1 One-step error probability

```
#= This script answers the first question "One-step error probability" of Homework 1 =#

#= All the scrips are written in julia, since I didn't want to be bothered with the  =#
#=   slow programing languages such as python or matlab. I will do my best to make the =#
#=   scripts understandable for someone with no experience with Julia. The performance =#
#=   in this scrips are outstanding compared to the equivalent scripts that are written =#
#=   with python or matlab, it only takes 5 seconds to execute this script on my laptop =#

using Distributions # used for random sampling

#= The main script computes the error probability for different values for P =#
function main()
  P = [12,24,48,70,100,120]
  trials = 1e5
  for pt in P
    P_error(pt, trials)
  end
end

#= This function computes the error probability for a given value for P patterns =#
#=   and a given value for trials =#
function P_error(P, trials)
  N = 120

  #= generate the patterns X =#
  X = generate_patterns(N, P)

  #= Compute the weights W =#
  zero_dig = false
  W = hebbs_rule(X, zero_dig)

  total_errors = 0
  for t in 1:trials
    # sample a random pixle from a random pattern
    i = sample(1:N)
     = sample(1:P)

    # perform a asynchronous update
    S_i = async_update(W, X[], i)

    # check wheter the pixle changed or not
    if S_i != X[][i]
      total_errors += 1
    end
  end

  # compute P_error and print it
  total_errors /= trials
  print(total_errors, ", ")
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

#= 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

#= This function performs a asynchronous update on the pattern S at index i  =#
#=   using the weights W. Then returns the value, if the local field is equal =#
#=   to 0, then it returns 1 =#
function async_update(W, S, i)
  N = length(S)

  #= Compute the localfield =#
  local_field = 0
  for j in 1:N
    local_field += W[i, j] * S[j]
  end

  #= Return the sign of the local field =#
  s_i = sign(local_field)
  if s_i == 0
    return 1
  else
    return s_i
  end
end


main()
```