Take-Home Final Exam

36-462, Spring 2009

Due 12 noon, 12 May 2009

Choose 1 of the following problems. Present as complete a solution as you can to all parts. Submit your answers electronically as PDFs, along with your code in machine-readable form. (R is preferred but if you would rather use something else, ask me.)

You are free to read whatever you like in connection with this.

1. FLUCTUATING FEEDBACK A linear feedback system has a one-dimensional state variable X. There is a resting or set-point value of X, which without loss of generality we can take to be zero. When the state is perturbed away from zero, some fraction a_t of the perturbation is fed back into the state at the next time-point:

$$X_{t+1} = a_t X_t + U_t$$

where U_t is the new input to the system. (This is also called a first-order autoregressive or AR(1) model.) In a linear, time-invariant system, the feedback ratio is constant, $a_t = a \; \forall t$. This problem is about what happens when a_t fluctuates.

   (a) Suppose that $a_t = a$ and $U_t = u$ for all t, where $|a| < 1$, while $X_0 = 0$. Show that $X_t \rightarrow u\frac{1}{1-a}$ at $t \rightarrow \infty$. The ratio $1/(1-a)$ is called the **gain** of the feedback loop.

   (b) Suppose U_t has an IID Gaussian distribution with mean 0 and variance σ^2. Show that there is a ν^2 such that if X_t has a $\mathcal{N}(0, \nu^2)$ distribution, then so does X_{t+1}, and find ν^2 in terms of a and σ^2. (This is the invariant distribution of the Markov process.)

   (c) Suppose that the feedback ratio is a continuous random variable, A, with density f. Find a formula for the density of the gain $G = 1/(1-A)$. Show that it has a power-law right tail, i.e., that the density approach $g^{-\alpha}$ as $g \rightarrow \infty$, and find α. Does it depend on the original density f?

   (d) Simulate 10^4 random draws from $A \sim \mathcal{N}(0.6, 0.01)$, and plot the corresponding empirical survival function of G. Does it agree with your answer in the previous part?

   (e) Generate a series U_t of length 10^4, where each U_t is an IID draw from a $\mathcal{N}(0, 1)$ distribution. Set $X_0 = 0$ and find X_{10^4} for each value of A you generated in the previous part. Describe the distribution of X_{10^4}, particularly the tails. Compare it to the distribution of which you would expect from part (2) under the average gain.

   (f) In the previous part, the feedback factor was set randomly at the beginning of each time series, but stayed fixed thereafter. Suppose that it follows its own AR(1) model,

$$A_{t+1} = c + bA_t + V_t$$

   where V_t is IID Gaussian noise with mean zero and variance μ^2, independent of U_t. If $b = 0.9999$, find the values of c and μ^2 which will give A an invariant $\mathcal{N}(0.6, 0.01)$ distribution.

   (g) Write a function to simulate the time-evolution of A_t under these rules. It should take an initial value A_0, values of b, c and μ^2, and a time-length T, and return the sequence $A_1, A_2, \ldots A_T$. Verify that your code works by checking the distribution of A_{9001} through A_{10000}

with $A_0 = 0.6$, $T = 10^4$ and the other parameters as in the previous part.

(h) Write a function which will simulate the time-evolution of X_t given an initial value X_0, a sequence of A_t values, σ^2, and a time-length T, again returning $X_1, X_2, \ldots X_T$. Verify that it works by checking the long-run distribution of X_t when $A_t = 0.6$ for all t, $\sigma^2 = 1$ and $X_0 = 0$.

(i) Combine the functions from the previous two parts to give a single function which takes as inputs X_0, A_0, b, c, σ^2 and μ^2, and returns both the A and X sequences. Using the same parameter values as in those parts, and 10^4 simulation runs of length $T = 10^4$, find the long-run distribution of X_T.

2. IMITATION WITH MEMORY Recall from the lectures on heavy-tailed distributions that in basic Yule-Simon model works as follows. Objects are divided into piles. At each round t, we add one object. With probability ρ the object is added to a new pile. With probability $1 - \rho$, the object is added to an existing pile; the probability of being added to a pile of size k is proportional to k. The Yule-Simon distribution is the limiting distribution of pile sizes, $p(k) = \alpha B(k, \alpha + 1)$ where $\alpha = 1/(1 - \rho)$ and B is the beta function (`beta` in R); The CDF is $F(k) = 1 - kB(k, \alpha + 1)$. This has power-law tails, since $p(k) = O(k^{-\alpha+1})$ for large k.

   (a) Write a function to simulate one step of the Yule-Simon process. It should take as its input ρ and a vector of pile sizes, and return a new vector of pile sizes. Check that the sum of all pile sizes only increases by 1, and that when an existing pile is added to, the probability of adding to piles of size k is in fact proportional to k.

   (b) Write a function to generate a random value from simulating the Yule-Simon process, starting with a single pile of size 1 and continuing to a large, specified size. Check that the distribution of these random values is close to that of the Yule-Simon distribution as given above. (Do this for at least two values of ρ.)

   (c) Modify your simulation so it keeps track of how long it has been since each pile was added to. (One way to do this is to modify the function from the first part so it takes two vectors, one of sizes and one of times; when a pile is added to or created, its time is set to 1, otherwise all times increase by 1.) At the end of a long simulation run, what is the correlation between the size rank of piles and how recently they were modified? (Hint: use `rank`!) Does it depend on ρ?

   (d) Modify the simulation so that it takes an extra parameter, m: only piles which have been added to in the last m time-steps can be added to this time-step. (If all existing piles are too old, pick one at random.) Include a special value of m (i.e., not a positive integer!) which turns off this memory restriction; you'll know it's working because it will give the same behavior as the previous model.

   (e) Simulate the model for 10^4 time-steps with $\rho = 0.01$ and $m = 1, 2, 10, 100$ and ∞. Using the methods discussed in class, which of the resulting distributions could plausibly be power laws?

3. JANET This problem is about re-implementing the JANET algorithm in Foulkes's 1959 paper. For simplicity, we'll stick to a binary, $\{0, 1\}$ alphabet.

   (a) Read the paper. (It's on the course website.)

   (b) Write a function to count the number of times a given binary word w appears in a binary sequence x. The function should take w and x as inputs and return the count. Include overlapping instance of w (so if $w = 00$ and $x = 000$, the count is 2). *Hint:* look at the old homework.

   (c) Write a function to simulate the 7-state process in the paper. This should take as its input a number of output symbols T, and return a vector of 0s and 1s generated by the machine. Check that the simulator is working properly by comparing the probability of producing a 1 in each state to the actual frequencies of 1s in the simulation output when T is large.

   (d) Write a function to test for whether states need to be split. It should take as inputs w, the word defining the state; x, the complete data sequence; and α_1, a significance level in $(0, 1)$. Use a χ^2 test to check whether the distribution of the next symbol conditional on $0w$ is significantly different from the distribution conditional on w. Explain why you do not also have to test the distribution conditional on $1w$.

   (e) Write a function testing for whether two states need to be amalgamated, with significance level α_2.

   (f) Write a function to implement Foulkes's algorithm. It should take as input the data sequence x and a significance level α, and return a list which gives the words defining the states and the probability of producing a 1 in each state.

   (g) Apply your algorithm to data from

      i. A Bernoulli process with $p = 0.5$

      ii. A Bernoulli process with $p = 0.1$

      iii. The first-order Markov chain from homework 3

      iv. Foulkes's example process

      In each case, use $\alpha_1 = 0.10$ and $\alpha_2 = 0.05$, and report how large T has to be before the algorithm recovers the correct set of states.

4. El Farol Once upon a time in Santa Fe, there were $N = 50$ people, who only had two things to do at night: stay at home and look at the stars, or go to the El Farol bar.[1] Staying home had a relative pay-off of 5. Going to El Farol was more fun that star-gazing (pay-off 10), unless more than $N/2 = 25$ people tried to go; then the bar go too crowded, the waiters forgot about drink orders, fights broke out, etc., and the over-all pay-off was 0. Everyone then would rather go to El Farol than stay home, unless more than 24 others are also going; then everyone would rather stay home.

   (a) An outcome is *Pareto optimal* when it cannot be changed without making at least one agent worse off[2]. Show that any given night's configuration of activities (bar-going/star-watching) is Pareto optimal if and only if exactly 25 people are at the bar.

   (b) One measure of the efficiency of the social system is how far the number of attendees departs from 25. Write a function which calculates the mean squared distance from 25 of a time-series of numbers, $\mathbf{E}\left[(X-25)^2\right]$.

   (c) A mindless randomized strategy is for every agent to flip a coin every night with probability $p = 0.5$ and go if the coin comes up heads. Calculate the inefficiency of this rule analytically. (*Hint:* express this in terms of the mean and variance of the appropriate binomial.)

   (d) Write a simulation of the mindless random strategy where each agent makes independent decisions. The simulation should take as input a number of time-steps T, and return the number of attendees at El Farol each night, $X_1, X_2, \ldots X_T$. Verify that the algorithm is working properly by checking the time-average of X and the inefficiency.

   (e) Write a simulation where agents learn by reinforcement, as follows. At time t, agent i has weight $w_i(0,t)$ for staying home and weight $w_i(1,t)$ for going to the bar. The agent's move is $Y_i(t)$, which is 1 with probability $p_i(t) = w_i(1,t)/(w_i(0,t)+w_i(1,t))$. When all agents have made their decisions, they receive the appropriate pay-off $f_i(t)$. Then agents update their weights:

   $$w_i(j,t+1) = \alpha w_i(j,t) + (1-\alpha) f_i(t) \mathbf{1}_{Y_i(t)=j}$$

   where $\alpha \in [0,1]$, i.e., they increment the weight of the move they made by a fraction of the pay-off they received, and otherwise shrink all weights towards zero. Initially, all agents give equal weights to the two moves, $w_i(0,0) = w_i(1,0) = J$.

   Your function should take as inputs the number of rounds T, the memory decay rate α, and the initial weights J. It should return

[1]This is a slight simplification of the original El Farol game due to Brian Arthur. El Farol is a real bar and restaurant in Santa Fe, and a very good one, though there *are* others. — More seriously, there are many social activities, like timing the stock market, where, all else being equal, it's better to be in the minority.

[2]This is the same Pareto as in power laws.

the sequence of the number of attendees, $X_1, X_2, \ldots X_T$, and the final relative probabilities of going to the bar for each agent (i.e. $\frac{w_i(1,T+1)}{w_i(0,T+1)+w_i(1,T+1)}$ for $i = 1, 2, \ldots N$).

Some steps towards making this work:

i. Write a function which takes an $N \times 2$ matrix of weights and returns the vector of length N which is the probability of picking action 1 for each agent. Check that it works by giving it the weight matrix $\left[\begin{array}{cc} 5 & 10 \\ 20 & 20 \\ 30 & 15cc \end{array}\right]$.

ii. Write a function which takes a vector of N probabilities and returns N independent random choices. (Hint: see `help(rbinom)`.)

iii. Write a function that takes a vector of N actions and returns the N corresponding pay-offs. Check that it gives the right pay-offs for all combinations of actions by 3 players.

iv. Write a function that updates the weights. It should take the old weight matrix, the vector of actions, the vector of pay-offs, and the decay-rate α, and return a new matrix of weights. Check that it correctly updates the weight matrix given earlier when $\alpha = 0.5$.

Putting these parts together in a loop should give the full simulation. (However, if you want to organize it differently, you can do so; just check that choices are being made and weights updated appropriately.)

As an over-all check, note that if $\alpha = 0$, there is no learning, and the behavior should be the same as the random-choice model.

When you think the model is working, run it for $T = 200$ time-steps and initial weights $J = 5$ with at least 4 different values of α. Report the time-course of average attendance, the inefficiency, and the variance of the final probabilities of attending as functions of α. Can you get more efficiency than in the IID scenario by adjusting α? What's the relationship (if any) between inefficiency and the variance in the probabilities?

(f) Modify your simulation so that agents update the weights of *both* actions with that time-step's pay-offs (i.e., they don't just update the weight of the action they made). What happens?

(g) Modify your simulation so that the new weights are just the sum of the old weights plus the payoffs, but the probability of taking action j is proportional to $e^{w_i(j,t)}$ rather than $w_i(j,t)$. What happens?

5. Networks from Games In this model, there are N agents, who decide whom to interact with in each time step. (This is not necessarily symmetric.) Agent i has weights w_{ij} for all other agents, and interacts with agent j with probability proportional to w_{ij}. Initially, $w_{ij} = 1$ for all i and j, except on the diagonal, $w_{ii} = 0$.

   (a) Write a function to take as input a weight matrix and return as output the matrix of interaction probabilities. Check it by hand on an example with $N = 3$.

   (b) Write a function to take as input the matrix of interaction probabilities and return the actions of each agent.

   (c) Write a function to update a weight matrix by adding each agent's current pay-offs to their weight for the agent they interacted with.

   (d) Write a simulation where each agent always receives a pay-off of 1 no matter what. What happens to the probability matrix for $N = 50$ and large T, say 10^3? (Find a way of displaying the weight matrix; you might consider `heatmap`, or making it into a network graph using the `statnet` package.)

   (e) Write a simulation where if i interacts with j, w_{ij} and w_{ji} are both increased by 1. how do the resulting probability matrices differ from those with asymmetric pay-offs?

   (f) Each agent now has one of two types, R and S. If their types match when they interact, they both get pay-off 1, otherwise they get pay-off 0. What happens to the interaction probabilities?

   (g) Change the pay-offs so that if two Ss interact, the both get pay-off 1; if two Rs interact, they get pay-off 0.75; and if an R and an S interact, the R gets 0.75 and the S gets 0. (That is S pays off better if you have a cooperative partner, but worse if you don't.) What happens to the interaction probabilities if half the agents are of each type?

   (h) Introduce imitation into the model: each time step, with probability q, each agent will switch to the type with the highest average pay-off from the last time step. What happens to the interaction probabilities and the distribution of types with $q = 0.1$? With $q = 0.01$? (You will need to do multiple runs to answer these questions.)