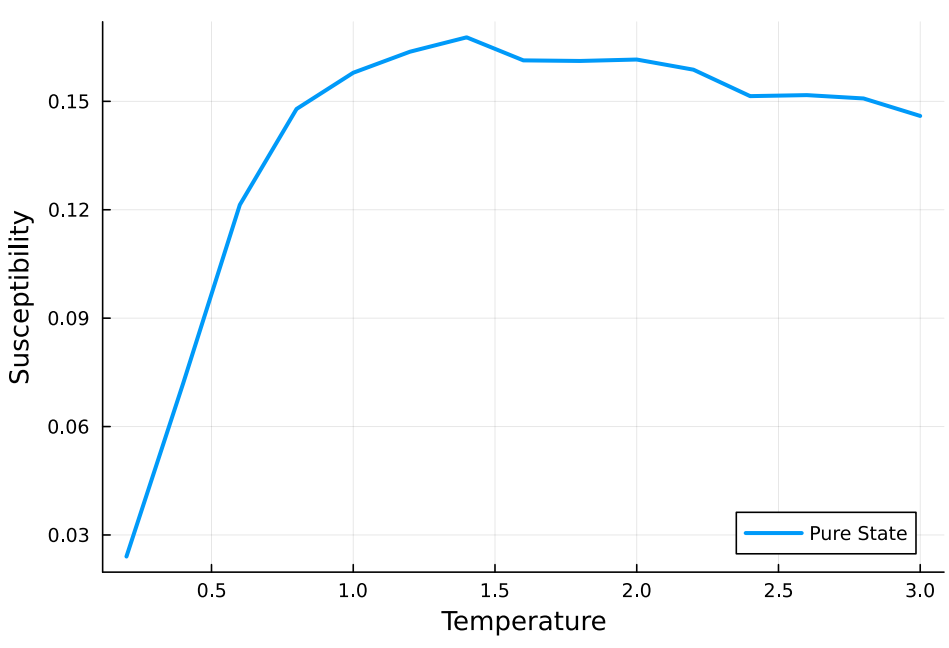
In [1]:
```
import Pkg
Pkg.add("DataStructures")
Pkg.add("Shuffle")
Pkg.add("Plots")
Pkg.add("Distributions")
Pkg.add("Random")
using DataStructures
using Shuffle
using Plots
using Random
using Distributions


#takes in an integer n
#returns array of length n of integers +/- 1 (randomly chosen) that represent sp
function initial_config(n::Int)
  config = zeros(n)
  if n%2 != 0
    println("Size n must be even.")
    return
  else
    for i=1:n
      config[i] = rand([-1, 1])
    end
  end
  #print(config)
  return config
  #essentially from N, we get a randomized array of spin ups and downs
  #e.g., N=2 may equal [1,-1]; N=4 may equal [1,-1,-1,1] as our initial configura
end

function gaussian_rf(N)
  nd = Normal(0, 1)
  return rand(nd, N)
end

function unit_rf(N)
  field = zeros(N)
  for i=1:N
    field[i] = rand([-.5, .5])
  end
  return field
end


#standard function for getting the hamilitonian of 1/r^2
#Ising Model
function get_energy(s::AbstractArray, h::AbstractArray, J)
  E0 = 0.0
  E1 = 0.0
  E2 = 0.0
  for i=1:length(s)
    #if i != length(s)
      #E0 += J*s[i]*s[i+1]
    #else
      #E0 += J*s[i]*s[1]
    #end
    for j=i:length(s)
      if j != i
        E1 += J*(s[i]-s[j])^2/(i-j)^2
      end
    end
    E2 += h[i]*s[i]
  end
  E = -E1/2 - E2
  return E
end


#magnization equals the summation of spins in
#a configuration.
#magnetization per spin = <M>/N, where
#N is basically length of the configuration
function get_magnetization(config::AbstractArray)
  M = sum(config)
  return M
end

function get_susceptibility(M_list, Msq_list, kT, N)
  avg_M = 0
  #avg_M = sum(M_list)/mcsteps
  avg_Msq = sum(Msq_list)/mcsteps
  X = (1/kT)*(avg_Msq - avg_M^2)/N
  return X
end

function do_MC_Step(config, kT, J, h, M_list, Msq_list)
  N = length(config)
  E = get_energy(config, h, J)
  for i=1:N
    site = rand(1:N)
    config[site] = -1*config[site]
    #attempt to update one site of the configuration
    E_new = get_energy(config, h, J)
    #look at the hamiltonian of the configuration
    #given this updated site
    dE = E_new - E
    #look at the difference between old and new hamiltonian
    prob = exp(-dE/(kT))
    #Metropolis acceptance ratio
    r = rand(Float64)

    if min(1, prob) > r
     #if true, configuration is "updated"
      #accepted_states += 1
      E += dE
    else
      config[site] = -1*config[site]
      #if false, then configuration stays the same
    end
  end
  M = get_magnetization(config)
  M_sq = M^2
  #println("M squared: ", M_sq)
  #push!(M_list, abs(M))
  push!(Msq_list, M_sq)
  #we get the magnization per spin of the updated system
end


function metropolis(config_initial::AbstractArray, kT, J, h, mcsteps)
  config = copy(config_initial)
  #starts out with a configuration of randomized N spin array
  mags = Vector{Float64}()
  square_mags = Vector{Float64}()
  M = get_magnetization(config)
  push!(mags, M)
  push!(square_mags, M^2)

  accepted_states = 0
  #initial state of the configuration before a MCMC update
  E = get_energy(config, h, J)
  # hamiltonian of a particular configuration
  for i=1:mcsteps
    #For MCMC an arbitrary number steps are executed for precision
    do_MC_Step(config, kT, J, h, mags, square_mags)
  end
  return mags, square_mags
end


#CONSTANTS:

#Number of spins in initialized configuration
N = 800

#Interaction constant
J = 1.0

#Initialize number of montecarlo steps
mcsteps = 10000

config0 = initial_config(N)

#Initialize random field(s)
h1 = unit_rf(N)
h2 = gaussian_rf(N)
h3= zeros(N) #PURE STATE

initkT = 0.2
iter = 0.2
finalkT = 3.0
```

```
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Project.toml`
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Project.toml`
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Project.toml`
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Project.toml`
  No Changes to `/gpfs/home/agassaml/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
```

In [2]:
```
#Pure State:
@time begin X_list1 = Vector{Float64}()
for kT = initkT:iter:finalkT
  println("At ", kT, " kT.")
  data = metropolis(config0, kT, J, h3, mcsteps)
  X = get_susceptibility(data[1], data[2], kT, length(config0))
  push!(X_list1, X)
end
plot(initkT:iter:finalkT, X_list1, xlabel = "Temperature", ylabel = "Susceptibil
end
```

```
At 0.2 kT.
At 0.4 kT.
At 0.6 kT.
At 0.8 kT.
At 1.0 kT.
At 1.2 kT.
At 1.4 kT.
At 1.6 kT.
At 1.8 kT.
At 2.0 kT.
At 2.2 kT.
At 2.4 kT.
At 2.6 kT.
At 2.8 kT.
At 3.0 kT.
97789.158662 seconds (4.41 M allocations: 244.342 MiB, 0.00% gc time, 0.01% com
pilation time: 22% of which was recompilation)
```

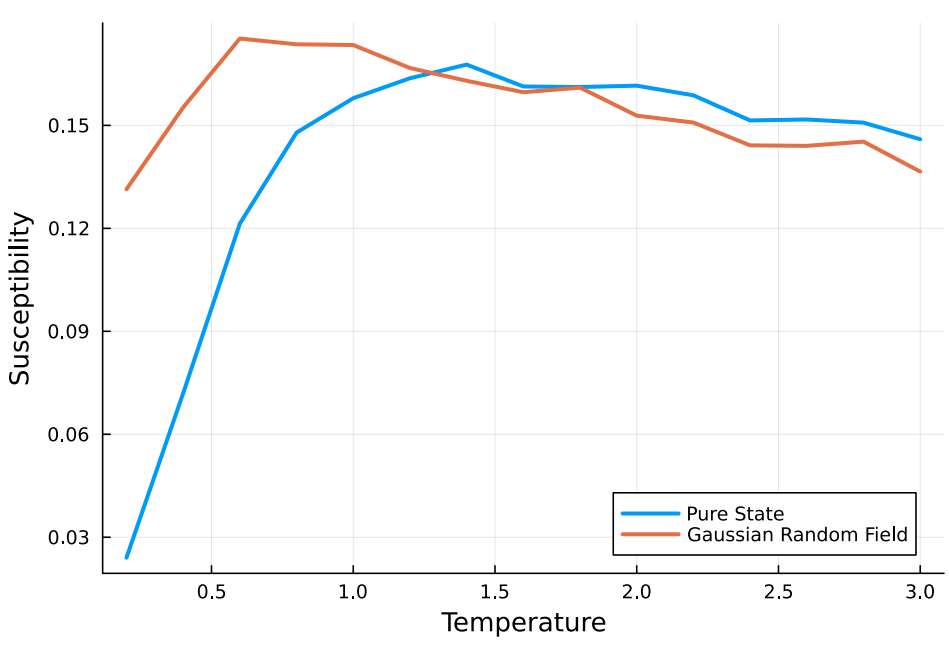
Out[2]:



In [3]:
```
#Gaussian Random Field:
@time begin X_list2 = Vector{Float64}()
for kT = initkT:iter:finalkT
  println("At ", kT, " kT.")
  data = metropolis(config0, kT, J, h2, mcsteps)
  X = get_susceptibility(data[1], data[2], kT, length(config0))
  push!(X_list2, X)
end
plot!(initkT:iter:finalkT, X_list2, xlabel = "Temperature", ylabel = "Susceptibi
end
```

```
At 0.2 kT.
At 0.4 kT.
At 0.6 kT.
At 0.8 kT.
At 1.0 kT.
At 1.2 kT.
At 1.4 kT.
At 1.6 kT.
At 1.8 kT.
At 2.0 kT.
At 2.2 kT.
At 2.4 kT.
At 2.6 kT.
At 2.8 kT.
At 3.0 kT.
97617.048186 seconds (335.45 k allocations: 22.453 MiB, 0.00% compilation time)
```

Out[3]:



In [17]:
```
print(X_list2)
```

```
[0.25007999999999997, 0.28896, 0.24372000000000005, 0.23897, 0.223208, 0.218373
33333333334, 0.2132457142857143, 0.201125, 0.19762666666666667, 0.1914239999999
9998, 0.17792727272727274, 0.18039666666666668, 0.1668, 0.16540285714285716, 0.
15995199999999998]
```

In [ ]:

In [ ]: