Problem Set #1

MACS 30250, Dr. Clindaniel
Due Wenesday, April 29 at 5:00pm

1. **Clocking parallelism.** In this exercise, you will document the degree to which a parallel processing approach speeds up the compute time of your entire program as you increase the number of CPU's. Let z_t be an individual's health index value in period t, where each period is a week. A high value of z_t signifies good health and a low value of z_t signifies bad health. Assume you are studying the properties of the health of individuals over time z_t, which follows an AR(1) process (persistence) and has a normally distributed idiosyncratic error,

$$z_t = \rho z_{t-1} + (1-\rho)\mu + \varepsilon_t \quad \text{where} \quad \varepsilon_t \sim N(0,\sigma)$$

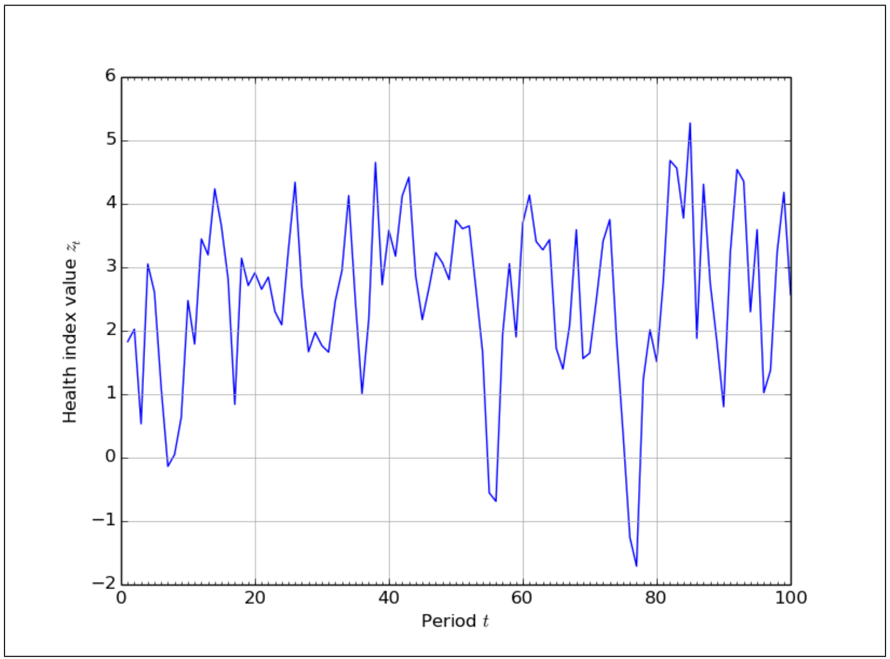
where $\rho \in [-0.95, 0.95]$ represents the degree of persistence in the process, μ is the long-run average value of the time series, and σ is the standard deviation of the normally distributed idiosyncratic errors. Because you don't know what health shocks ε_t an individual will get, one way to study this is to simulate many lifetimes of health shocks and the resulting outcomes on the individual's health index z_t. The following code is how you would simulate 1,000 lifetimes of health shocks, each lifetime being 4,160 weeks (80 years times 52 weeks per year) given the particular parameter values of $\rho = 0.5$, $\mu = 3$, $\sigma = 1$, and $z_0 = \mu$.

```
import numpy as np
import scipy.stats as sts

# Set model parameters
rho = 0.85
mu = 3.0
sigma = 0.8
z_0 = mu

# Set simulation parameters, draw all idiosyncratic random shocks,
# and create empty containers
S = 1000 # Set the number of lives to simulate
T = int(4160) # Set the number of periods for each simulation
np.random.seed(25)
eps_mat = sts.norm.rvs(loc=0, scale=sigma, size=(T, S))
z_mat = np.zeros((T, S))
z_mat[0, :] = z_0

for s_ind in range(S):
    z_tm1 = z_0
    for t_ind in range(T):
        e_t = eps_mat[t_ind, s_ind]
```

```
            z_t = rho * z_tm1 + (1 - rho) * mu + e_t
            z_mat[t_ind, s_ind] = z_t
            z_tm1 = z_t
```

The first 500 periods of the first simulated z_t series in `z_mat` is shown in Figure 1. Note that for this particular simulated lifetime, the individual starts at a value below average $z_1 < \mu$ and the first period in which her health dips below 0 is $t = 8$.

Figure 1: Single time path of z_t for first 100 periods



(a) Log onto RCC Midway and time 100 different runs of the simulation above—i.e., 100 different runs of 1,000 simulations each of the time series of 4,160 periods. Do one run in serial using only one core as in the code above (1,000 simulations on one core), the second run using two cores (500 simulations per core), the third run using three cores (333 simulations per core), and the last run using 100 processors (10 simulations per core). Use the `mpi4py` library to implement your parallelism. Plot the computation time for the 1,000 simulations on the y-axis against the number of cores for the particular run on the x-axis.

(b) Why is the speedup not linear? That is, why is the 100-core run not 100 times faster than the single-core run?

2. **Embarrassingly parallel processing (grid search).** Assume that really bad things happen to people whose health index falls to or below zero $z_t \leq 0$. And assume that you are concerned about individuals who are already sick or whose current health level is below average $z_0 = \mu - 3\sigma$. Assume that you can formulate a health policy that changed the persistence ρ of an individual's health.

   (a) Use grid search (embarassingly) parallel processing to find the optimal value of ρ that maximizes the number of average number of periods across each of the 1,000 simulations that it takes to first reach a negative health index value $z_t \leq 0$. Perform your grid search over 200 values of ρ equally spaced between -0.95 and 0.95 using the RCC Midway cluster with 100 cores and using `mpi4py` to implement your parallel processing of two values of ρ per core. Report the computation time to find the optimal ρ. [TIPS: Make sure to draw your $T \times S$ matrix of epsilon shocks ε_t only once at the beginning of this code before any parallel processing is implemented. Then pass that same matrix of shocks into every different value of ρ.]

   (b) Plot the average periods to the first negative or zero health value $z_t \leq 0$ for each value of ρ (plot ρ values on the x-axis and average periods to first negative health value on the y-axis.

   (c) Report the optimal persistence ρ and the corresponding average periods to negative health.

3. **More sophisticated parallelism (minimizer).** In this exercise, we will perform the same optimization from Exercise 2, but in a more efficient way (should be faster) by embedding the parallel processing inside of a larger serial program. Instead of grid search, we will use a minimizer to find the optimal ρ.

   (a) Use `scipy.opimize.minimize()` to find the optimal ρ. Use an initial guess of $\rho = 0.1$. For each guess of ρ that the minimizer makes, use 100 cores on RCC Midway to do the 1,000 simulated time series of z_t (i.e., 10 simulations of the time series per core). Report the computation time to find the optimal ρ.

   (b) Report the optimal persistence ρ and the corresponding average periods to negative health.

   (c) How does the computation time for this Exercise compare to the computation time from Exercise 2? Why is this method faster or slower than the grid search method in Exercise 2?