In [ ]:
```
using Graphs
using GraphPlot
using Plots
using LaTeXStrings
using Distributions
using StatsBase
using DataFrames
using HTTP
using JSON
```

Lucas Schmidt Ferreira de Araujo

Report 03

Exercice I

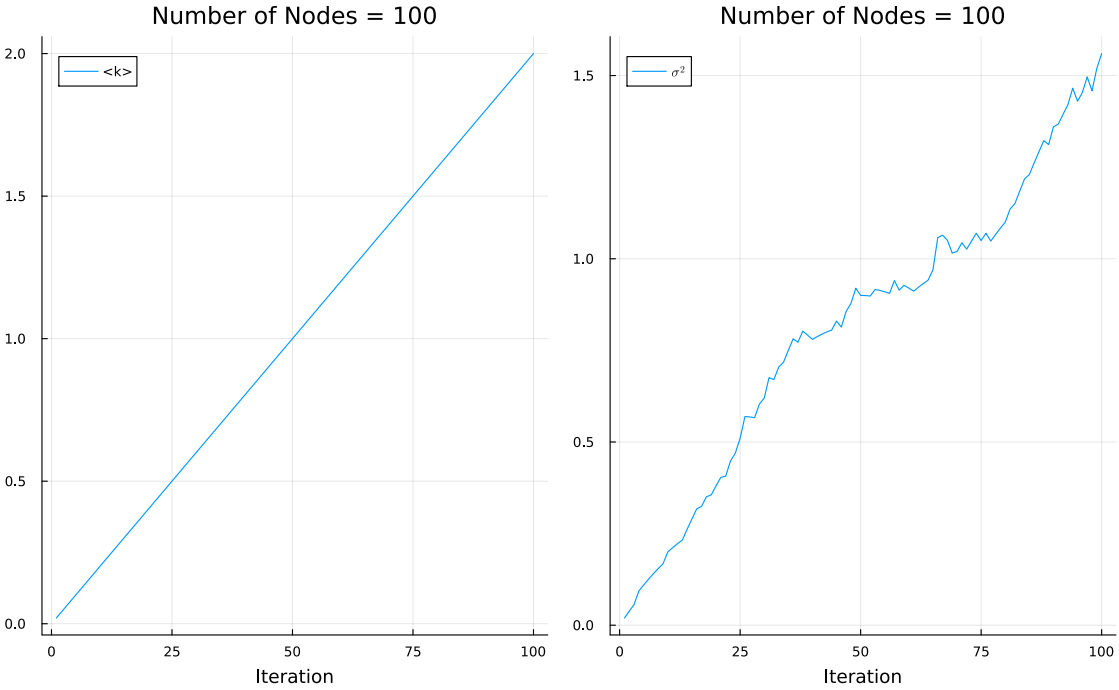
Random Graph - 1

Algorithm description

1. Start with a set of N nodes
2. While there are unchosen edges, randomly select one unchosen edge from the list of edges and then remove it from the list
3. Repeat until all edges are chosen.

In [ ]:
```
function choice(N,pool)
    chosed = rand(1:length(pool))
    rtn = pool[chosed]
    splice!(pool,chosed)
    return rtn
end

function Edges(N)
    pool = []
    elements = [q for q in 1:N]
    for q in elements
        for j in elements
            if !([j,q] in pool) && (q != j)
                push!(pool,[q,j])
            end
        end
    end
    return pool
end

function RandomGraph1(N,nit)
    pool = Edges(N)
```

```
    G = Graph(N)
    trunk = Vector{Graph}(undef,nit)
    for q in 1:nit
        pair = choice(N,pool)
        add_edge!(G,pair[1],pair[2])
        trunk[q] = copy(G)
    end
    return trunk
end

N = 100          ### Number of Nodes
nit = 100        ### Number of iterations
model = RandomGraph1(N,nit)
avrdgree = [sum( degree(model[q]) ) / N for q in 1:nit]
vardgree = [sum((degree(model[q]) .- avrdgree[q]).^2) / N for q in 1:nit]
plot1 = plot(avrdgree,label="<k>")
plot2 = plot(vardgree, label = L"\sigma^2")
plot(plot1,plot2,layout = (1,2),size=(1000,600),xlabel="Iteration",title=
```



Random Graph - 2

Algorithm description

1. Select a pair of nodes q and j
2. Generate a random number r between 0 and 1. If $r < \beta$, then add a link between q and j
3. Repeat (1) and (2) for all pairs of nodes

The expected average degree $< k >$ is given by

$$< k >= \beta(N-1)$$

and the probability distribution of k is given by the binomial distribution

$$P(k) = \binom{N-1}{k} \beta^k (1-\beta)^{N-1-k}$$

which in the limit $N \to \infty$ we have

$$< k >= \beta(N-1)$$

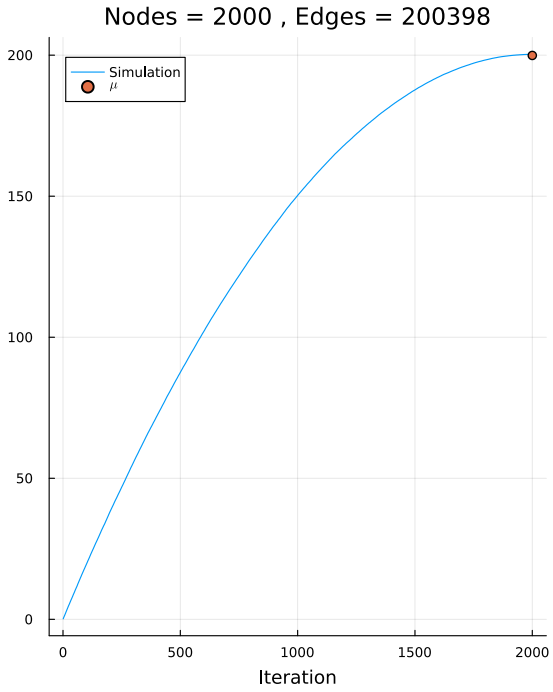
$$\sigma^2 = \beta(N-1)$$

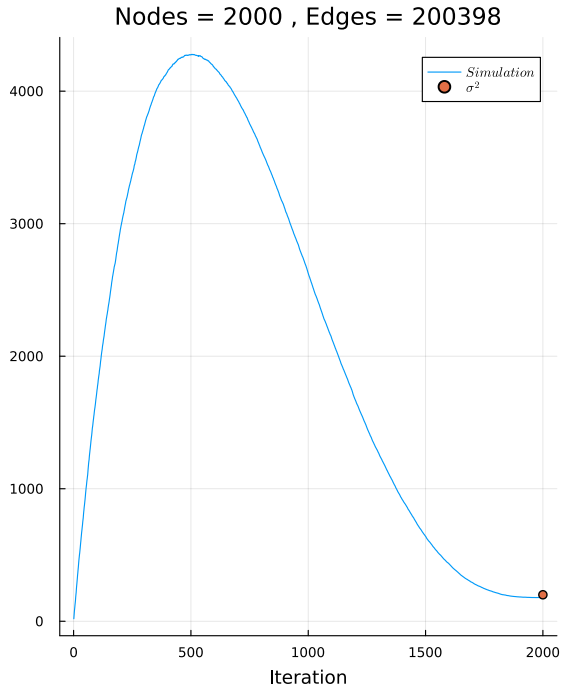
$$P(k) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\bar{x})^2}{2\sigma^2})$$

In [ ]:
```
N = 2000          ### Number of Nodes
β = .1           ### Probability of connection

μ = β*(N-1)
σ2 = μ

G = Graph(N)
avrdgree = zeros(N)
vardgree = zeros(N)
for q in 1:N
    for j in q:N
        if(rand() < β && !has_edge(G,q,j))
            add_edge!(G,q,j)
        end
    end
    avrdgree[q] = sum(degree(G)) / N
    vardgree[q] = sum((degree(G) .- avrdgree[q]).^2) / N
end

println("μ = $(μ)")
println("σ2 = $(σ2)")
```

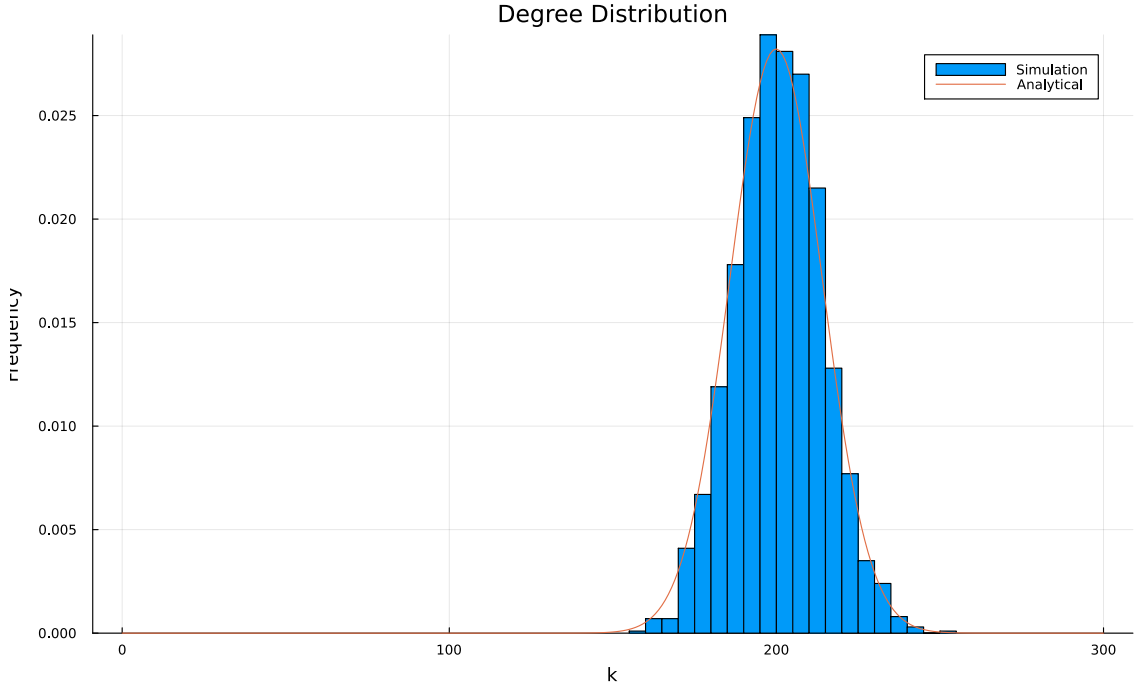
```
μ = 199.9
σ2 = 199.9
```

In [ ]:
```
plot1 = plot(avrdgree,label="Simulation")
scatter!([N],[β*(N-1)],label=L"μ")

plot2 = plot(vardgree, label = L"Simulation")
scatter!([N],[β*(N-1)],label=L"σ^2")
plot(plot1,plot2,layout = (1,2),size=(1000,600),xlabel="Iteration",title=
```





In [ ]:
```
function gaussian(x, μ, σ)
    coefficient = 1 / sqrt(2π * σ^2)
    exponent = exp(-((x - μ)^2) / (2 * σ^2))
    return coefficient * exponent
end

k = range(0,300)
histogram(degree(G),xlabel = "k",ylabel = "Frequency",normalize=true,labe
plot!(k,gaussian.(k,μ,(σ2)^.5),label="Analytical",size=(1000,600))
title!("Degree Distribution")
```



Watts-Strogatz model

Algorithm Description

1. Create a ring lattice with N nodes of mean degree K. Each node is connected to its $K/2$ nearest neighbors on either side.

2. For each edge in the graph, rewire the target node with probability β. The rewired edge cannot be a duplicate or self-loop.

The expected degree remains constant and equal the initial number of connections

$$< k >= 2K$$

The degree distribution for a large N and small β is given by

$$p_x = e^{-K\beta}\frac{(K\beta)^{x-K}}{(x-K)!}$$

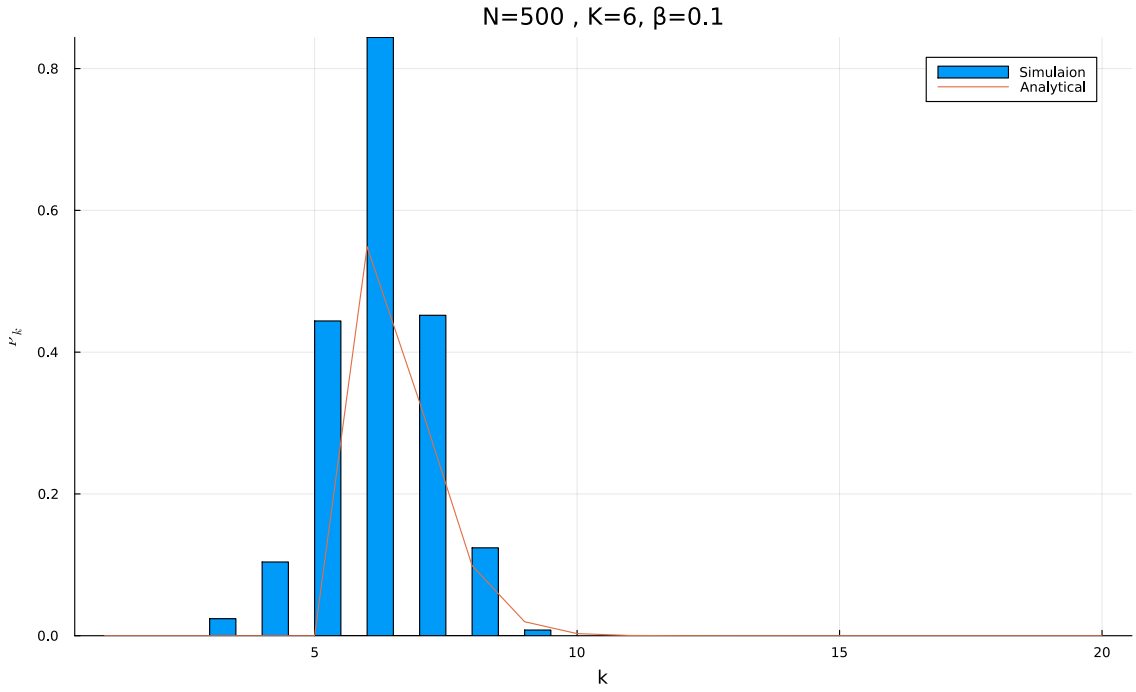
In [ ]:
```
### Creates a Ring Lattice Graph
function Ring(N,K)
    G = Graph(N)
    for q in 1:N
        for j in 1:Int(K/2)
            if(q + j > N)
                add_edge!(G,q,N - (q + j))
                add_edge!(G,q,q-j)
            elseif(q - j <= 0)
                add_edge!(G,q,N + (q-j))
                add_edge!(G,q,q+j)
            else
                add_edge!(G,q,q+j)
                add_edge!(G,q,q-j)
            end
        end
    end
    return G
end

function WattsStrogatz(N,K,β)
    G = Ring(N,K)
    for q in 1:N
        connections = neighbors(G,q)
        available = filter(x -> !(x in connections) && x!= q, 1:N)
        for j in connections
            if(rand() < β)
                new = rand(available)
                rem_edge!(G,q,j)
                add_edge!(G,q,new)
            end
        end
    end
    return G
end

function DegDistWatss(x,β,K)
    if(x >= K)
        return return exp(-K*β) * ( (K*β)^(x-K) ) / (factorial(x-K))
    else
        return 0
    end
```

```
end

function P_p(c, k, p)
    if c < k
        return 0
    else
        total_prob = 0.0
        for n in 0:min(c-k, k)
            prob_n = binomial(k, n) * (1 - p)^n * p^(k - n) * (k * p)^(c
            total_prob += prob_n
        end
        return total_prob
    end
end

function simulate(N,K,β,nsim)
    deg_counts = zeros(N)
    for q in nsim
        G = WattsStrogatz(N, K, β)
        deg = degree(G)
        for d in deg
            deg_counts[d] += 1 /nsim
        end
    end
    return deg_counts
end

function pxWatts(x,β,K)
    if(x >= K)
        return exp(-K*β) * ( (K*β)^(x-K) ) / (factorial(x-K))
    else
        return 0
    end
end
```

```
pxWatts (generic function with 1 method)
```

In [ ]:

```
β = .1
K = 6
N = 500

G = WattsStrogatz(N,K,β)
histogram(degree(G),normalize=true,label="Simulaion",size=(1000,600))
plot!(pxWatts.(1:20,β,K),label="Analytical")
xlabel!("k")
ylabel!(L"p_k")
title!("N=$(N) , K=$(K), β=$(β)")
```



Barabási-Albert Model

Algorithm description

1. Start with a small initial connected graph of N_0 nodes
2. While the number of nodes in the graph is less than the desired total number of N nodes, connect a new node to k existing nodes in the graph. 2.1 The probability of connecting to a new node i is proportional to its degree:

$$p_i = \frac{k_i}{\sum_j k_j}$$

3. Repeat the process untill N nodes are connected

The degree distribution can be approximated by

$$p(k) \approx 2m^{1/\beta} k^{-\gamma}$$

where $\gamma = 3$ and $\beta = 1/2$

The exact degree distribution of the Barabási-Albert model is

$$p_k = \frac{2m(m+1)}{k(k+1)(k+2)}$$

In [ ]:
```
function barabasi(N,m0)
    G = Graph(N)
    ### Create a fully connected graph
    for q in 1:m0 , j in q:m0
        add_edge!(G,q,j)
    end
```

```
    ### Iteration Part
    for q in m0+1:N
        prob = degree(G) / sum(degree(G))
        chosed = sample(1:N , Weights(prob), m0 , replace = false)
        for j in chosed
            add_edge!(G,q,j)
        end
    end
    return G
end

function degree_dist_barabasi(x,m0)
    return 2 * m0 * (m0+1) / (x*(x+1)*(x+2))
end


function simulate(N,m0,nsim)
    deg_counts = zeros(N)
    for q in nsim
        G = barabasi(N, m0)
        deg = degree(G)
        for d in deg
            deg_counts[d] += 1 /nsim
        end
    end
    return deg_counts
end
```

```
simulate (generic function with 2 methods)
```

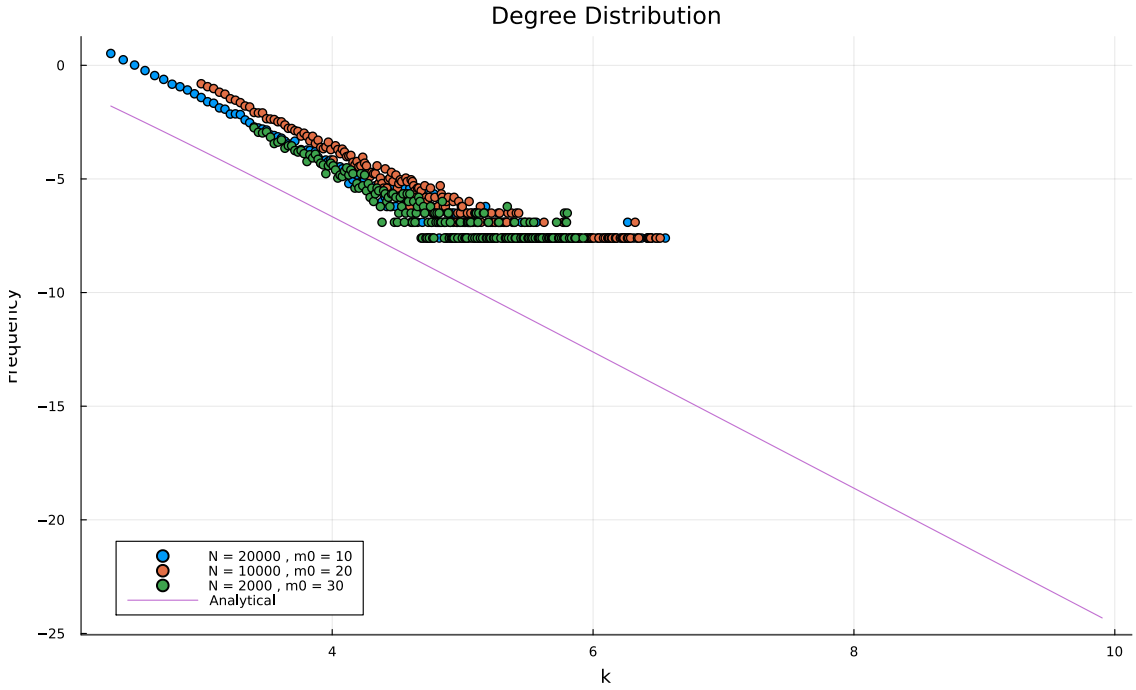
In [ ]:
```
nsim = 2000 ### Number of Simulations

N1 = 20000  ### Number of Nodes
m1 = 10     ### Number of Connections per node

N2 = 10000  ### Number of Nodes
m2 = 20     ### Number of Connections per node

N3 = 2000   ### Number of Nodes
m3 = 30     ### Number of Connections per node
```

```
30
```

In [ ]:
```
sim1 = simulate(N1,m1,nsim)
sim2 = simulate(N2,m2,nsim)
sim3 = simulate(N3,m3,nsim)

μ1 = sum(sim1[m1:N1] .* range(m1,N1))
μ2 = sum(sim2[m2:N2] .* range(m2,N2))
μ3 = sum(sim3[m3:N3] .* range(m3,N3))

Var1 = (sum((μ1 .- range(m1,N1)).^2))
Var2 = (sum((μ2 .- range(m1,N1)).^2))
Var3 = (sum((μ3 .- range(m1,N1)).^2))

println("μ1 = $(μ1) , Var1 = $(Var1)")
println("μ1 = $(μ2) , Var1 = $(Var2)")
println("μ1 = $(μ3) , Var1 = $(Var3)")
```

```
μ1 = 199.94999999999646 , Var1 = 2.587681928940479e12
μ1 = 199.8000000000001 , Var1 = 2.58774073321664e12
μ1 = 59.550000000000026 , Var1 = 2.6431163762086777e12
```

In [ ]:
```
scatter(log.(m1:N1) , log.(sim1[m1:N1]) , label = "N = $(N1) , m0 = $(m1)
scatter!(log.(m2:N2) , log.(sim2[m2:N2]) , label = "N = $(N2) , m0 = $(m2
scatter!(log.(m3:N3) , log.(sim3[m3:N3]) , label = "N = $(N3) , m0 = $(m3
plot!(log.(m1:N1) , log.(degree_dist_barabasi.(m1:N1 , m1)) , label = "An
xlabel!("k")
ylabel!("Frequency")
title!("Degree Distribution")
```



Exercice II

The following database was extracted from the Stanford Network Analysis Project(SNAP)

Dataset information:

A road network of California. Intersections and endpoints are represented by nodes and the roads connecting these intersections or road endpoints are represented by undirected edges.

In [ ]:
```
function get_data(title::AbstractString)
    base_url = "https://en.wikipedia.org/w/api.php"
    params = Dict(
        "action" => "query",
        "format" => "json",
        "titles" => title,
        "prop" => "links",
        "pllimit" => "max"
    )

    response = HTTP.get(base_url; query=params)
    data = JSON.parse(String(response.body))
    pages = data["query"]["pages"]
    links = []
```

```
    for page_id in keys(pages)
        if haskey(pages[page_id], "links")
            for link in pages[page_id]["links"]
                push!(links, link["title"])
            end
        end
    end
    return links
end

function Walk(title,edge,s,depth)
    links = get_data(title)
    s+=1
    for j in links
        push!(edge,[title,j])
    end
    for j in links
        if s >= depth
            break
        else
            Walk(j,edge,s,depth)
        end
    end
end

depth = 2
edges = []
title = "Bill Evans"
s = 0

Walk(title,edges,s,depth)
```

In [ ]:
```
df = permutedims(DataFrame(edges, :auto))
aux = unique(vcat(df[1:end,1],df[1:end,2]))
Dict_aux = Dict(zip(aux,1:length(aux)))
G = Graph(length(aux))
for row in eachrow(df)
    add_edge!(G,Dict_aux[row[1]],Dict_aux[row[2]])
end
```

In [ ]:
```
file = open("Data.txt", "w")

# Write DataFrame to the file
show(file, df)

# Close the file
close(file)
```

b)

In [ ]:
```
nodes = nv(G)
edges = ne(G)
println("Number of nodes: $(nodes)")
println("Number of edges: $(edges)")
```

```
Number of nodes: 63201
Number of edges: 144656
```

c)

In [ ]:
```
a1 = argmax(degree_centrality(G))

println("Node with biggest Degree of Centrality: $(a1)")
```

```
Node with biggest Degree of Centrality: 258
```

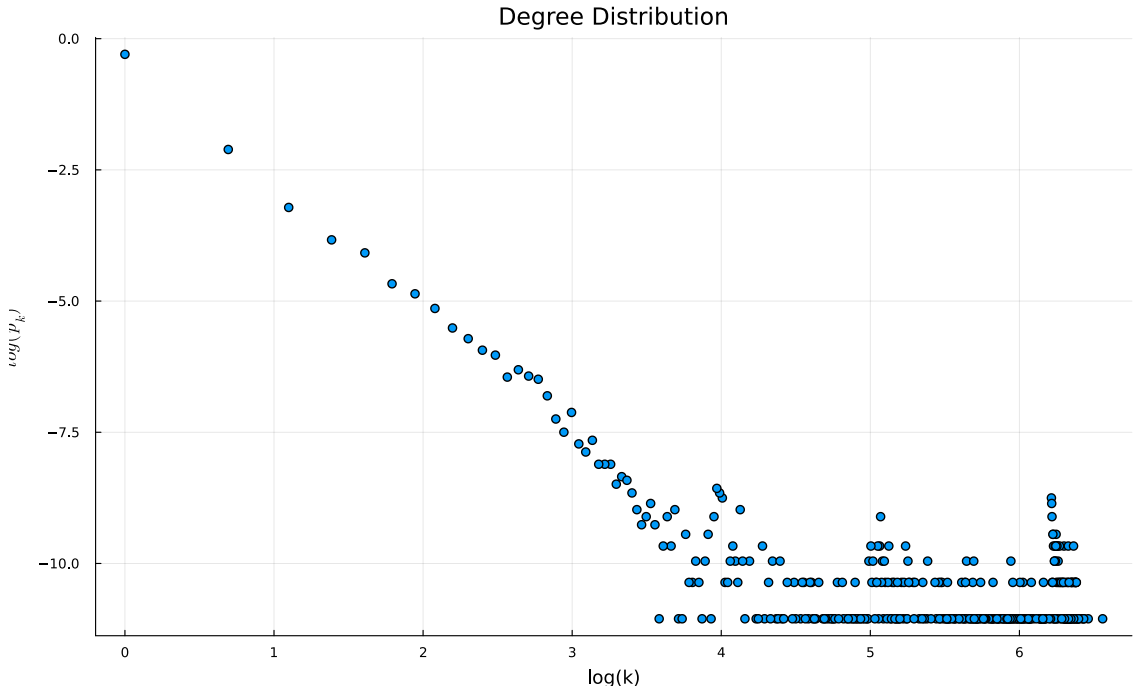
d)

In [ ]:
```
df_ = DataFrame("Number of Conncections" => zeros(nv(G)),
                "Number of Conncections of Connections" => zeros(nv(G)),
                "Indicator" => zeros(nv(G)))
for node in 1:nv(G)
    nb = neighbors(G,node)
    number_friends = length(nb)
    df_[node,1] = number_friends
    s = 0
    for j in nb
        s += length(neighbors(G,j)) / number_friends
    end
    df_[node,2] = s
    if( s >= number_friends)
        df_[node,3] = 1
    end
end
println("Fraction of nodes who are connected to less roads then the roads
```

```
Fraction of nodes who are connected to less roads then the roads connectio
ns: 0.9933545355295011
```

e)

In [ ]:
```
degrees = collect( keys(degree_histogram(G)) )
freq = collect( values(degree_histogram(G)) )

scatter(log.(degrees),log.(freq ./ sum(freq)),label=false,size=(1000,600)
xlabel!("log(k)")
ylabel!(L"log(p_k)")
title!("Degree Distribution")
```



f)

In [ ]:
```
a2 = argmax(betweenness_centrality(G))
println("Node with biggest Betweenness Centrality: $(a2)")
```

```
Node with biggest Betweenness Centrality: 1
```

g)

In [ ]:
```
gplot(G)
```