Code Available at https : // github . com/zimmermannJakob/InfectionSpreadingOnNetworks

Author: Jakob Zimmermann

Infection Spreading on Networks

- Distributed connectivity of nodes and SIR model
- Super spreader as super connected nodes
- Visualization of the one-wave spreading dynamic

Overview

1. Algorithm/Visualization/Evaluation

2. Comparison to SIR model with differential equations

3. Comparing simulation results for graphs with different vertex connectivity

4. Wave spreading dynamics

---

1. The Algorithm

In[1]:=
```
epidemicSimulation[graph_, numNodes_,
  startInfectionNumber_, transitionIR_, infectionRate_, tMax_ ] :=
 {(*Setting random seed for reproductivity of results*)
  SeedRandom[42];
  (*statusList keeps track of the status of each node ;
  0: S;
  1: I;
  2: R
  *)
  statusList = ConstantArray[0, numNodes];
  (*infectionTimeList keeps track of the time
   each node has been infected (initilized with '-1')*)
  infectionTimeList = ConstantArray[-1, numNodes];

  (*defining tCurrent to keep track of
   the number of itterations which are actually needed*)
  tCurrent = 0;

  (*number of people in S, I and R respectively*)
```

```
numS = numNodes - startInfectionNumber;
numI = startInfectionNumber;
numR = 0;
(*Initializing S(t), I(t), R(t)*)
listS = List[];
listI = List[];
listR = List[];
(*Initializing statusList(t) and infectionTimeList(t)*)
infectionTimeListT = List[];
statusListT = List[];
(*Infecting startInfectionNumber-people, picked random out of all nodes*)
firstInfectionList =
 Table[RandomInteger[{1, numNodes}], startInfectionNumber];
(*Updating statusList and infectionTimeList
 to infect the random selected nodes from before*)
For[i = 0, i < Length[firstInfectionList] , i++;
  statusList[[firstInfectionList[[i]]]] = 1;
  infectionTimeList[[firstInfectionList[[i]]]] = transitionIR - 1;
 ]
 (*Updating numS(t), numI(t) and numR(t)*)
 AppendTo[listS, numS];
AppendTo[listI, numI];
AppendTo[listR, numR];
(*Updating infectionTimeList(t) and statusList(t)*)
AppendTo[infectionTimeListT, infectionTimeList];
AppendTo[statusListT, statusList];

(*Start of the actual simulation here*)
For[t = 0, t < tMax && numI ≠ 0, t++;
 tCurrent++;

 (*infecting people*)
 infectedNodes = List[];
 (*save every node to a seperate list*)
 For[nodeIndex = 0, nodeIndex < numNodes, nodeIndex++;
  If[statusList[[nodeIndex]] == 1,
   AppendTo[infectedNodes, nodeIndex];
  ];
 ];
 (*for every infected node: infect neighbors:*)
 For[infectedNodesIndex = 0,
  infectedNodesIndex < Length[infectedNodes], infectedNodesIndex++;
  (*get neighbors*)
  adjList = AdjacencyList[graph, infectedNodes[[infectedNodesIndex]]];
  For[neighborIndex = 0, neighborIndex < Length[adjList], neighborIndex++;
   (*If the node is not infected yet,
   infect it with the probability of infection rate*)
```

```
    If[RandomReal[] ≤ infectionRate && statusList[[adjList[[neighborIndex]]]] == 0,
     statusList[[adjList[[neighborIndex]]]] = 1;
     infectionTimeList[[adjList[[neighborIndex]]]] = transitionIR;
     (*update the number of susceptable and infected people*)
     numS--;
     numI++;
    ];

   ];
  ];

  (*removing people*)
  (*for every node ...*)
  For[nodeIndex = 0, nodeIndex < numNodes, nodeIndex++;
   (*... if the node is infected...*)
   If[infectionTimeList[[nodeIndex]] ≠ -1,
    (**... reduce the days the node is infected by 1*)
    infectionTimeList[[nodeIndex]] = infectionTimeList[[nodeIndex]] - 1;
    (*If the status goes back to not being infected...*)
    If[infectionTimeList[[nodeIndex]] == -1,
     (*update the status*)
     statusList[[nodeIndex]] = 2;
     (*update the number of infected and removed people*)
     numI--;
     numR++;
    ];

   ];
  ];

  (*Appending after each itteration*)
  AppendTo[listS, numS];
  AppendTo[listI, numI];
  AppendTo[listR, numR];
  (*Updating infectionTimeList(t) and statusList(t)*)
  AppendTo[infectionTimeListT, infectionTimeList];
  AppendTo[statusListT, statusList];
 ]
}
```

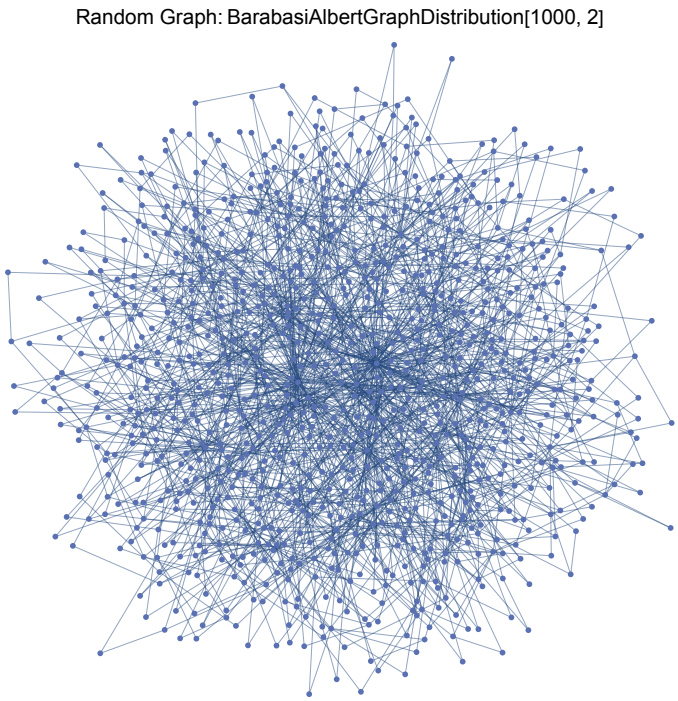
Running the simulation on a random graph

In[2]:=
```
numberOfNodes = 1000;
startInfectionNumber = 3;
transitionIR = 5;
infectionRate = .2;
tMax = 30;
randomGraph =
  RandomGraph[BarabasiAlbertGraphDistribution[numberOfNodes, 2]];

epidemicSimulation[randomGraph, numberOfNodes,
  startInfectionNumber, transitionIR, infectionRate, tMax ];
```

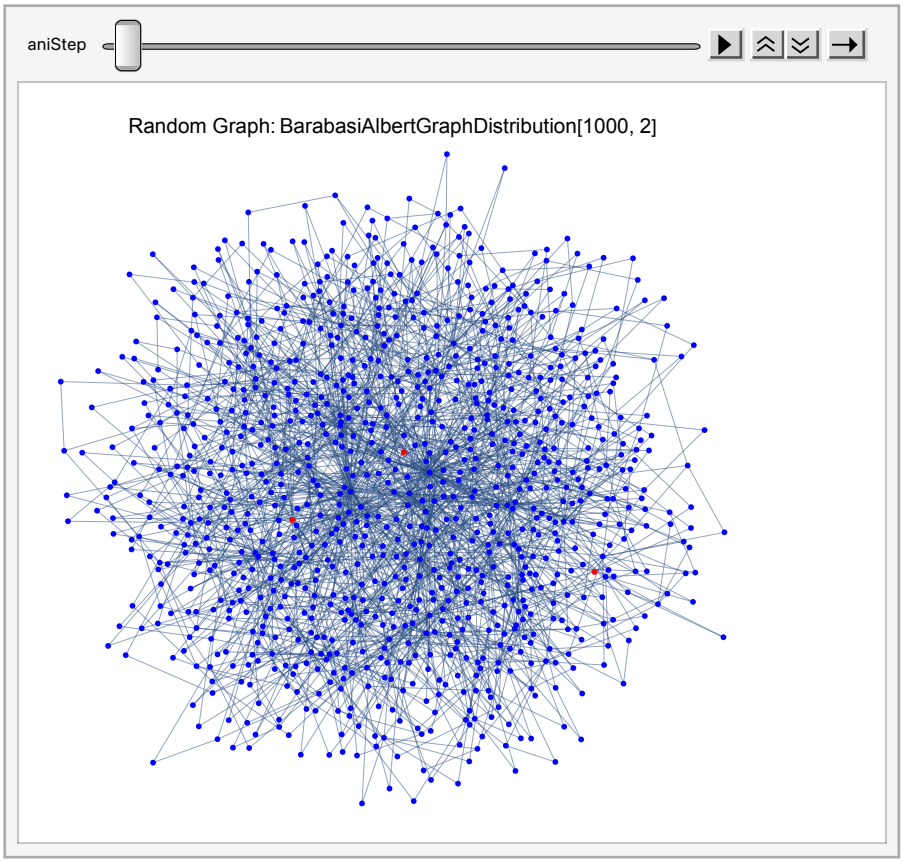
Visualization of the Results

In[9]:=
```
graphConnectivity = VertexDegree[randomGraph];
maxConnection = Max[graphConnectivity];

(*Changing the size of the nodes (doesn't work with large numbers of nodes)*)
For[i = 1, i ≤ Length[graphConnectivity], i++,
 randomGraph =
  Annotate[randomGraph, VertexSize → {(i → graphConnectivity[[i]] / (10))}]
]
(*Annotate the graph to have thin edges and labels*)
randomGraph = Annotate[randomGraph, {EdgeStyle → Thin, PlotLabel →
    "Random Graph:" BarabasiAlbertGraphDistribution[numberOfNodes, 2]}]
(*--------------------*)
(*creating the animation object*)
animation = Animate[HighlightGraph[randomGraph,
   {Style[Flatten[Position[statusListT[[aniStep]], 0]], Blue],
    Style[Flatten[Position[statusListT[[aniStep]], 1]], Red],
    Style[Flatten[Position[statusListT[[aniStep]], 2]], Gray]
   } ], {aniStep, 1, tCurrent + 1, 1}, AnimationRunning → False, AnimationRate → 2]
```

Out[12]=


Random Graph: BarabasiAlbertGraphDistribution[1000, 2]

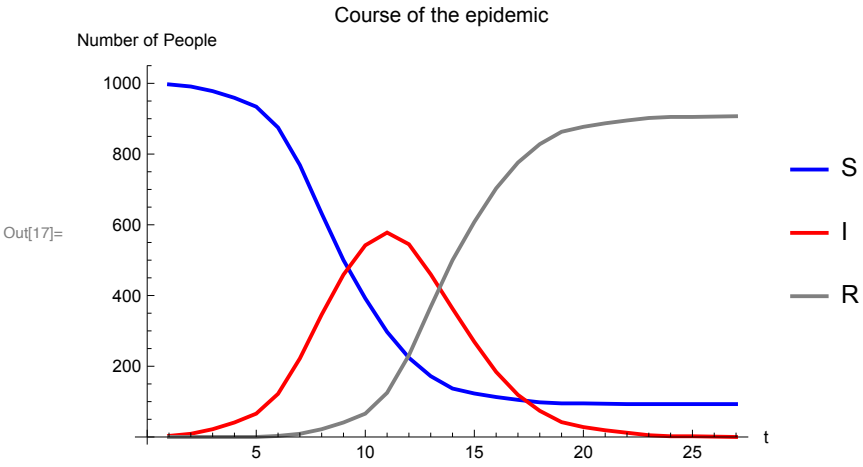
Out[13]=


aniStep
Random Graph: BarabasiAlbertGraphDistribution[1000, 2]

Plot the course of the Epidemic

In[14]:=
```
(*Plotting course of the compartment S*)
plotS = ListLinePlot[listS, PlotStyle → {Blue, Thick},
    AxesLabel → {"t", "Number of People"},
    PlotLegends → {"S"}];
(*Plotting course of the compartment I*)
plotI = ListLinePlot[listI, PlotStyle → {Red, Thick}, PlotLegends → {"I"}];
(*Plotting course of the compartment R*)
plotR = ListLinePlot[listR, PlotStyle → {Gray, Thick}, PlotLegends → {"R"}];
(*Showing all the plots created above in one image*)
img = Show[plotS, plotI, plotR, PlotRange → All,
  PlotLabel → "Course of the epidemic"]
```

Out[17]=



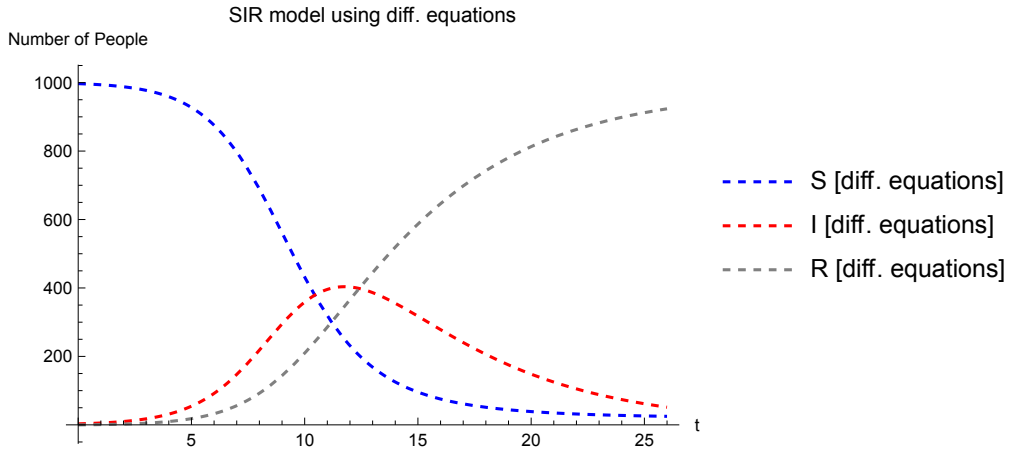
2. Comparison to SIR model with Differential Equations

In[18]:=
```
Clear[s, i, r, β, γ, n, t]

β = infectionRate * Mean[VertexDegree[randomGraph]];
(*average number of contacts per node*)
γ = 1 / transitionIR;

n = numberOfNodes;
sol = NDSolve[{
    s'[t] == -β i[t] × s[t] / n,
    i'[t] == β i[t] × s[t] / n - γ i[t],
    r'[t] == γ i[t],

    s[0] == n - startInfectionNumber,
    i[0] == startInfectionNumber,
    r[0] == 0
   },
   {s, i, r},
   {t, tCurrent}
  ];
```

Plotting Results

In[23]:=
```
solutionS = First[s /. sol];
solutionI = First[i /. sol];
solutionR = First[r /. sol];

plotDiffEqu = Plot[{solutionS[t], solutionI[t], solutionR[t]}, {t, 0, tCurrent},
  PlotStyle → {{Dashed, Blue}, {Dashed, Red}, {Dashed, Gray}},
  PlotLabel → "SIR model using diff. equations", PlotLegends →
   {"S [diff. equations]", "I [diff. equations]", "R [diff. equations]"},
  AxesLabel → {"t", "Number of People"}]
```

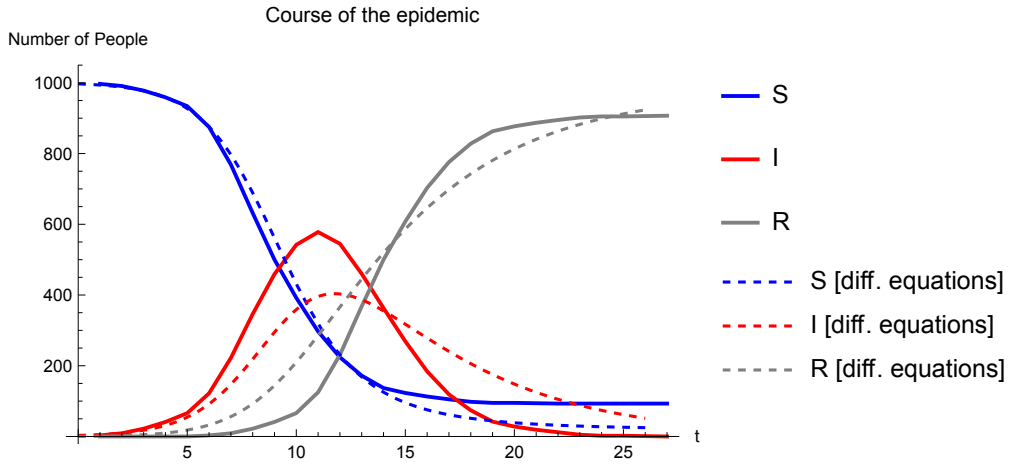
Out[26]=



Direct Comparison

In[27]:=
```
Show[img, plotDiffEqu]
```

Out[27]=



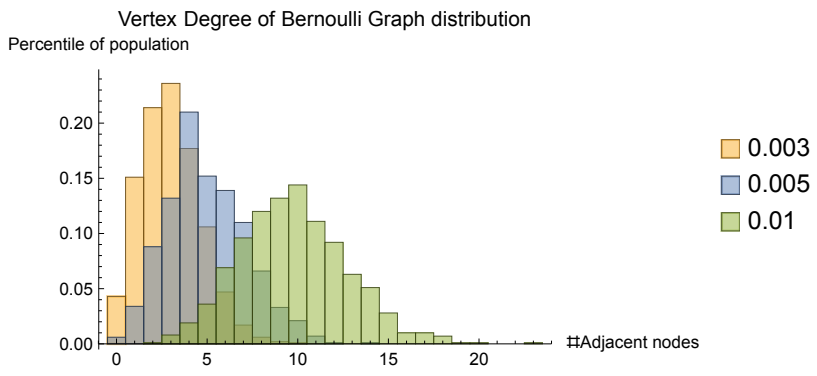
3. Random - Graphs with different Vertex connectivity

Using the Bernoulli - graph distribution (Bernoulli distribution with E[X] = n*p)
Three models of connectivity :
- very few super-spreaders (e . g practicing of good social distancing, sparse settled regions)

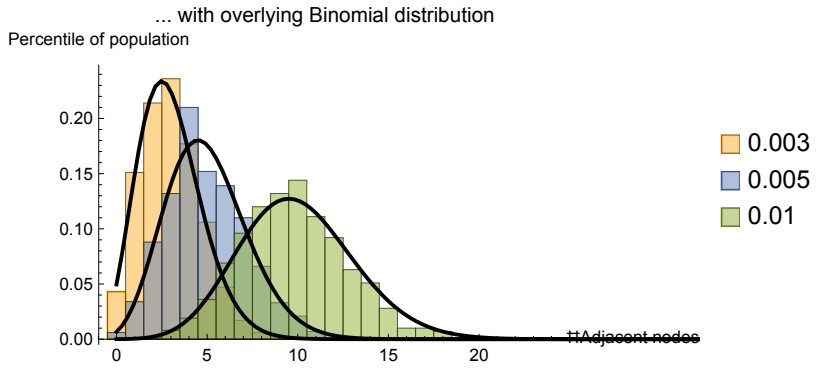
- a lot of super-spreaders (e . g densely populated areas, frequent events with a lot of person to person contacts)
- balanced amount of super-spreaders

In[28]:=
```
SeedRandom[42];
n = 1000;
(*defining the probability used for crating the graphs in listP*)
listP = {N[3 / n], N[5 / n], N[10 / n]};
(*listD is used to store the vertex degree for the three models used*)
listD = {};

(*Plot the PDF for every element of listP
  using the binomial distribution with p ∈ listP*)
pdfPlot = Plot[
   Evaluate@Table[PDF[BinomialDistribution[n, p], x], {p, listP}], {x, 0, n},
   PlotRange → All,
   PlotStyle → {{Thick, Black}, {Thick, Black}, {Thick, Black}}
  ];

(*Get the vertex degree for every graph*)
For[i = 1, i ≤ Length[listP], i++,
 g = RandomGraph[BernoulliGraphDistribution[n, listP[[i]]]];
 AppendTo[listD, VertexDegree[g]];
]
(*plot the hists for every obtained vertex degree*)
pdfHist = Histogram[listD, n - 1, "PDF", PlotRange → All, ChartLegends → listP,
  PlotLabel → "Vertex Degree of Bernoulli Graph distribution",
  AxesLabel → {"#Adjacent nodes", "Percentile of population"}]

(*showing the hists overlayed with the PDF of the Binomial distribution*)
Show[pdfHist, pdfPlot, PlotLabel → "... with overlying Binomial distribution"]
```

Out[34]=



Out[35]=



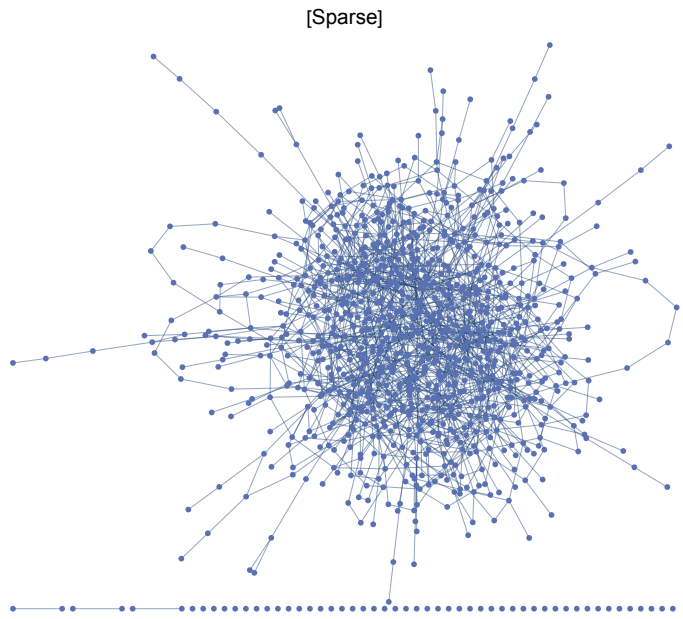
Creating Graphs and their Histograms

In[36]:=
```
scaling = 20;
```

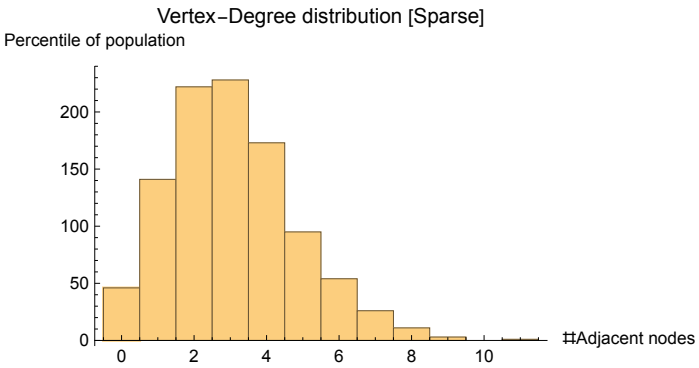
Sparse

In[37]:=
```
graphSparse = RandomGraph[BernoulliGraphDistribution[n , listP[[1]]]];
(*Annotating the graph*)
graphConnectivity = VertexDegree[graphSparse];
maxConnection = Max[graphConnectivity];
For[i = 1, i ≤ Length[graphConnectivity], i++,
 graphSparse =
  Annotate[graphSparse, VertexSize → {(i → graphConnectivity[[i]] / (scaling))}]
]
graphSparse = Annotate[graphSparse, {EdgeStyle → Thin, PlotLabel → "[Sparse]"}]

(*plotting the histogram*)
Show[VertexDegree[graphSparse] // Histogram,
 PlotLabel → "Vertex-Degree distribution [Sparse]",
 AxesLabel → {"#Adjacent nodes", "Percentile of population"}]
```

Out[41]=



Out[42]=

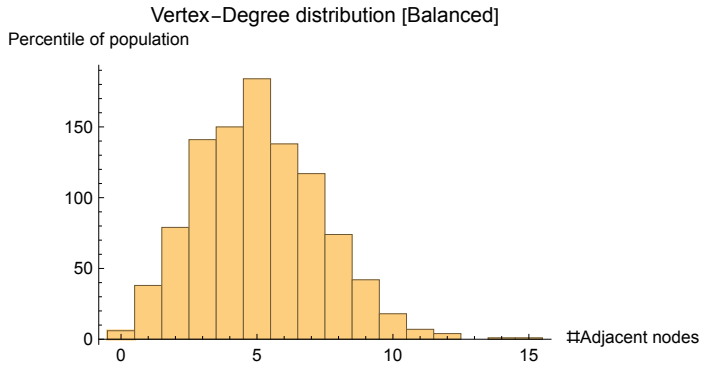


Balanced

In[43]:=
```
graphBalanced = RandomGraph[BernoulliGraphDistribution[n, listP[[2]]]];

(*Annotating the graph*)
graphConnectivity = VertexDegree[graphBalanced];
maxConnection = Max[graphConnectivity];
For[i = 1, i ≤ Length[graphConnectivity], i++,
 (*every node gets a corresponding
  size depending of connectivity and scaling*)
 graphBalanced =
  Annotate[graphBalanced, VertexSize → {(i → graphConnectivity[[i]] / (scaling))}]
]
(*change the edges to be thin*)
graphBalanced =
 Annotate[graphBalanced, {EdgeStyle → Thin, PlotLabel → "[Balanced]"}]

(*plotting the histogram*)
Show[VertexDegree[graphBalanced] // Histogram,
 PlotLabel → "Vertex-Degree distribution [Balanced]",
 AxesLabel → {"#Adjacent nodes", "Percentile of population"}]
```

Out[47]=

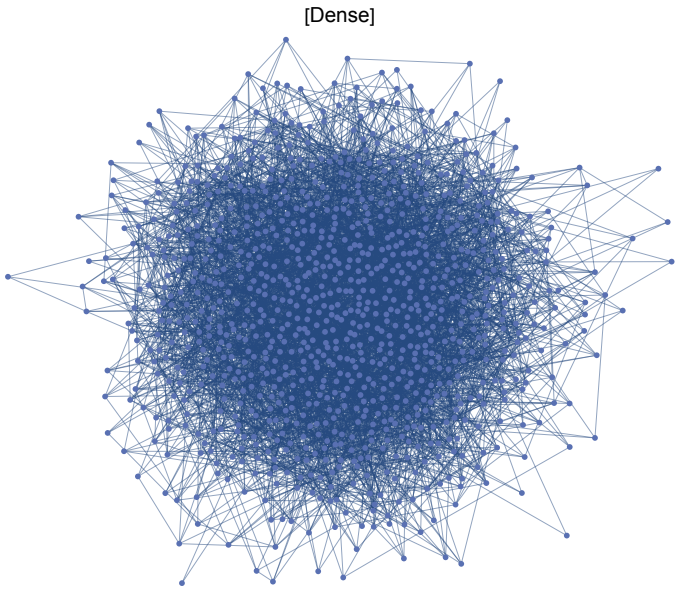
Out[48]=


Vertex−Degree distribution [Balanced]
Percentile of population
150
100
50
0
0
5
10
15
#Adjacent nodes

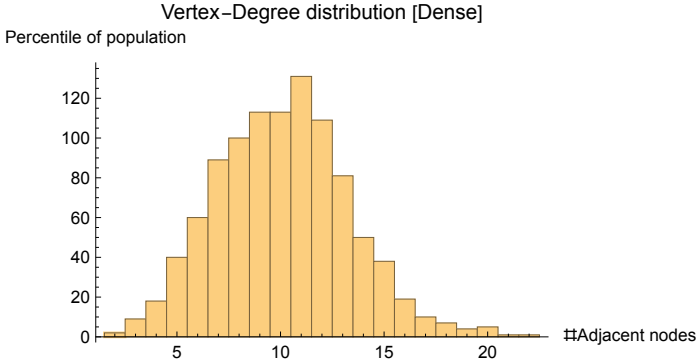
Dense

In[49]:=
```
graphDense = RandomGraph[BernoulliGraphDistribution[n, listP[[3]]]];

(*Annotating the graph*)
graphConnectivity = VertexDegree[graphDense];
maxConnection = Max[graphConnectivity];
For[i = 1, i ≤ Length[graphConnectivity], i++,
 graphDense =
  Annotate[graphDense, VertexSize → {(i → graphConnectivity[[i]] / (scaling))}]
]
graphDense = Annotate[graphDense, {EdgeStyle → Thin, PlotLabel → "[Dense]"}]

(*Plotting the histogram*)
Show[VertexDegree[graphDense] // Histogram,
 PlotLabel → "Vertex-Degree distribution [Dense]",
 AxesLabel → {"#Adjacent nodes", "Percentile of population"}]
```

Out[53]=



Out[54]=

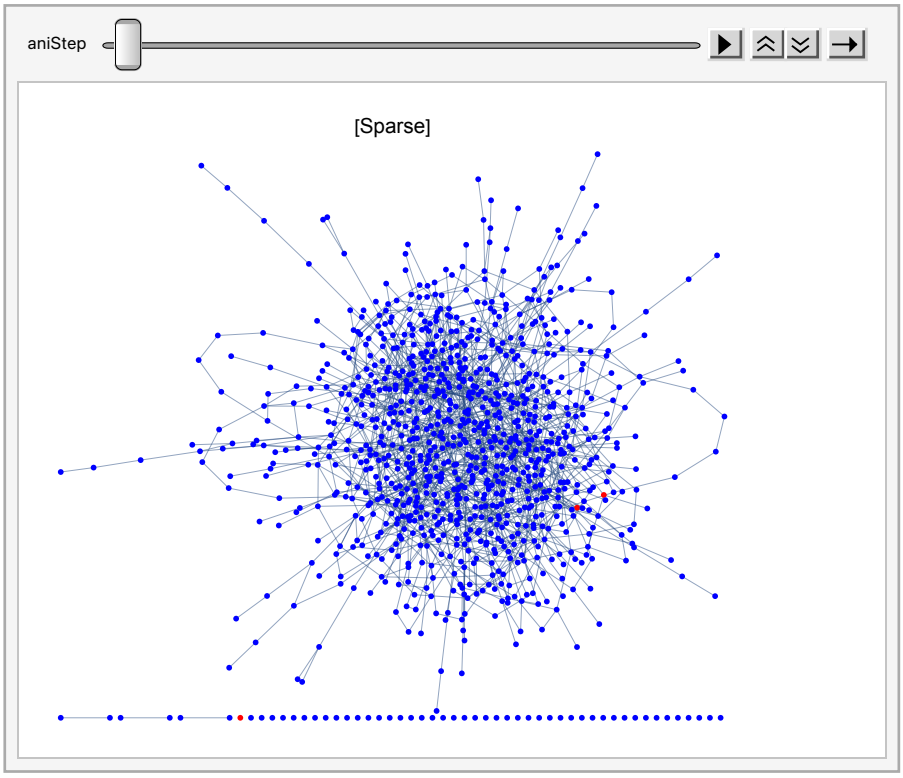


Running the Simulation

Sparse

In[55]:=
```
(*running the simulation on the respective graph*)
epidemicSimulation[graphSparse, n, 3, 5, .4, 30];
(*get the values needed for further evaluation after each run*)
listSSparse = listS;
listISparse = listI;
listRSparse = listR;

(*animating the course of the epidemic*)
animation = Animate[HighlightGraph[graphSparse,
   {Style[Flatten[Position[statusListT[[aniStep]], 0]], Blue],
    Style[Flatten[Position[statusListT[[aniStep]], 1]], Red],
    Style[Flatten[Position[statusListT[[aniStep]], 2]], Gray]
   } ], {aniStep, 1, tMax, 1}, AnimationRunning → False, AnimationRate → 2]
```

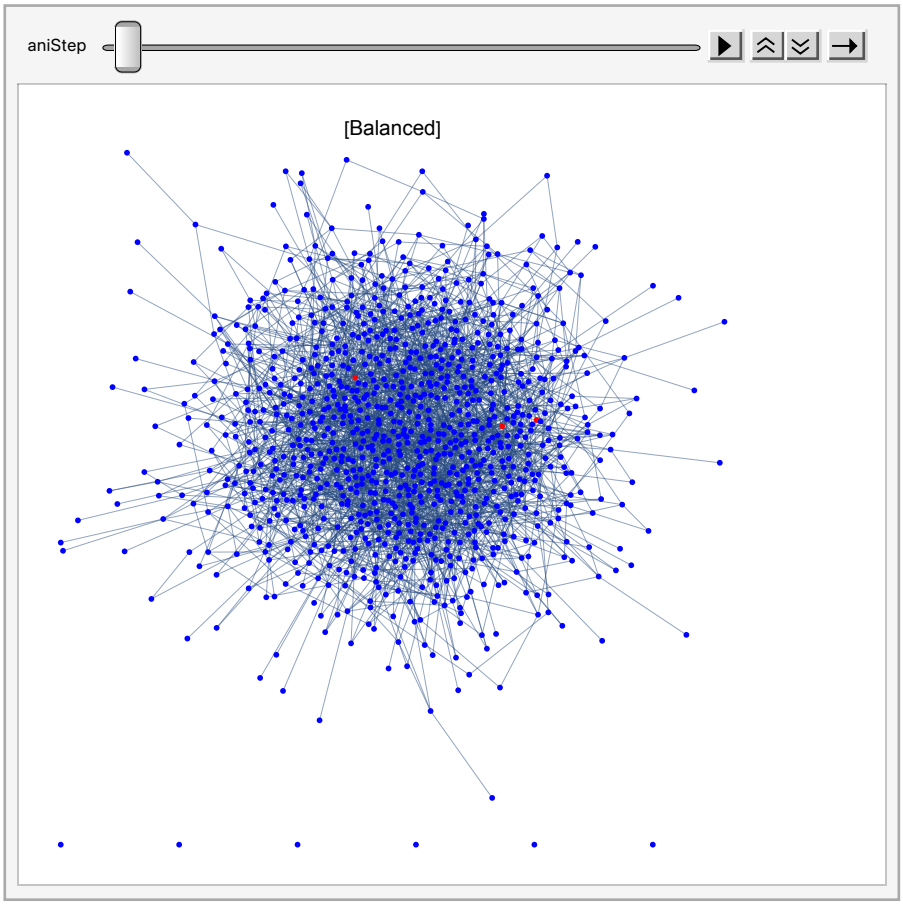
Out[59]=



Balanced

In[60]:=
```
epidemicSimulation[graphBalanced, n, 3, 5, .4, 30];
statusListTBalanced = statusListT;
listSBalanced = listS;
listIBalanced = listI;
listRBalanced = listR;

animation = Animate[HighlightGraph[graphBalanced,
   {Style[Flatten[Position[statusListT[[aniStep]], 0]], Blue],
    Style[Flatten[Position[statusListT[[aniStep]], 1]], Red],
    Style[Flatten[Position[statusListT[[aniStep]], 2]], Gray]
   } ], {aniStep, 1, tMax, 1}, AnimationRunning → False, AnimationRate → 2]
```

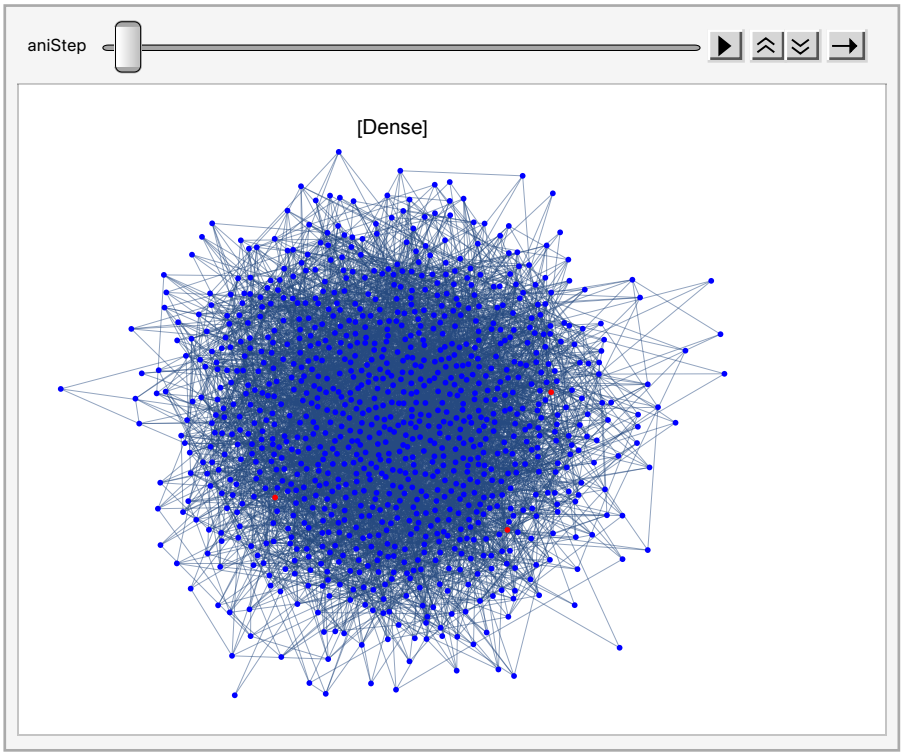
Out[65]=



Dense

In[66]:=
```
epidemicSimulation[graphDense, n, 3, 5, .4, 30];
statusListDense = statusListT;
listSDense = listS;
listIDense = listI;
listRDense = listR;

animation = Animate[HighlightGraph[graphDense,
   {Style[Flatten[Position[statusListT[[aniStep]], 0]], Blue],
    Style[Flatten[Position[statusListT[[aniStep]], 1]], Red],
    Style[Flatten[Position[statusListT[[aniStep]], 2]], Gray]
   } ], {aniStep, 1, tMax, 1}, AnimationRunning → False, AnimationRate → 2]
```

Out[71]=

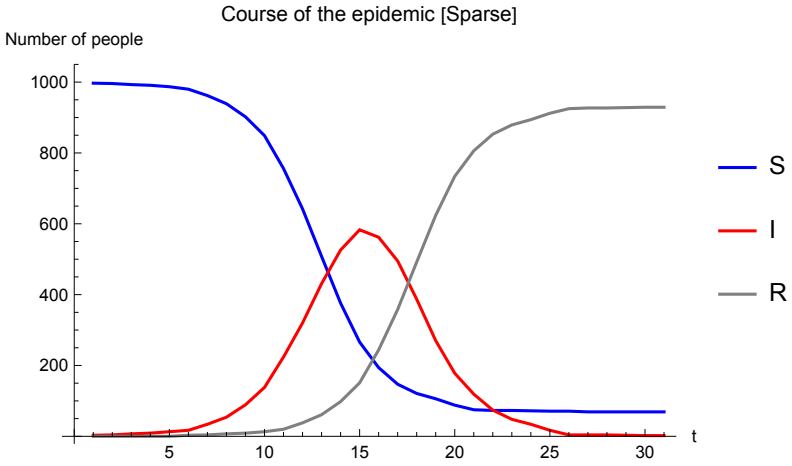


Evaluating the Results

Sparse

In[72]:=
```
(*Plotting the course of S, I and R*)
plotSSparse = ListLinePlot[listSSparse, PlotStyle → Blue,
    PlotLegends → {"S"}, AxesLabel → {"t", "Number of people"}];
plotISparse = ListLinePlot[listISparse, PlotStyle → Red, PlotLegends → {"I"}];
plotRSparse = ListLinePlot[listRSparse, PlotStyle → Gray, PlotLegends → {"R"}];

(*combining the plots from above*)
img = Show[plotSSparse, plotISparse, plotRSparse,
  PlotRange → All, PlotLabel → "Course of the epidemic [Sparse]"]
```

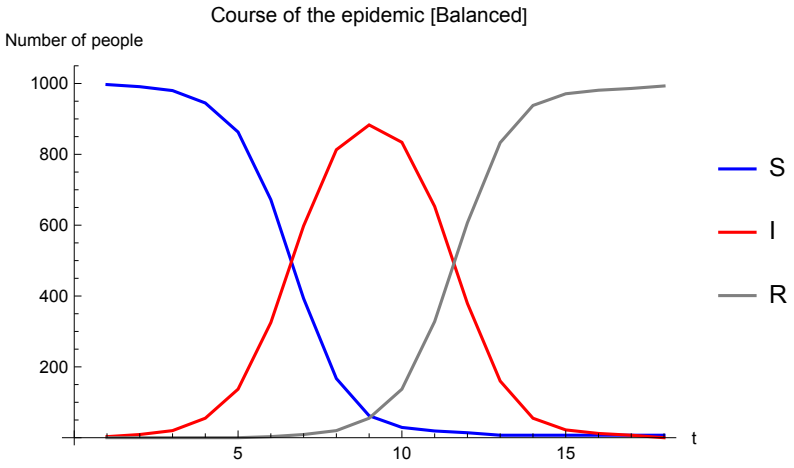
Out[75]=



Balanced

In[76]:=
```
plotSBalanced = ListLinePlot[listSBalanced, PlotStyle → Blue,
    PlotLegends → {"S"}, AxesLabel → {"t", "Number of people"}];
plotIBalanced =
  ListLinePlot[listIBalanced, PlotStyle → Red, PlotLegends → {"I"}];
plotRBalanced =
  ListLinePlot[listRBalanced, PlotStyle → Gray, PlotLegends → {"R"}];
img = Show[plotSBalanced, plotIBalanced, plotRBalanced,
  PlotRange → All, PlotLabel → "Course of the epidemic [Balanced]"]
```

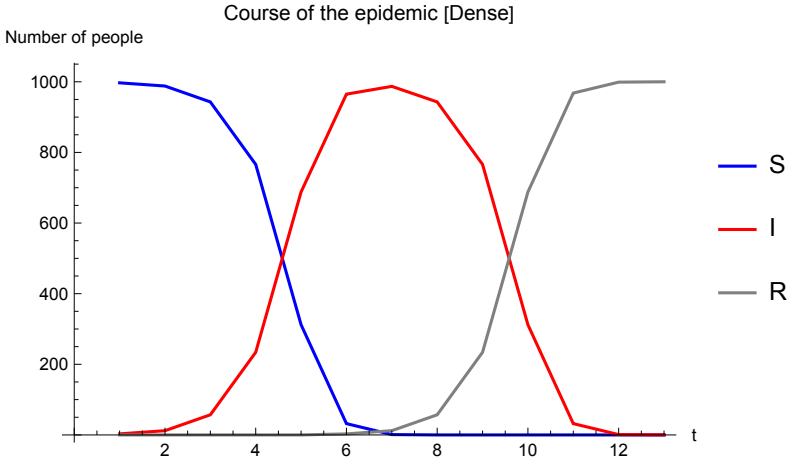
Out[79]=



Dense

In[80]:=
```
plotSDense = ListLinePlot[listSDense, PlotStyle → Blue,
    PlotLegends → {"S"}, AxesLabel → {"t", "Number of people"}];
plotIDense = ListLinePlot[listIDense, PlotStyle → Red, PlotLegends → {"I"}];
plotRDense = ListLinePlot[listRDense, PlotStyle → Gray, PlotLegends → {"R"}];
img = Show[plotSDense, plotIDense, plotRDense,
  PlotRange → All, PlotLabel → "Course of the epidemic [Dense]"]
```

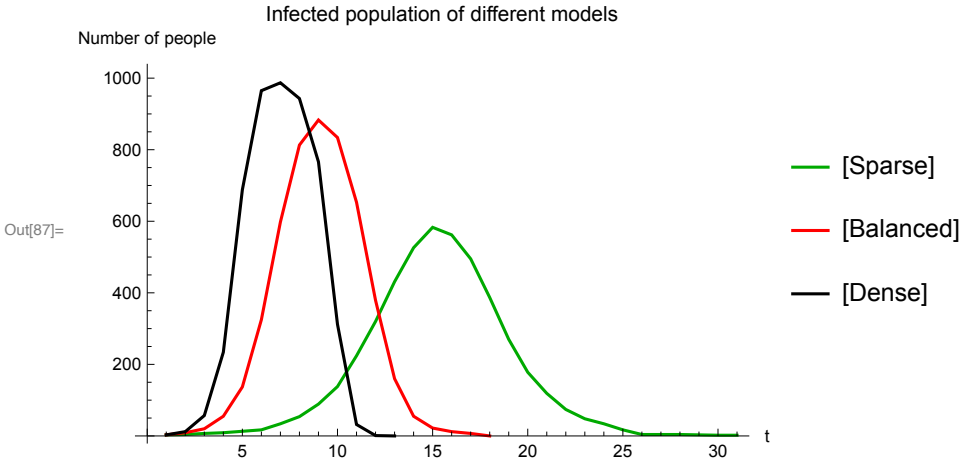
Out[83]=



Overlaying curves of infectious individuals

In[84]:=
```
plotISparse = ListLinePlot[listISparse, PlotStyle → Darker[Green],
    PlotLegends → {"[Sparse]"}, AxesLabel → {"t", "Number of people"}];
plotIBalanced =
  ListLinePlot[listIBalanced, PlotStyle → Red, PlotLegends → {"[Balanced]"}];
plotIDense =
  ListLinePlot[listIDense, PlotStyle → Black, PlotLegends → {"[Dense]"}];
(*Showing everything together in one plot*)
Show[plotISparse, plotIBalanced, plotIDense, PlotRange → All,
 PlotLabel → "Infected population of different models"]
```

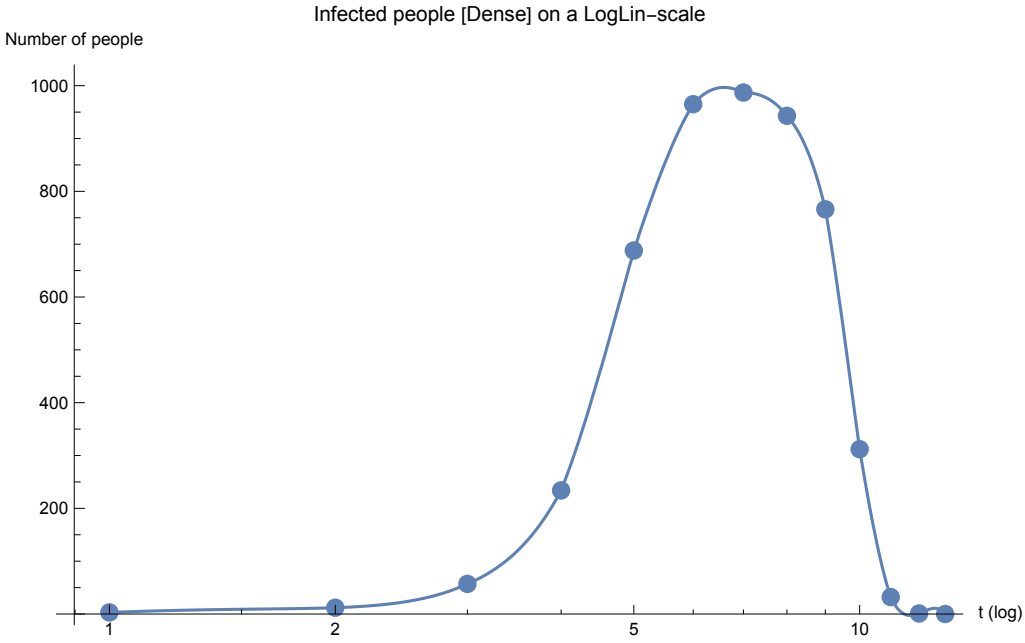
Out[87]=



(Trying to) Show Exponential Growth

In[88]:=
```
(*interpolating the values to get a nice courve to look at,
while the values are plotted on a logarithmic scale*)
f = ListInterpolation[listIDense];

(*Plot the I- values on a LogLin scale to show that the spreading
  of the epidemic is exponential (hence linear in LogLin)*)
Show[ListLogLinearPlot[listIDense, PlotMarkers → {Automatic, 10},
  AxesLabel → {"t (log)", "Number of people"}],


 LogLinearPlot[f[x], {x, 1, Length[listIDense]}],
 PlotLabel → "Infected people [Dense] on a LogLin-scale"]
```

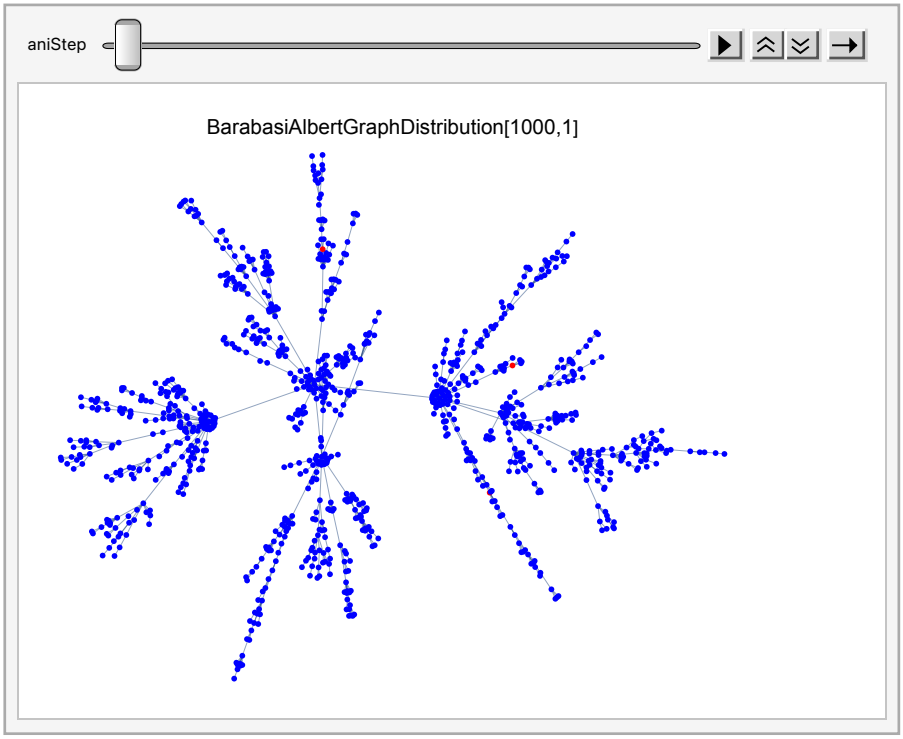
Out[89]=



Visualizing Wave Spreading Dynamics

In[90]:=
```
SeedRandom[42];
n = 1000;
k = 1;
graphCluster = RandomGraph[BarabasiAlbertGraphDistribution[n, k]];
epidemicSimulation[graphCluster, n, 3, 5, 1, 30];
(*annotating*)
graphCluster = Annotate[graphCluster,
   {EdgeStyle → Thin, PlotLabel → "BarabasiAlbertGraphDistribution[1000,1]"}];
(*animating*)
animation = Animate[HighlightGraph[graphCluster,
   {Style[Flatten[Position[statusListT[[aniStep]], 0]], Blue],
    Style[Flatten[Position[statusListT[[aniStep]], 1]], Red],
    Style[Flatten[Position[statusListT[[aniStep]], 2]], Gray]
   } ], {aniStep, 1, tCurrent + 1, 1}, AnimationRunning → False,
  AnimationRate → 2, AnimationDirection → Forward]
```

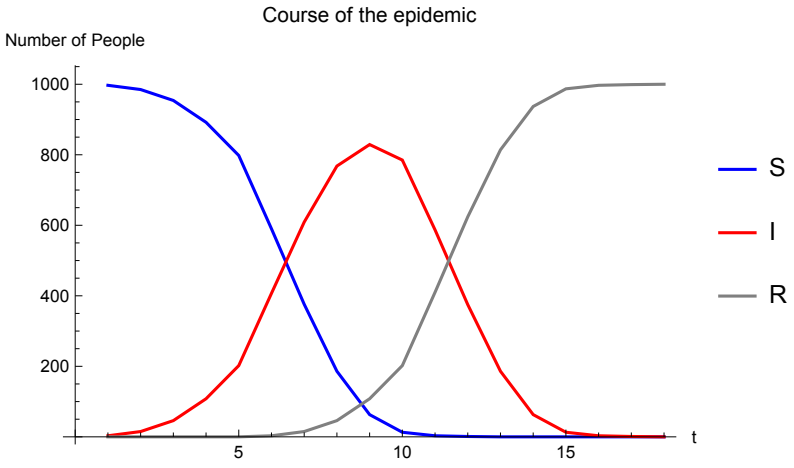
Out[96]=



Evaluation

In[97]:=
```
plotS = ListLinePlot[listS, PlotStyle → Blue,
    PlotLegends → {"S"}, AxesLabel → {"t", "Number of People"}];
plotI = ListLinePlot[listI, PlotStyle → Red, PlotLegends → {"I"}];
plotR = ListLinePlot[listR, PlotStyle → Gray, PlotLegends → {"R"}];
img = Show[plotS, plotI, plotR,
  PlotRange → All, PlotLabel → "Course of the epidemic"]
```

Out[100]=



Show Exponential Growth

In[101]:=
```
f = ListInterpolation[listI];
Show[ListLogLinearPlot[listI, AxesLabel → {"t (log)", "Number of people"}],
 LogLinearPlot[f[x], {x, 1, Length[listI]}],
 PlotLabel → "Infected people [Dense] on a LogLin-scale"]
```

Out[102]=