of their links.

- Loop through nodes.
- Set current graph as "winner".
- For each node, create a powerset (set of all possible subsets) of existing links.
- Loop through powerset, creating graph$'$, where i's links are the relevant subset. If $U[i](\text{graph}') > U[i](\text{winner})$, then winner $\leftarrow$ graph$'$.
- Delete all of graph's edges, add all edges from winner.

# Find Pairwise Nash

- Does this with nested while loops (I know)
- While loop, running the adding code from 2 slides ago until it doesn't change anything. (G.edges() $= G^{'}$.edges()).
- While loop running the cutting code from 1 slide ago, until it doesn't change anything.
- If you can get through both loops with no changes, you've found a pairwise Nash.

# Issues

- Solved Issue: I'm terrible at not reusing variable names during loops, and also at making sure that the return code is at the correct indent level so all loops conclude.
- Solved Issue: Graphs are objects so when you try $G' = G$ and then change $G'$ it changes G too. Solved through use of copy.deepcopy.
- Outstanding Issue: Code does not scale up well. Tried it on a network of countries in the world, and it takes about 6000 seconds to run the adding code. Possible solutions: make loops more efficient (already took a step by halving the adding loop), replace some loops with recursion (possible issue with recursion depth on sufficiently large networks), or redo in faster language.
- Multiple equilibria: this code finds only one!
- Improving Cycles: This code can get stuck.

# Issues: Multiple Equilibria

- Identify different equilibria by using different starting graphs.
- Algorithm: Generate a series of Bernoulli networks with different probabilities.
- Run Find Pairwise Nash: If this is the first time this was done, store the dictionary representation and all utilities as the first item in a list. If not, compare it to the items in the list. If any key (node) has differnt values (edges) the dictionaries differ. If minimum difference $>0$, we have new equilibria, add it to the list.

# Issues: Improving Cycles

- In some graphs, a series of moves that each are individually rational will lead us in a cycle and the algorithm will not find a Pairwise Stable Network.
- We solve this by tracking the steps the graph has taken on its path in a list. If ever the Adding and Cutting algorithm leads to the same graph already in a list–BUT NOT THE MOST RECENT ONE– we are in an improving cycle.
- Solution: Pick a new random network and rerun the code recursively (or change the order and run the code recursively.)

# Miscellanius Functionality: Network Comparison

Want abilility to calculate similarity between networks to determine if observed networks are similar to theoretical ones.

- Degree Distribution.
  1. Proportion of networks that have 0, 1, 2...N-1 links.
  2. Make dictionary. Loop through nodes, get their degree, add one to the value in dictionary with key = degree (or make it if none).
- Triadic Census Distribution.
  1. Number of triads in network with 0,1,2,3 links.
  2. Make dictionary. Loop through graph to find those triads with appropriate number of links, add to dictionary, then divide all values by 6 due to duplication. (Ideally fix code to avoid said duplication).
- Shared Partner Distribution.
  1. Number of neighbors with 0, 1..N-2 shared partners.
  2. Make dictionary. Loop through all dyads with shortest path == 1. Loop through neighbors of i, adding to some counter if they are neighbors of j, then increment value of dictionary with key = counter.
- Geodesic Distribution.
  1. Dyads with shortest path 1, 2..N-1.
  2. Make dictionary. Loop through all dyads. If their shortest path = "None", x = "Nones", else x = shortest path. Then...

# Testing

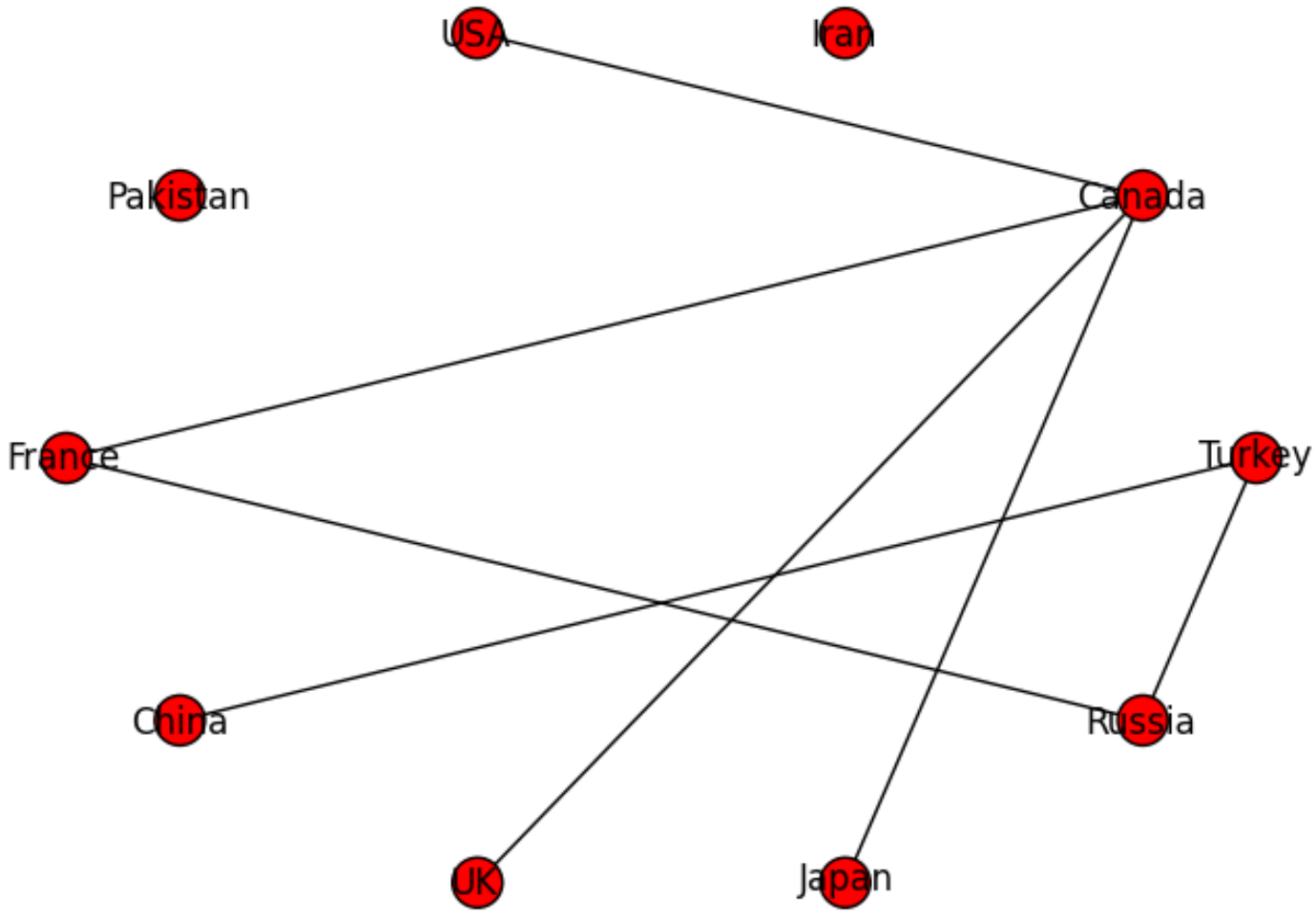
Tested the extreme cases.

- Checked if when c $= 0$ and $\delta < 1$, the edges are the same as in the full graph.
- Checked if when c $= 1$ and $\delta < 1$, the edges are the same as in the empty graph.
- Checked if when c $= 0$ and $\delta < 1$, the equilibrium is unique.
- Checked if when c $= 1$ and $\delta < 1$, the equilibrium is unique.

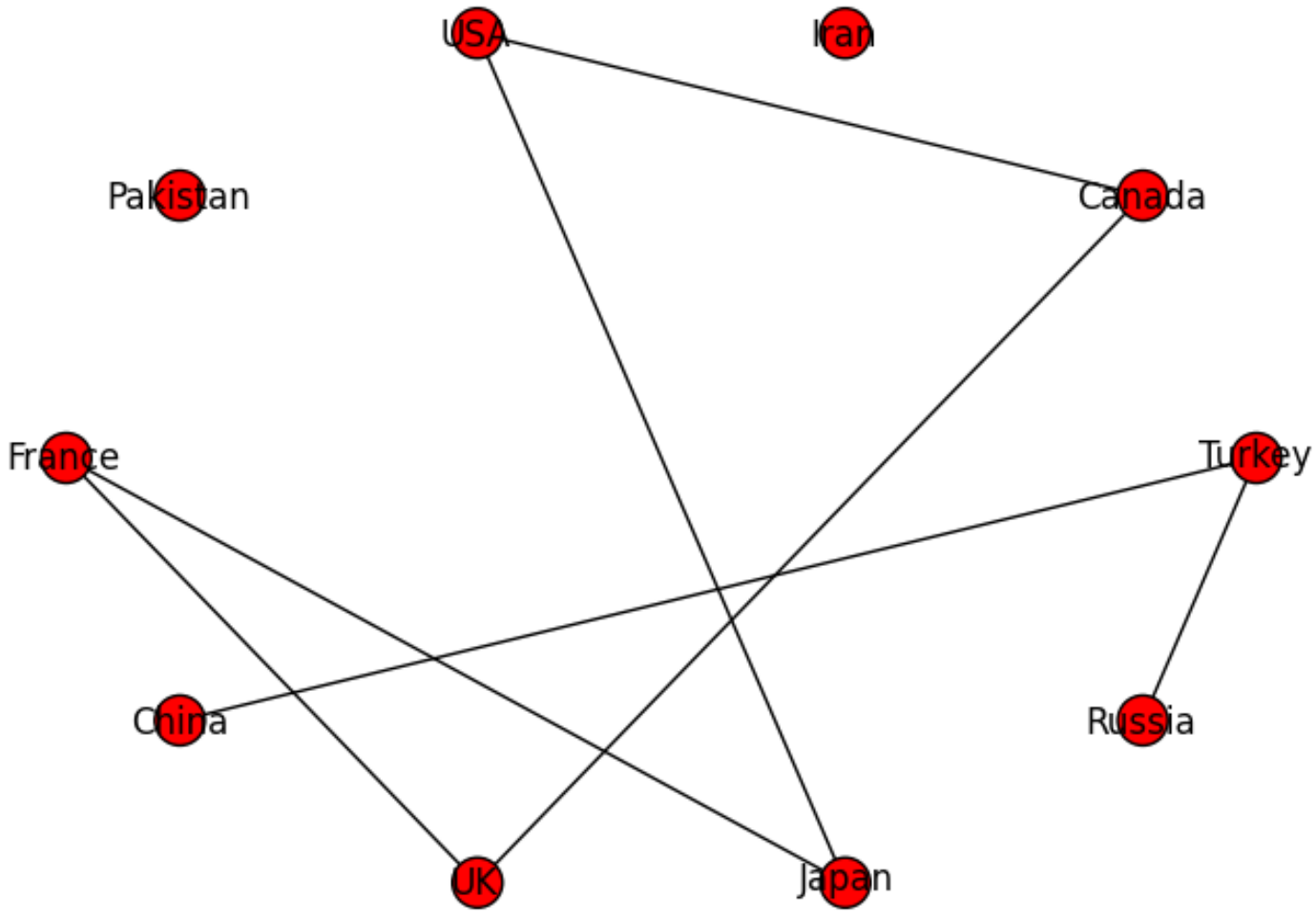
# Useful Code

- Generally created a number of useful functions for dealing with networks in networkx, particularly in terms of network comparison and a few easy ways to generate networks (from csv and at random).
- I'd also claim the endogeneous network framework is useful more generally, and would acknowledge Anna Schultz, who helped fix some of the bugs in the code, think of ways to get at multiple equilibria (though I went in a different direction) because she needed the code for work she was presenting at MPSA/this week.
- However, for this to be portable to other players work need to fix two things:
  1. MOAR SPEED. Seriously, my other computer is in the middle of a run that is over 150000 seconds as I write these slides.
  2. Less need for users to write their own utility function.

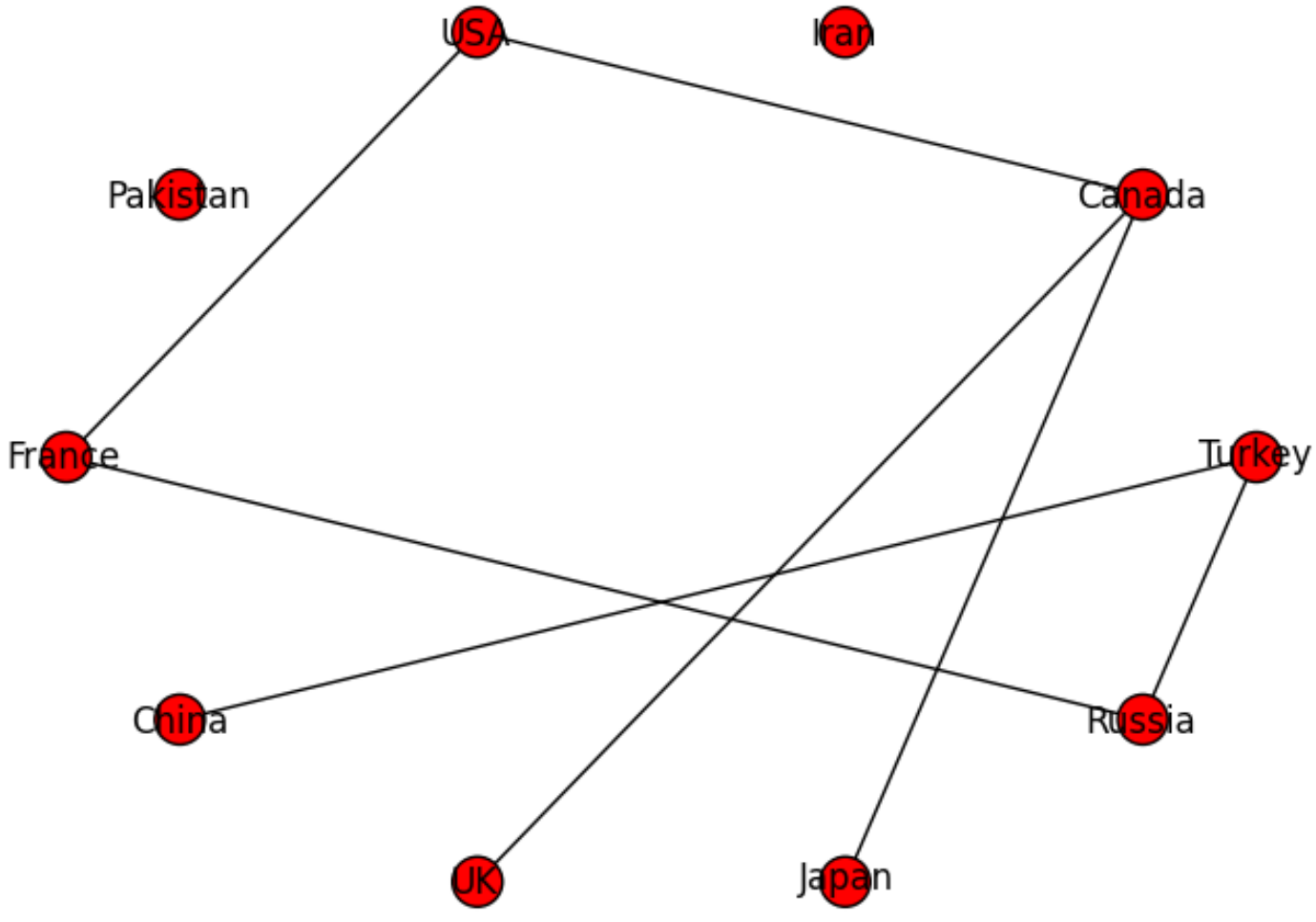
# Some Pictures: $\delta = .5, c = .1$: Equilibria1



# More Pictures: $\delta = .5, c = .1$: Equilibria2



# Even More: $\delta = .5, c = .1$: Equilibria3



# And so it goes: $\delta = .5, c = .1$: Equilibria4