# Fish Population Dynamics

A controlled fish population was simulated using a computer model. Specifically, that of a fish farm having zero immigration and emigration. The only growth factor was fish births. Due to the controlled (and idealized) environment, factors such as weather, seasons, la nina, etc. will be ignored. The only reduction factor in the model was fish deaths. Another simplification in the model is that floating points will be used to represent all quanitities. This is appropriate since the conclusions of the model are most appropriate when applied to millions of fish.

This population model can be described by the following equation

$$N_{t+1} = N_t + dN/dt$$
$$= N_t + r\,N_t\,(1 - N_t/K) - E_t\,v\,N_t$$

and the effort is updated according to:

$$E_{t+1} = E_t + dE/dt$$
$$= E_t + a\,(P\,E_t\,v\,N_t - c\,E_t)$$

where:

- $N_t$ = number of fish at time t
- $N_{t+1}$ = is the number of fish one timestep later.:
- r = birth rate
- K = carrying capacity
- $E_t$ = effort during timestep t
- v = fishability
- a = reinvestment rate
- P = price / fish
- c = cost / effort

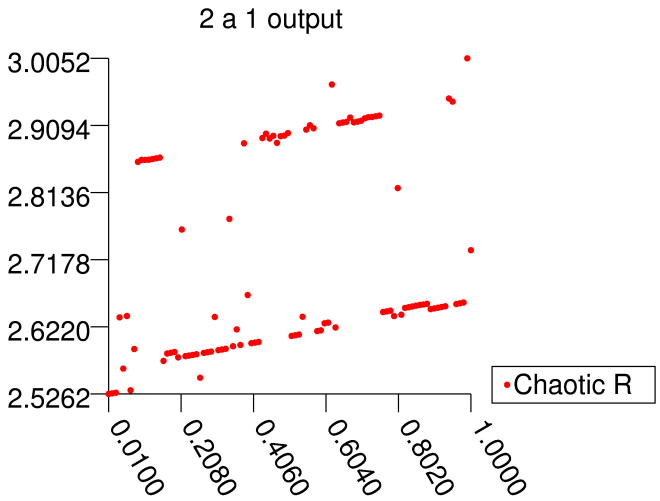
It is expected that some combination of correct values for the parameters above will produce a stable fish population. Intuition says that increasing birth rate will cause effort to decrease, or profits to rise.

## 2. Running the model

2.a - 2.b. Regions of chaos were sought in order to avoid them by finding stable zones. Using the following initial conditions the population becomes chaotic at r=2.5783.:

N=50, K=100, E=.2, v=.1, a=.3, c=4.0, P=1.0, dE/dt = dP/dt = 0

In fact, keeping E constant produces a very strange relationship between the chaotic point of r and E, as seen below.:

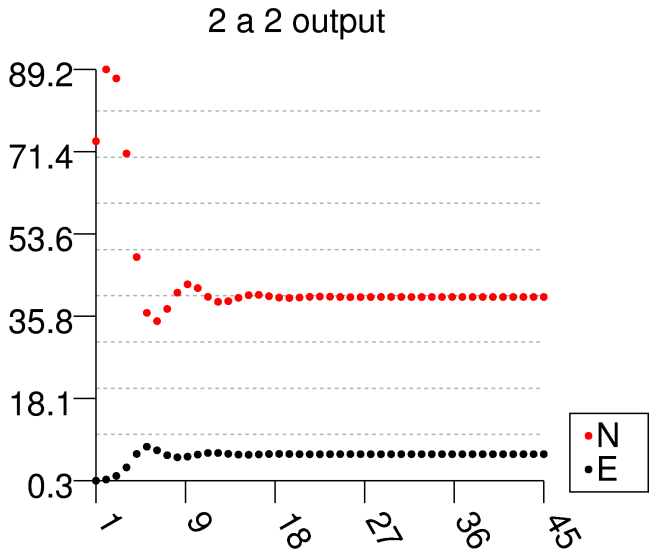



For some values of E there is a striking linearity. For others there is some linearity with other values. It is an interesting graph.

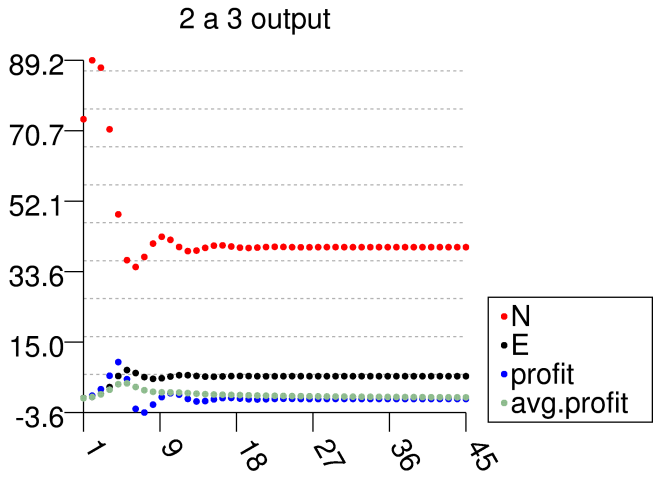
Next E was allow to variate according to the equation from part 1.:

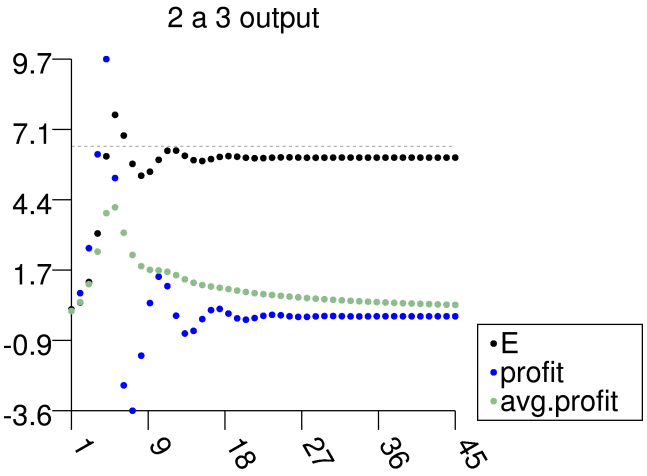
$E_{t+1} = E_t + a (P E_t v N_t - c E_t)$

Below is Effort and and number of fish plotted against time.




The stability of the system looks quite appealing until one also plots net profit (closeup below right).:



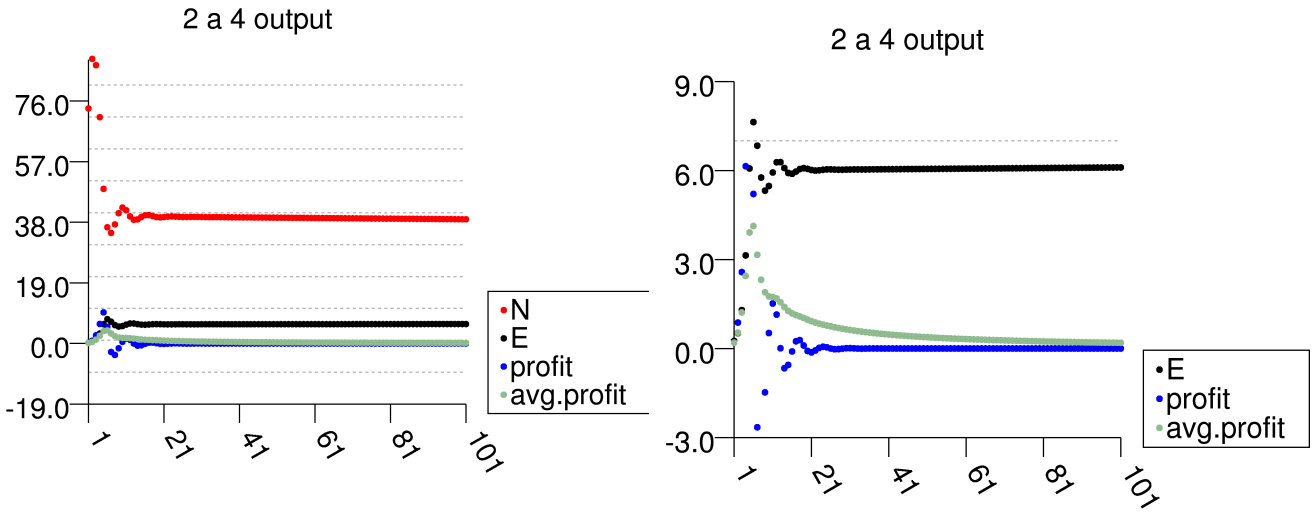




Since we are attempting to model a macrocosmic fishing community, let us also allow the price P per fish to be affected by the yield. It stands to reason that higher yields cause the price to drop, and that there is a certain size to the fish market (the number of fish people would like to buy), and that the price has a certain sensitivity to the difference between the market and the actual yield. So, the change in the price P is:

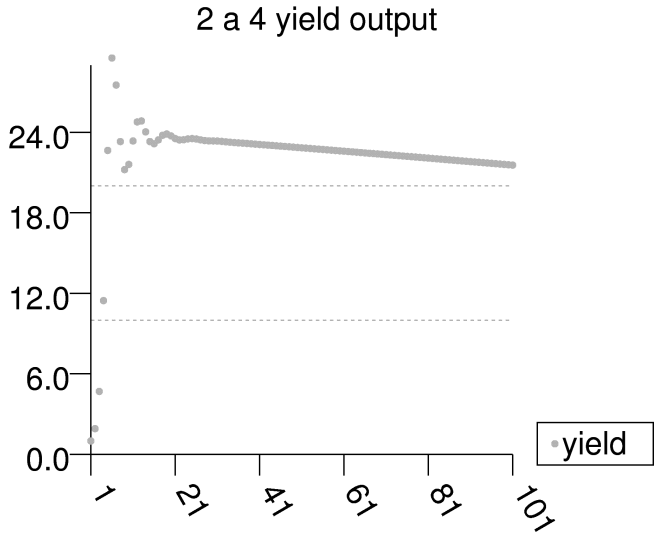
dP/dt = sensitivity (market - yield)

This equation implies that when the yield is above the market, the price goes down. Keeping all other factors the same, let us apply a very small sensitivity to the market: .00001. Also, let

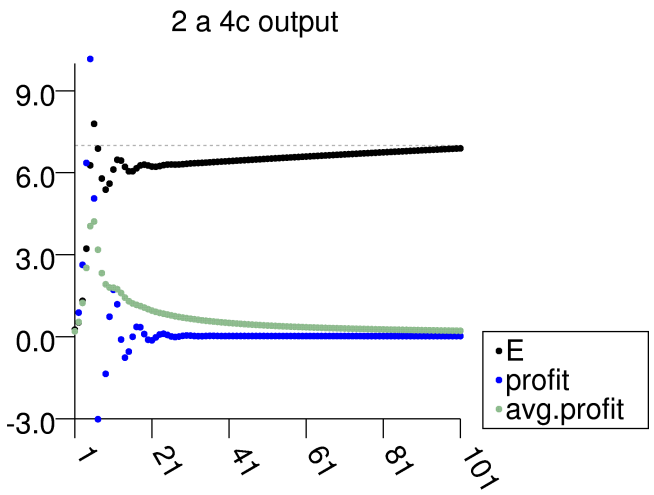
us assume a best-case scenario (for fisherman), that the market = 50, the amount of fish we are starting with.  The same variables as before are now as plotted below.:




A graph of the yield during this time period shows that it doesn't ever approach the market.:




So, now let us boost the sensitivity a little, to .0001.  Theoretically, this should make life better for the fisherman.  Results shown below.:



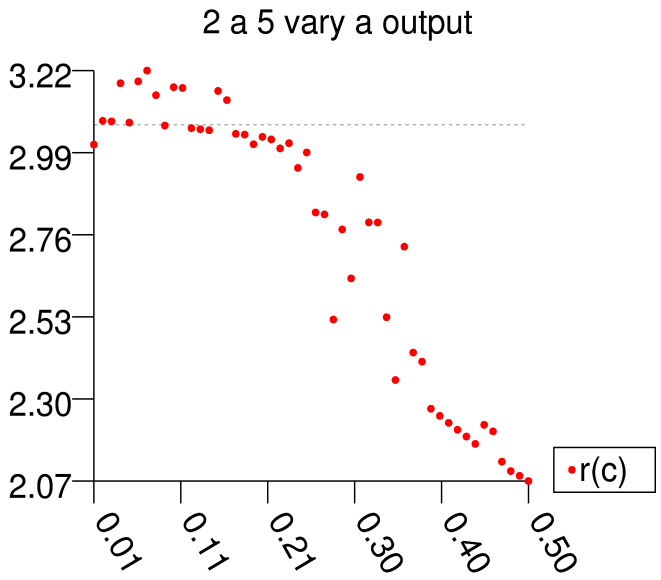

This situation looks pretty grim. The fisherman live with worse and worse profits while putting in more effort.

Putting profits aside for the moment, let us look briefly at values of our parameters and whether or not they cause a system to find equilibrium or stay chaotic. Starting with the default values and the system as just described, with N, E, and P all variating, r remains stable until it reaches 2.640154. At 2.640155 it leaves the zone of bifurcation/cycling, and enters a chaotic zone, where the N at each timestep is unpredictable. Let this point be called $r_c$. $r_c$ was found for various parameter combinations by finding the highest stable value, using smaller and smaller steps until a highest stable value was found within a small distance from a known chaotic value.

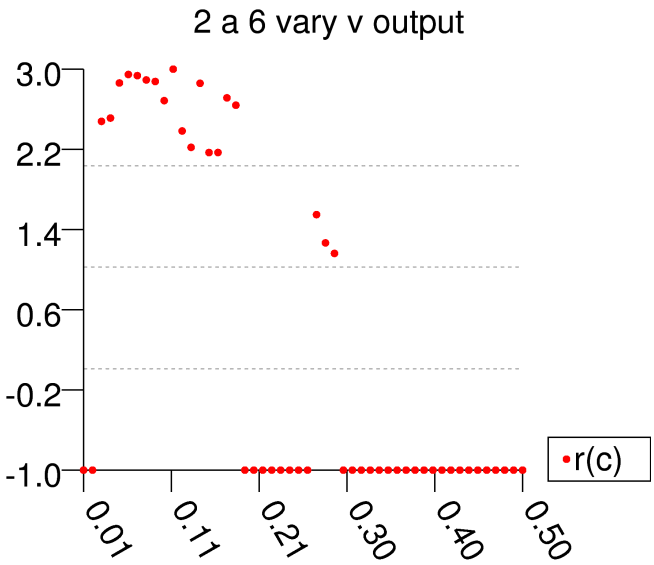
A system was classified as chaotic under the following conditions: at least 100 timesteps had past, and the current tuple (N, E, P) was not roughly equivalent to any previous tuple.

The most obvious parameter to modulate is reinvestment. Below is $r_c$ plotted against reinvestment.

The graph above shows different values of a (reinvestment), and the corresponding $r_c$ on the y-axis. If the fish in question has a high r, such as 3, the reinvestment should not be set above .2 or so.

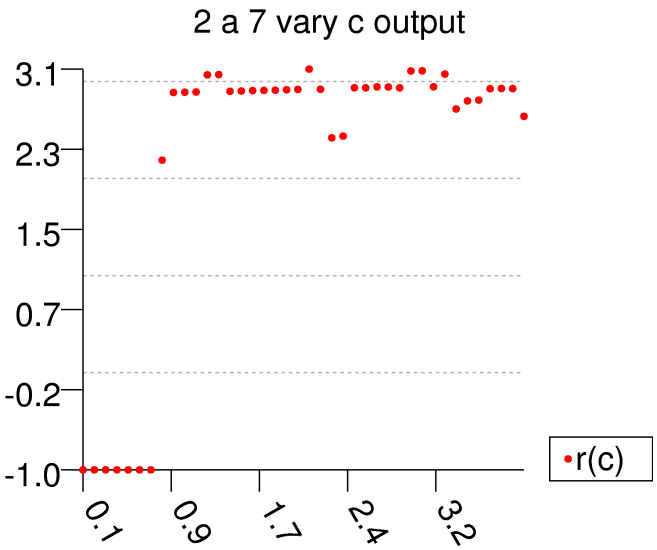



Let us say that technology improvements allow the fish to become more fishable, increasing v. Below is plotted $r_c$ against v.




The above values for $r_c$ were found with all other parameters set to defaults. As can be seen, some of the values for $r_c$ are -1. These are merely placeholders to show that there was no value of v for which a stable r could be found (leaving all other parameters at default values.

Now let us say the government decides to subsidise fishermen by paying for some of their expenses. This brings down the cost per effort, c. Below, $r_c$ is plotted against c.

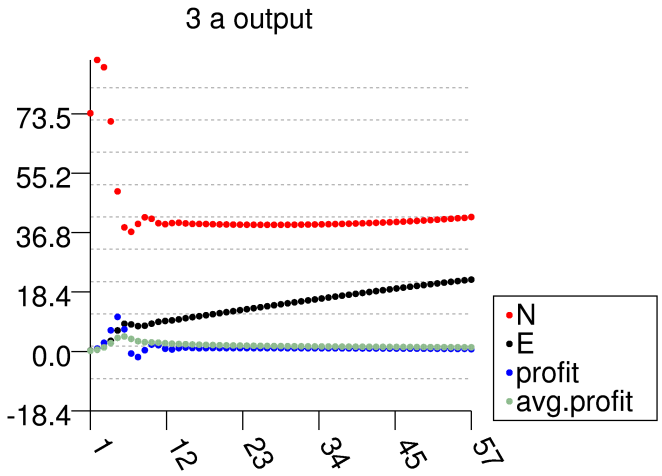



As long as the fishermen incur a cost of about .9, there appears to be plenty of stable parameter space.

## 3. Fix the model

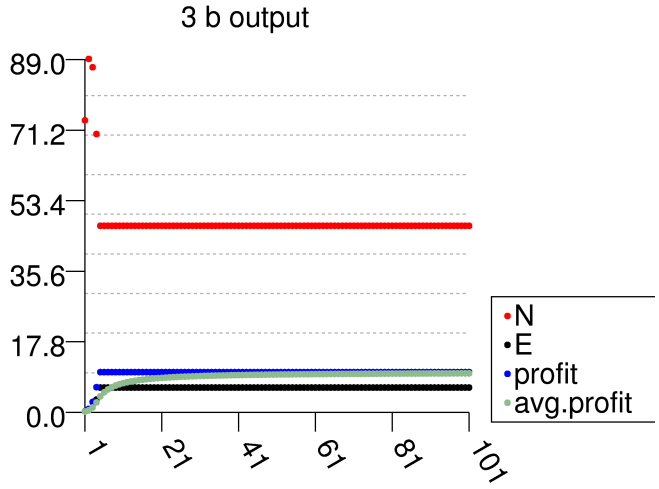
A proposed solution to the long term problem of fishing profitability and fish population stability is controlled genetic breeding. Assuming a fish farm, fish can be kept in numerous relatively small groups. The groups that have the highest yield could be left unculled. All others would be sold on the fish market. This kind of targeted fish selling could increase profit by increasing the number of fish sold without increasing effort.

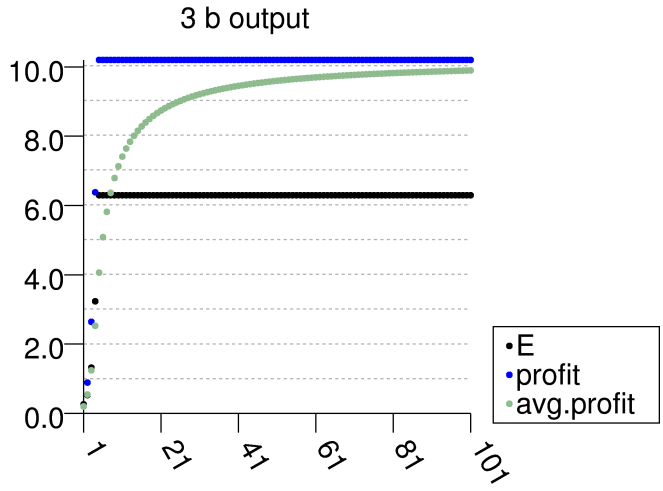
Let us now say that r changes with time, always increasing. The purpose of this paper is not to investigate genetic modifications, so no support is given here for the viability of this plan from a biological standpoint, only a mathematical one. Let us say that r increases slowly, but linearly. Let dr/dt = .05. Below is a graph showing the viability of this system.




As can be seen, the solution is still no good. Profit hovers around zero while the effort rises continutally. At least the fish population is still stable.

Instead of using something as fancy as genetic modification, why not use the old standby that state governments all across this great nation already use -- the fishing limit. If we set a max yield to 10 fish per timestep, all parameters are stable, including the profit.




Although wild fish situations are much less controlled, the indication of this simple model is that a correctly-set limit on the total yield of the fishing community will produce a stable situation with reasonable profits.

BIBLIOGRAPHY

[1] Wikipedia article

http://en.wikipedia.org/wiki/Population_dynamics_of_fisheries

*Explanation of fish population models*

CODE Appendix

(Plotting code not included as it was already included in the atomic simulation project)

```
##  POPULATION CLASS  ##

class Population:

    def __init__(self, N0=50, r=1.0, K=100, E0=.2, v=.1, a=.3, c=4.0, P0=1.0):
        self.t = 0
        self.N = self.N0 = N0     # number of fish
        self.r = self.r0 = r      # birth rate
        self.K = K                # carrying capacity
        self.E = self.E0 = E0     # effort
        self.v = v                # fishability
        self.a = a                # reinvestment
        self.c = c                # cost/effort
        self.P = self.P0 = P0     # price/fish

        self.initial = [self.N0, self.E0, self.P0]
        self.sensitivity = .0001    # price sensitivity to yield
        self.market = 50          # market for fish

        # also in self.reset()
        self.p = 0                # net profit
        self.y = 0                # yield
        self.past = []
        self.present = self.initial
        self.maxYield = sys.maxint

        self.takeSnapshot()


    def next(self):
        self.t += 1
        if self.p > self.maxYield:
            return

        self.E += self.dEdt()
        self.N += self.dNdt()
        self.P += self.dPdt()

        if self.N < 0:    # just in case
            self.N = 0.0
        if self.E < 0:    # just in case
            self.E = 0.0
        if self.P < 0:    # just in case
            self.P = 0.0
        self.takeSnapshot()
        #self.printout()


    def dNdt(self):
        return self.r * self.N * (1 - self.N/self.K) - self.E * self.v * self.N


    def dEdt(self):
        self.y = self.E * self.v * self.N
        self.p = self.P * self.y - self.c * self.E
        return self.a * self.p


    def dPdt(self):
        return self.sensitivity * (self.market - self.y)


    def printout(self):
        print '%d\t%f\t%f\t%f'% (self.t, self.N, self.E, self.P)
```

```
    def takeSnapshot(self):
        self.past.append(self.present)
        self.present = [self.N, self.E, self.P]


    def reset(self):
        self.N, self.E, self.P = self.initial
        self.t = 0
        self.past = []
        self.preset = self.initial
        self.p = 0                # net profit
        self.y = 0                # yield

    def stable(self):
        i = -1
        for p in self.past:
            # lv({'p':p, 'now':[self.N, self.E, self.P]})
            i += 1
            if closeEnough([self.N, self.E, self.P], p):
                return True
        return False

    def chaotic(self):
        if self.t > 100:
            if not self.stable():
                return True
        return False

    def run(self, func=None):
        while not self.stable() or self.t < 40:
            #self.printout()
            self.next()
            if func != None:
                func(self)
            # self.printout()
            # choice = sys.stdin.readline()[:-1]
            # if choice == 'q':
            #    exit()
            if self.t > 100:
                return False    # must be chaotic
        return True

    def chaoticR(self, func=None):
        self.reset()
        self.r = self.r0

        if not self.run(func):
            # print "Must start in stable zone"
            return -1

        # iterate upward until highest stable r is found
        g = 1
        stable = self.r
        last = self.r
        while abs(g) > 1e-5:
            # lv({'r':self.r, 'g':g, 'E':E})
            self.reset()
            self.r += g
            if self.run(func):
                stable = self.r
                last = self.r
            else:
                g = .9*g
                self.r = last
        return stable


##  FUNCTIONS  ##

def closeEnough(S, T, prec=.01):
    e = .00000000001   # just to prevent division by 0
    if S[0] == None or T[0] == None:
```

```
        return False
    for i in range(len(S)):
        if 2*abs(S[i]-T[i] +e)/(S[i]+T[i] +e) > prec:
            return False
    return True


def print_NE(self):
    print '%d\t%f\t%f'% (self.t, self.N, self.E)


def print_NEpy(self):
    global avg_p, obs
    avg_p += self.p
    obs += 1
    print '%d\t%f\t%f\t%f\t%f\t%f'% (self.t, self.N, self.E, self.p, avg_p/obs, self.y)


def gm(self):
    global avg_p, obs
    self.r += 0 #0.05
    avg_p += self.p
    obs += 1
    print '%d\t%f\t%f\t%f\t%f\t%f'% (self.t, self.N, self.E, self.p, avg_p/obs, self.y)



##  MAIN  ##

def usage():
    print '''
Usage:
  > python fish.py <N0> <r> <K> <E> <v> <a> <c> <P>

'''
if __name__ == '__main__':


    if len(sys.argv) >= 2:

        if sys.argv[1] == '2.a.2':
            fish = Population()
            func = print_NE
            fish.run(func)

        elif sys.argv[1] == '2.a.3':
            avg_p = obs = 0
            fish = Population()
            func = print_NEpy
            fish.run(func)

        elif sys.argv[1] == '2.a.4':
            avg_p = obs = 0
            fish = Population()
            func = print_NEpy
            fish.run(func)


        # look for chaotic regions
        elif sys.argv[1] == '2.a.5':
            for a in range(50):
                a = .01 + .01*a
                avg_p = obs = 0
                fish = Population()
                fish.a = a
                print '%f\t%f'% (a, fish.chaoticR())

        elif sys.argv[1] == '2.a.6':
            for v in range(50):
                v = .01 + .01*v
                avg_p = obs = 0
                fish = Population()
```

```
            fish.v = v
            print '%f\t%f'% (v, fish.chaoticR())

    elif sys.argv[1] == '2.a.7':
        for c in range(40):
            c = .1 + .1*c
            avg_p = obs = 0
            fish = Population()
            fish.c = c
            print '%f\t%f'% (c, fish.chaoticR())

    elif sys.argv[1] == '3.a':
        avg_p = obs = 0
        fish = Population()
        func = gm
        fish.run(func)

    elif sys.argv[1] == '3.b':
        avg_p = obs = 0
        fish = Population()
        fish.maxYield = 10
        func = gm
        fish.run(func)


    exit()

##  Default action
N0 = 50.0
fish = None
if len(sys.argv) > 2:
    r = float(sys.argv[2])
    K = float(sys.argv[3])
    E0 = float(sys.argv[4])
    v = float(sys.argv[5])
    a = float(sys.argv[6])
    c = float(sys.argv[7])
    P0 = float(sys.argv[8])
    fish = Population(N0, r, K, E0, v, a, c, P0)
else:
    fish = Population(N0)

#print fish.run()

#for E in range(100):
if True:
    print '%f'% (fish.chaoticR())
    fish.reset()
```