1 Stochastic Hopfield network

```
#= This script computes the order parameter using a asynchronous stochastic update, =#
#=   it answers the third question "Stochastic Hopfield network" in homework 1 =#

using Distributions

function main()
  #= initalizing paramers =#
   = 2
  N = 200
  P = 7
  T = 2e5
  order_parameter = 0
  iterations = 100

  #= Compute the order parameter 100 times and compute the average =#
  for it in 1:iterations
    println(it)
    order_parameter += compute_order_parameter(, N, P, T)
  end
  order_parameter /= iterations

  #= print the results  =#
  println(order_parameter)
end

#= A function that computes the weights for a hopfield network using Hebb's rule =#
#= Inputs the patterns and a boolean that says wheter the diagonal should be zero or not =#
function hebbs_rule(X, zero_dig)
  P = length(X)
  N = length(X[1])
  W = zeros(Int, N, N)
  for it in 1:N
    for jt in 1:N
      for pt in 1:P
        W[it,jt] += X[pt][it] * X[pt][jt]
      end
    end
  end
  W /= N

  # if we want the diagonal to equal zero we need this extra loop
  if zero_dig
    for it in 1:N
      W[it,it] = 0
    end
  end

  return W
end

#= Generates a pattern of length N, where each element is = 1 with p=1/2 and -1 otherwise =#
function generate_pattern(N)
  sample([-1, 1], N)
end

#= Generates a list of P patterns and returns it =#
function generate_patterns(N, P)
  #= Initalize a empty list and append(push!) each pattern to it =#
  X = []
```

```
  for it in 1:P
    push!(X, generate_pattern(N))
  end

  return X
end

#= This function performs a stochastig asynchronous update and returns the value  =#
#=   for that pixle =#
#= The inputs are the weight matrix, the patterns we want to update, and the index =#
#=   in that pattern and  the noise parameter =#
function stoch_async_update(W, S, i, )
  N = length(S)
  local_field = 0
  for j in 1:N
    local_field += W[i, j] * S[j]
  end
  if rand() < p(local_field, )
    return 1
  else
    return -1
  end
end

#= This function computes the probability for the new state being equal to 1, given =#
#=   the local field and noise parameter =#
function p(b, )
  return 1 / (1 + exp(-2**b))
end

#= This function computes the equality measure for two patterns, it equals 1 if the are =#
#=   the same and -1 if inverse =#
function equality_measure(S, p_t)
  count = 0
  for it in 1:length(S)
    count += S[it] * p_t[it]
  end
  return count / length(S)
end

function compute_order_parameter(, N, P, T)
  # Matrix for the random patterns
  X = generate_patterns(N, P)
  S = copy(X[1])

  # Get weights with zero diagonal
  zero_dig = true
  W = hebbs_rule(X, zero_dig)

  #= Perform a asynchronous update and compute the equality measure, =#
  #=   repeat for T times and lastly computing the average, which is to be =#
  #=   returned =#
  m = 0
  for it in 1:T
    ind::Int = it % 200 + 1
    S[ind] = stoch_async_update(W, S, ind, )
    m += equality_measure(S, X[1])
  end
  m /= T

  return m
```

```
end

main()
```