Lukas Gesing, Patrick Kaster
MA-INF 4201 - Artificial Life
Exercise Sheet 6

Assignment 37

We have:

$$\begin{aligned}\log(N(t)) &= \log\left(a\frac{1}{s^\alpha}\right)\\ &= \log a + \log\left(\frac{1}{s^\alpha}\right)\\ &= \log(a) + \log(1) - \log\left(s^\alpha\right)\\ &= \log(a) + \log(1) - \alpha\cdot\log\left(s\right)\end{aligned}$$

So, apart from the logarithmic scaling, the expression is linear.

■

Assignment 38

$$\begin{aligned}N(k\cdot s) &= a\cdot\frac{1}{(k\cdot s)^\alpha}\\ &= a\cdot\frac{1}{k^\alpha}\cdot\frac{1}{s^\alpha}\\ &= \frac{1}{k^\alpha}\left(a\cdot\frac{1}{s^\alpha}\right)\end{aligned}$$

So, we have a proportionality constant of $\frac{1}{k^\alpha}$. All power-law functions are scale invariant. ■

Assignment 39

The primary problem is to the determine the corpus of the language/languages. Valid reserved words from the programming languages that compromise the corpus have to be distinguished from variable, etc., which might be terms from natural languages.

So one first has to know which programming languages to examine, in order to to derive keywords from their BNFs.

Subsequently the data might be gathered from large online software repositories.

■

Assignment 40

no guaranteed global minimum The gradient decent method will converge to a local minimum, but no guarantee can be given that the minimum found is a global one. Thus, the method might not give the optimal solution.

result depends on initial conditions Because of the above, the result will depend heavily on the initialization of the method, i.e. where on the surface of the objective function the decent is started.

not applicable to all objective functions In order to do a decent and calculate a gradient, the objective function has to be continuous and differentiable, which makes the method non applicable to a whole range of functions.

■

Assignment 41

Both methods have in common, that the objective function doesn't have to be continuous nor differentiable. Instead both function pick a random position in search space.

The difference lies in the picking of a new random point in search space. While *random search* samples the new point from the surface of a hypersphere that surrounds the current position, given a certain radius, the *random optimization* draws a normally distributed vector and adds it to the current position. ■

Assignment 42

For a starting concentration of $N_0 = 1(100\%)$, we seek a factor λ, such that the function

$$N(t) = N_0 e^{-\lambda t}$$

yields a value of $0.1(10\%)$ after $t = 42$.

To find the solution for λ we solve the ODE

$$\begin{aligned}\frac{dN}{dt} &= -\lambda N \Leftrightarrow \\ \frac{dN}{N} &= -\lambda dt\end{aligned}$$

After integrating both sites we obtain:

$$\begin{aligned}\ln\left(\frac{N}{N_0}\right) &= -\lambda t \Rightarrow \\ -\lambda &= \frac{\ln\left(\frac{N}{N_0}\right)}{t}\end{aligned}$$

This gives the solution:

$$\begin{aligned}-\lambda &= \frac{\ln(0.1)}{42} \\ &= -0.054823454595096\end{aligned}$$

As we easily verify:

$$\begin{aligned}N(42) &= 1 \cdot e^{-0.054823454595096 \cdot 42} \\ &= 0.1\end{aligned}$$

■

Assignment 43

Without *evaporation* the solution of the algorithm is static, i.e. a solution that is found once remains static, so that a dynamical change of the environment can't be accounted for.

Suppose a food source is found via an Ant-algorithm with *evaporation*. After n ants have transported back the food to the colony, the food source is depleted. Initially ants will follow the previous path to the food source, without finding more food. Thus, they won't return to the colony on the original path. The *pheromone* trail will get weaker, until no ants returning from the colony will follow it anymore.

These ants will start searching for new food sources in random directions, increasing the probability to find a new food source.

Without *evaporation* of the *pheromone* trail the dynamic environmental change of food depletion doesn't lead to a new search phase of the algorithm. ■

PA-C

The task has been implemented in python script `PA-C.py` and has options `-X` and `-Y` for setting the starting conditions, as well as parameters `-A` to `-H`. Option `-n` limits the number of iterations, option `-m` gives an index position starting the mean calculation from. Parameter `-S` determines the system of equations to calculate. It defaults to 0 for the first system and 1 for the second system. Options `-G`, `-H` are ignored for system 0.

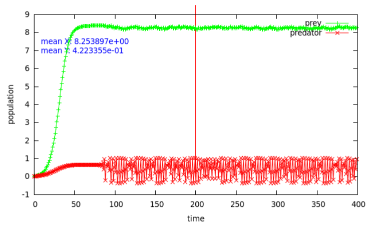
GNUPlot is called from within the script after the calculation is done.

For the first equation set we used the following set of parameters:
./PA-C.py -X 0.03 -Y 0.02 -A 0.1 -B 0.1 -C 0.11 -D 0.02 -E -0.02 -F -2.2 -n 400 -m 200

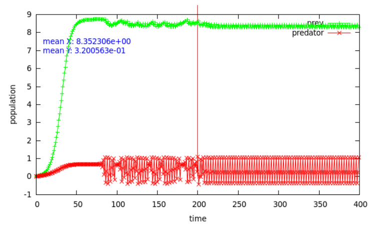
For increasing parameter a we observe that the mean grows, because variance increases, because oscillation becomes increasingly chaotic.

For the second equation set we used the following set of parameters:
./PA-C.py -X 0.03 -Y 0.02 -A 0.16 -B 0.1 -C 0.11 -D 0.02 -E -0.02 -F -2.2 -G 0.01 -H 0.01 -n 400 -m 200 -S 1

The major difference we observe is, that the system is coupled. Once the predator population begins oscillation, the prey population will follow shortly. Also the oscillation pattern is likewise.



(a) output for first predator/prey system



(b) output for second (coupled) predator/prey system

■