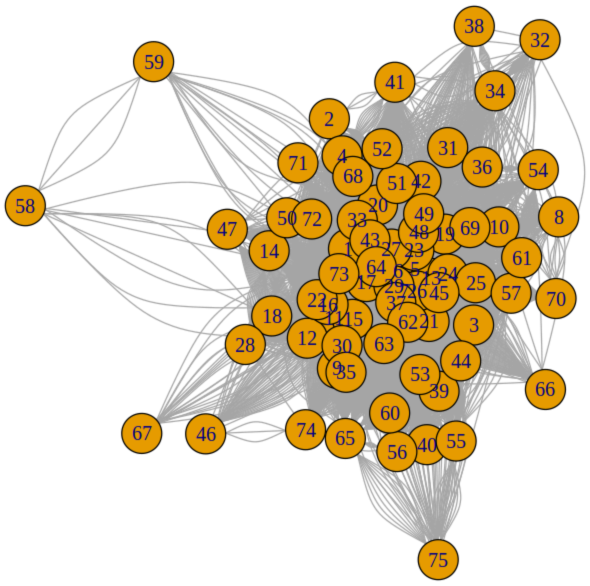
Getting started

Installation of the main packages

In [18]:
```
library(igraph)
library(ggplot2)
library("igraphdata")
data(karate)
data(UKfaculty)
data(USairports)
data(enron)
data(kite)
data(macaque)
data(rfid)
```

```
This graph was created by an old(er) igraph version.
  Call upgrade_graph() on it to use with the current igraph version
  For now we convert it on the fly...
```



This lab is divided in two parts. The first part is dedicated to study the robustness of a network by measuring the size of the giant component, when the network is under attack. The second part is about the Bass model, and how to learn its parameters on real data.

Robustness in network

1/ Create the function Nodes.to.protect(g) which takes a graph object *g* and do the following:

For every node i:

- remove it using the function *delete_vertices()*;
- then compute the size of the largest connected component after the attack using *components()*. We denote this value by r_i.

Let r be the size of the largest connected component before the attack. We define $1 - \frac{r_i}{r}$ to be the rank of node i.

The function will return the vector $[1 - \frac{r_i}{r}]_i$.

Can you explain why this ranking system is pertinent?

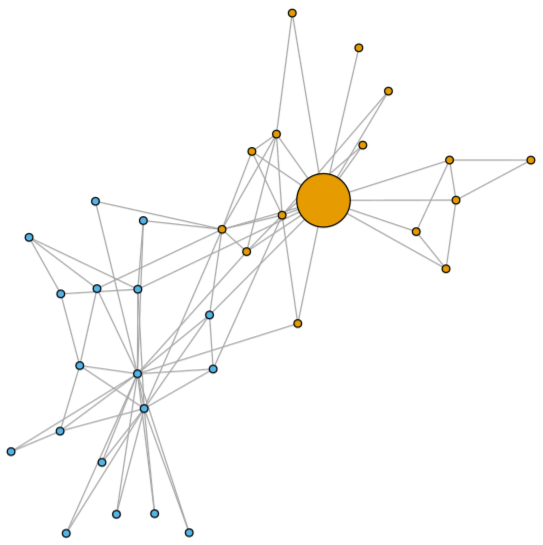
This ranking system allows to find and give more weight to critical nodes. That is, those nodes that, if they were to disappear, would divide the graph into relatively small subgraphs (i.e. the lower r_i is, the more distributed the nodes will be among the new disconnected subgraphs).

In [19]:
```
Nodes.to.protect <-function(g){
    g<-simplify(g)
    A<- as.matrix(as_adjacency_matrix(g))
    l<-dim(A)[1]
    set.to.attack<-c(1:l)
    size.max.components.after.attack<-c()
    r <- max(components(g)$csize)
    for(i in 1:l){
        g2 <- delete_vertices(g, V(g)[i])
        ri <- max(components(g2)$csize)
        size.max.components.after.attack<-c(size.max.components.after.attack
    }
    return(data.frame(set.to.attack,size.max.components.after.attack))
}
```

2/ Plot different networks from the available list (see first cell). In your plot the node size should be based on a scale version of the *Nodes.to.protect* ranking. Study the undirected version of the networks. What do you observe?

In [20]:
```
plot(karate, vertex.size = Nodes.to.protect(karate)[['size.max.components.af
```

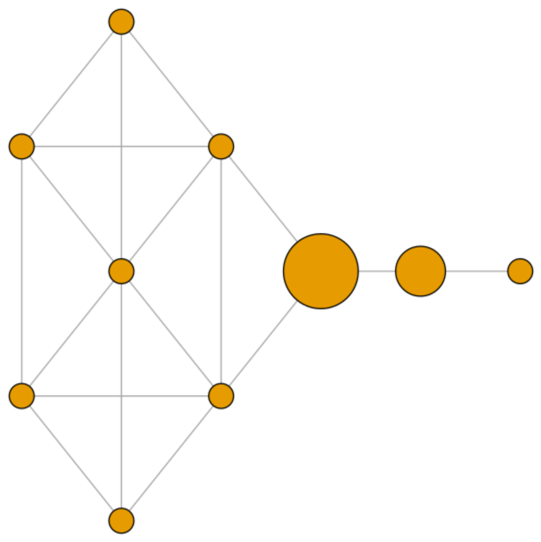
```
This graph was created by an old(er) igraph version.
  Call upgrade_graph() on it to use with the current igraph version
  For now we convert it on the fly...
```



For the karate graph, only one sensitive node is found, which is selected because, if erased, there would be one node left isolated.

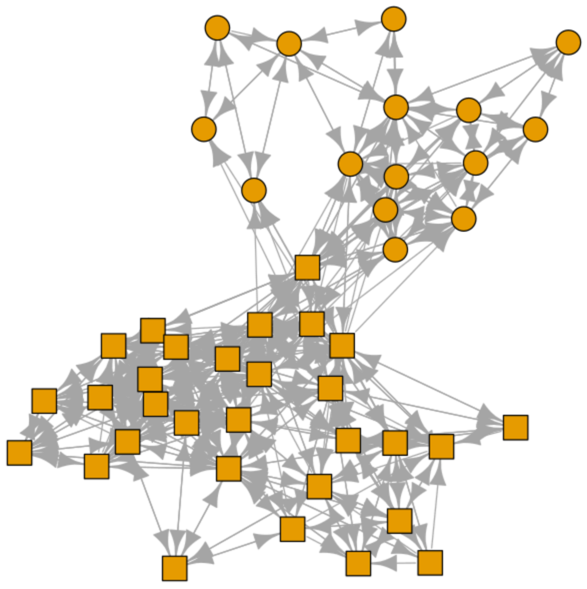
```
In [21]: plot(kite, vertex.size = Nodes.to.protect(kite)[['size.max.components.after.
```

```
This graph was created by an old(er) igraph version.
  Call upgrade_graph() on it to use with the current igraph version
  For now we convert it on the fly...
```



```
In [22]: plot(macaque, vertex.size = Nodes.to.protect(macaque)[['size.max.components.
```

```
This graph was created by an old(er) igraph version.
  Call upgrade_graph() on it to use with the current igraph version
  For now we convert it on the fly...
```



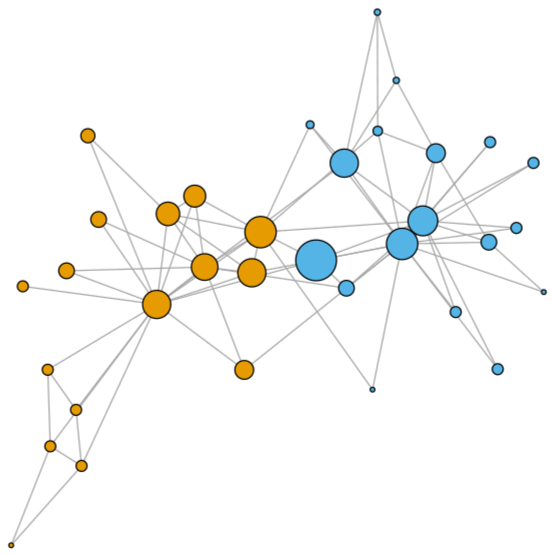
For the macaque graph, which is undirected, all nodes seem to be interconnected so that there are no nodes subdividing the graph into disconnected subgraphs. Therefore, the function Nodes.to.protect can't find any sensitive nodes.

3/ Create the function Nodes.to.protect.2(g) which takes a graph object *g* and instead of only removing one node, it will also remove its neighbors.

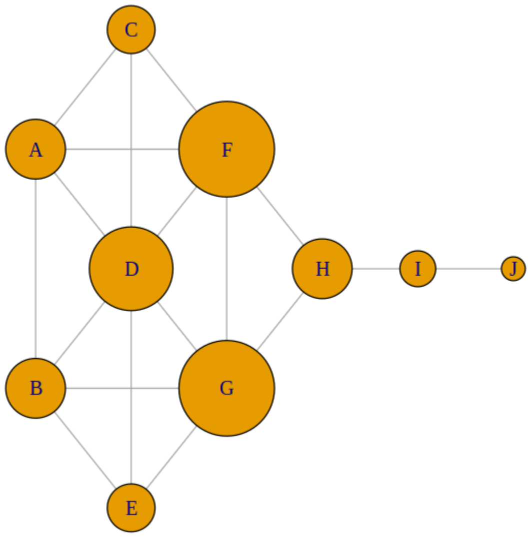
```
In [23]: Nodes.to.protect.2 <-function(g){
    r <- max(components(g)$csize)
    A<- as.matrix(as_adjacency_matrix(g))
    l<-dim(A)[1]
    set.to.attack<-c(1:l)
    size.max.components.after.attack<-c()
    for(i in 1:l){
        set.to.attack.with.neighbors<- union(i, which(A[,i]==1))
        g2 <- delete_vertices(g, set.to.attack.with.neighbors)
        ri <- max(components(g2)$csize)
        size.max.components.after.attack<-c(size.max.components.after.attack
    }
    return(data.frame(set.to.attack,size.max.components.after.attack))
}
```

4/ Try your new function on different networks and plot the results where, again the node size is based on a scale version of the *Nodes.to.protect.2* ranking. What is the difference between the first ranking and the second ranking? Are the ranking the same (in term of order, of course)?

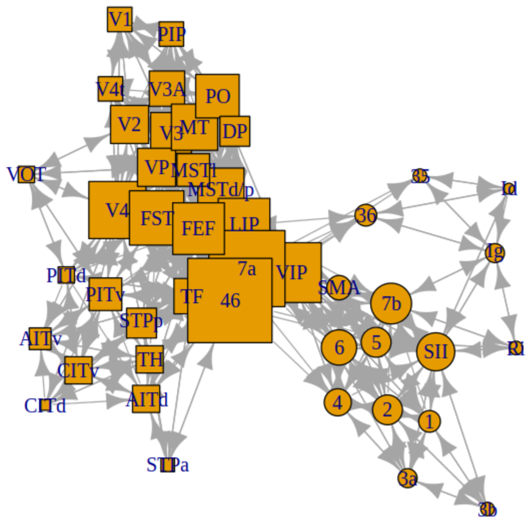
```
In [24]: plot(karate, vertex.size = Nodes.to.protect.2(karate)[['size.max.components.
```



```
In [25]: plot(kite, vertex.size = Nodes.to.protect.2(kite)[['size.max.components.afte
```



```
In [26]: plot(macaque, vertex.size = Nodes.to.protect.2(macaque)[['size.max.component
```



We now observe that more nodes are important in ensuring the robustness of the graph.

5/ There are standards measures, other than the ones that you have created above, to rank nodes in a network. These measures are called centrality measures. The famous ones are:

- degree centrality
- betweenness centrality
- closeness centrality
- eigenvector centrality

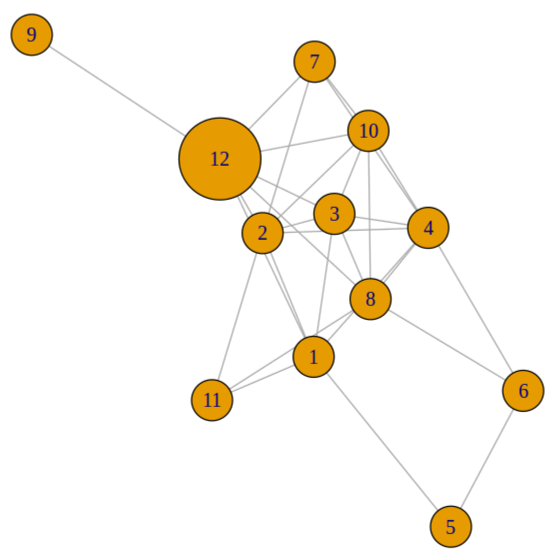
Please try to understand what is the meaning of these different centralities. For example you can have a look to the following videos to have a quick introduction to centrality measures:

- https://youtu.be/0unzqsPaPk8
- https://youtu.be/q8oBWwS2wAQ

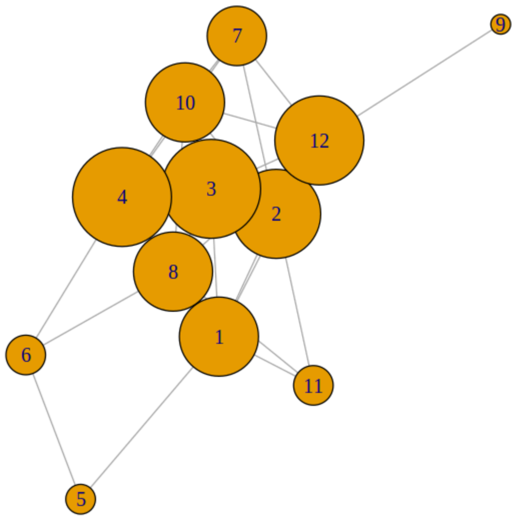
Compare the two ranking measures, that you have designed, with the other centralities. To do so, you can use the spearman correlation. Try different networks, justify ourself. *(You have to write a bit for this question)*.

```
In [85]: plot.graphs.for.centrality.metrics <-function(g){

             # Calculate ranking measures
             result_nodes_to_protect <- Nodes.to.protect(g)
             result_nodes_to_protect_2 <- Nodes.to.protect.2(g)
             degree_centrality <- degree(g)
             betweenness_centrality <- betweenness(g)
             closeness_centrality <- closeness(g)
             eigenvector_centrality <- eigen_centrality(g)$vector

             #print(result_nodes_to_protect)
             closeness_centrality[is.nan(closeness_centrality)] <- 0

             # Plots
             plot(g, vertex.size = result_nodes_to_protect[['size.max.components.afte
             plot(g, vertex.size = result_nodes_to_protect_2[['size.max.components.af
             plot(g, vertex.size = degree_centrality*5, main = "Degree Centrality")
             plot(g, vertex.size = betweenness_centrality*3, main = "Betweenness Cent
             plot(g, vertex.size = closeness_centrality*500, main = "Closeness Centra
             plot(g, vertex.size = eigenvector_centrality*30, main = "Eigenvector Cer
         }

         # Generate a sample network
         g <- erdos.renyi.game(12, p = 0.4)
         g_big <- erdos.renyi.game(100, p = 0.1)
         plot.graphs.for.centrality.metrics(g)
```

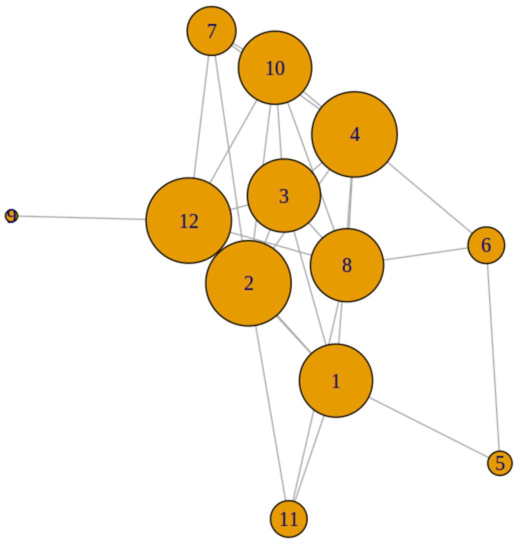
1 node erased



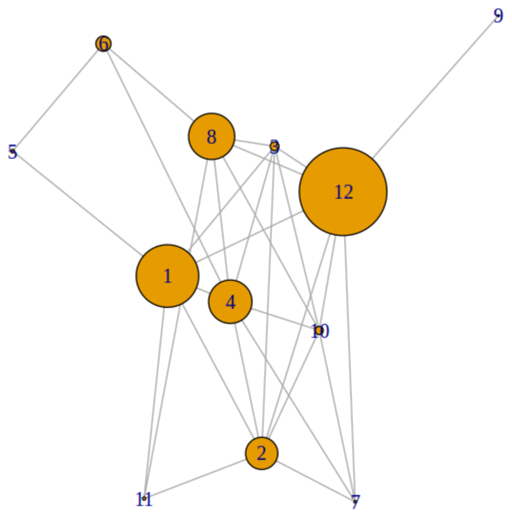
1 node and neighbours erased



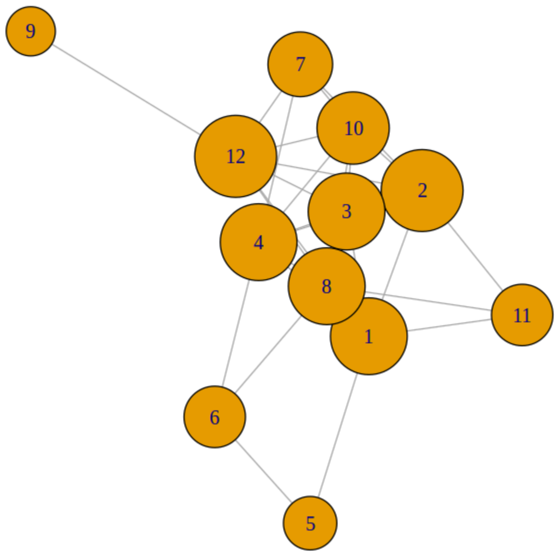
Degree Centrality



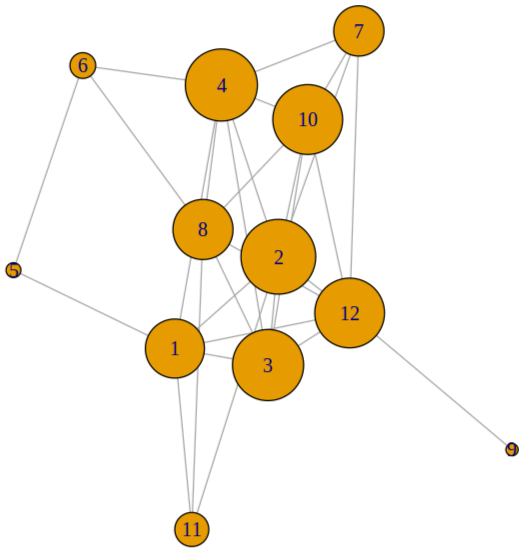
Betweenness Centrality



Closeness Centrality

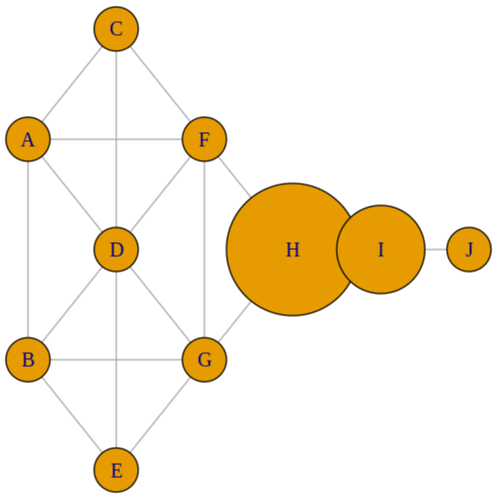


Eigenvector Centrality

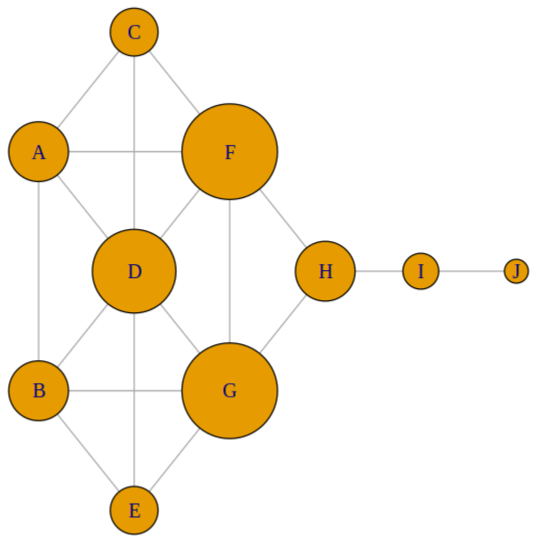


```
In [28]: plot.graphs.for.centrality.metrics(kite)
```

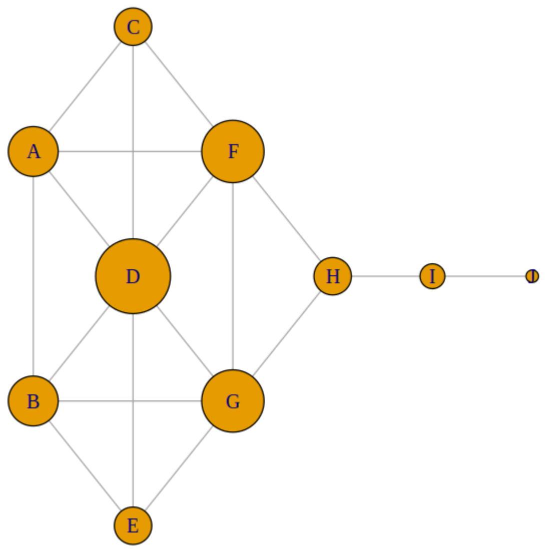
1 node erased



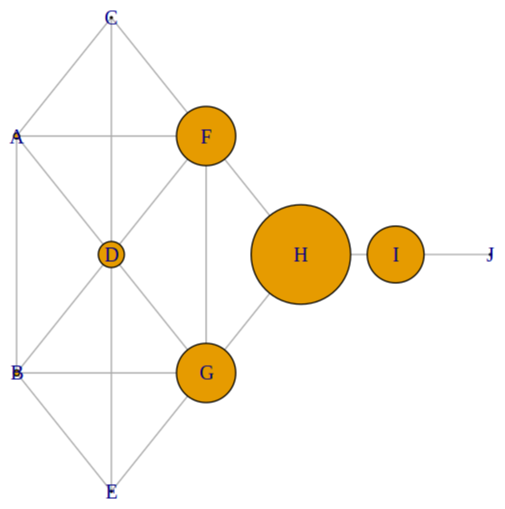
1 node and neighbours erased



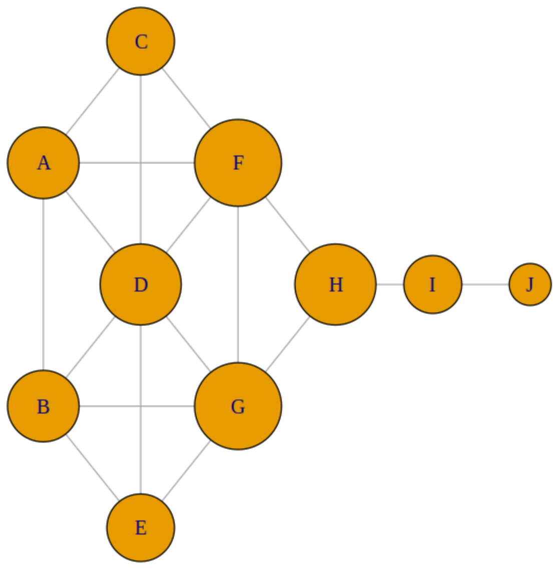
Degree Centrality

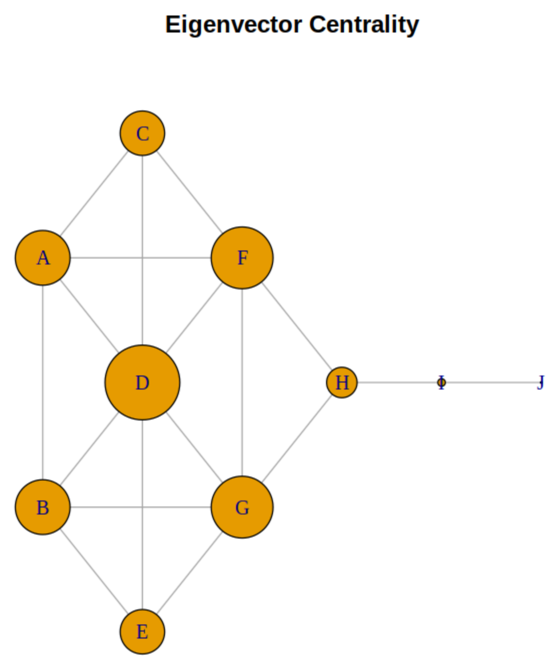


Betweenness Centrality



Closeness Centrality





```
In [105…  print.correlation.matrix.for.centrality.metrics <-function(g){
              # Calculate ranking measures
              result_nodes_to_protect <- Nodes.to.protect(g)
              result_nodes_to_protect_2 <- Nodes.to.protect.2(g)
              degree_centrality <- degree(g)
              betweenness_centrality <- betweenness(g)
              closeness_centrality <- closeness(g)
              eigenvector_centrality <- eigen_centrality(g)$vector

              #print(result_nodes_to_protect)

              # Extract the relevant columns
              size_max_components_after_attack_1 <- result_nodes_to_protect[['size.max
              size_max_components_after_attack_2 <- result_nodes_to_protect_2[['size.m
              #print(size_max_components_after_attack_1)
              #print(cor(size_max_components_after_attack_1, size_max_components_after

              # Combine the ranking measures into a data frame
              ranking_data <- data.frame(
                Nodes_to_protect = size_max_components_after_attack_1,
                Nodes_to_protect_2 = size_max_components_after_attack_2,
                Degree_Centrality = degree_centrality,
                Betweenness_Centrality = betweenness_centrality,
                Closeness_Centrality = closeness_centrality,
                Eigenvector_Centrality = eigenvector_centrality
              )

              # Add a small amount of noise to specific columns to avoid zero standard
              ranking_data$Nodes_to_protect <- ranking_data$Nodes_to_protect + runif(l
              ranking_data$Nodes_to_protect_2 <- ranking_data$Nodes_to_protect_2 + run
```

```
    # Normalize the data
    normalized_data <- scale(ranking_data)

    # Compute Spearman correlation matrix
    cor_matrix <- cor(normalized_data, method = "spearman")

    # Round the correlation matrix to 2 decimals
    cor_matrix_rounded <- round(cor_matrix, 2)

    # Print the rounded correlation matrix
    print(cor_matrix_rounded)
}
```

In [88]:
```
print.correlation.matrix.for.centrality.metrics(g)
```

```
[1] -0.4265734
                       Nodes_to_protect Nodes_to_protect_2 Degree_Centrality
Nodes_to_protect                   1.00               0.27              0.40
Nodes_to_protect_2                 0.27               1.00              0.92
Degree_Centrality                  0.40               0.92              1.00
Betweenness_Centrality             0.48               0.65              0.76
Closeness_Centrality               0.45               0.89              0.96
Eigenvector_Centrality             0.13               0.94              0.92
                       Betweenness_Centrality Closeness_Centrality
Nodes_to_protect                         0.48                 0.45
Nodes_to_protect_2                       0.65                 0.89
Degree_Centrality                        0.76                 0.96
Betweenness_Centrality                   1.00                 0.81
Closeness_Centrality                     0.81                 1.00
Eigenvector_Centrality                   0.54                 0.86
                       Eigenvector_Centrality
Nodes_to_protect                         0.13
Nodes_to_protect_2                       0.94
Degree_Centrality                        0.92
Betweenness_Centrality                   0.54
Closeness_Centrality                     0.86
Eigenvector_Centrality                   1.00
```

In [87]:
```
print.correlation.matrix.for.centrality.metrics(kite)
```

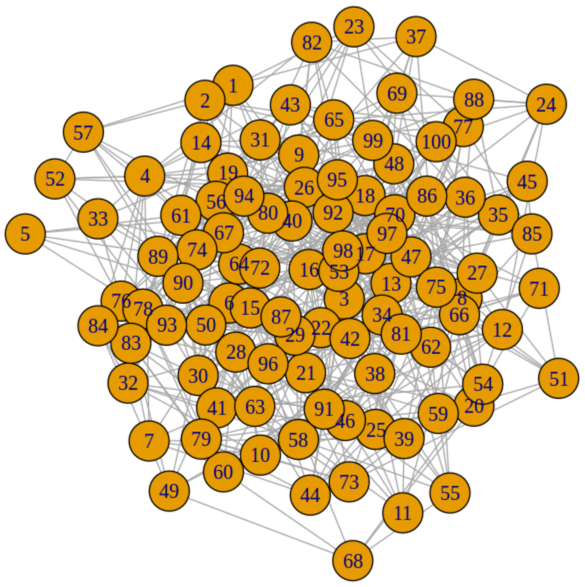
```
[1] 0.1393939
                       Nodes_to_protect Nodes_to_protect_2 Degree_Centrality
Nodes_to_protect                   1.00              -0.23             -0.42
Nodes_to_protect_2                -0.23               1.00              0.93
Degree_Centrality                 -0.42               0.93              1.00
Betweenness_Centrality             0.56               0.64              0.41
Closeness_Centrality              -0.08               0.98              0.86
Eigenvector_Centrality            -0.51               0.89              0.98
                       Betweenness_Centrality Closeness_Centrality
Nodes_to_protect                         0.56                -0.08
Nodes_to_protect_2                       0.64                 0.98
Degree_Centrality                        0.41                 0.86
Betweenness_Centrality                   1.00                 0.73
Closeness_Centrality                     0.73                 1.00
Eigenvector_Centrality                   0.30                 0.79
                       Eigenvector_Centrality
Nodes_to_protect                        -0.51
Nodes_to_protect_2                       0.89
Degree_Centrality                        0.98
Betweenness_Centrality                   0.30
Closeness_Centrality                     0.79
Eigenvector_Centrality                   1.00
```

As can be seen in the correlation matrices, there is a clear tendency:

1. Nodes.to.protect has a significant degree of correlation with betweenness centrality, but not much with the other metrics. This is logical since both Nodes_to_protect and betweenness centrality have some quite strict requirements over nodes. For Nodes.to.protect, for a node to be considered important, it must be in between of at least two significantly populated disconnected regions. At the same time, the betweenness centrality measures how many shortest paths pass through each node, so it does also impose strict requirements over nodes.
2. Nodes.to.protect.2 shows a signficant degree of correlation with degree_Centrality, closeness_centrality and eigenvalue centrality. This is logical since nodes.to.protect.2 erases a node and also its neighbours, meaning that it is no longer so strict over the requirement for each node. However, this method might give more importance to nodes that are not so important to ensure the flow of information, which explains the decrease in correlation with betweeness_centrality.

Attention: used graphs are of small size and might well throw conclusions that are not always true. As the graph becomes bigger, it may happen the correlation between nodes.to.protect.2 and all metrics to decrease, because at some point, even by disconnecting all neighbours of a node it will not be possible to separate the graph into disconnect subgraphs.

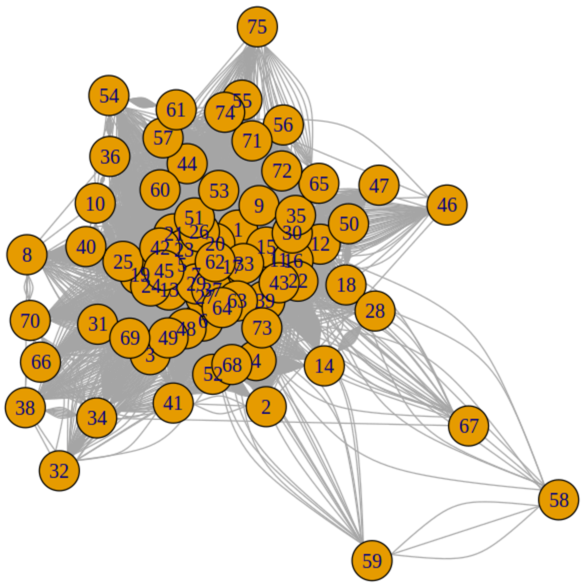
```
In [109…  plot(g_big)
```



```
In [106… print.correlation.matrix.for.centrality.metrics(g_big)
```

```
                       Nodes_to_protect Nodes_to_protect_2 Degree_Centrality
Nodes_to_protect                   1.00              -0.05             -0.06
Nodes_to_protect_2                -0.05               1.00              0.99
Degree_Centrality                 -0.06               0.99              1.00
Betweenness_Centrality            -0.04               0.97              0.98
Closeness_Centrality              -0.03               0.95              0.96
Eigenvector_Centrality            -0.06               0.97              0.97
                       Betweenness_Centrality Closeness_Centrality
Nodes_to_protect                        -0.04                -0.03
Nodes_to_protect_2                       0.97                 0.95
Degree_Centrality                        0.98                 0.96
Betweenness_Centrality                   1.00                 0.94
Closeness_Centrality                     0.94                 1.00
Eigenvector_Centrality                   0.92                 0.95
                       Eigenvector_Centrality
Nodes_to_protect                        -0.06
Nodes_to_protect_2                       0.97
Degree_Centrality                        0.97
Betweenness_Centrality                   0.92
Closeness_Centrality                     0.95
Eigenvector_Centrality                   1.00
```

```
In [110… plot(rfid)
```



```
In [111…  print.correlation.matrix.for.centrality.metrics(rfid)
```

```
                         Nodes_to_protect Nodes_to_protect_2 Degree_Centrality
Nodes_to_protect                     1.00              -0.26             -0.29
Nodes_to_protect_2                  -0.26               1.00              0.30
Degree_Centrality                   -0.29               0.30              1.00
Betweenness_Centrality              -0.27               0.26              0.94
Closeness_Centrality                -0.26               0.52              0.89
Eigenvector_Centrality              -0.28               0.26              0.91
                         Betweenness_Centrality Closeness_Centrality
Nodes_to_protect                          -0.27                -0.26
Nodes_to_protect_2                         0.26                 0.52
Degree_Centrality                          0.94                 0.89
Betweenness_Centrality                     1.00                 0.87
Closeness_Centrality                       0.87                 1.00
Eigenvector_Centrality                     0.85                 0.88
                         Eigenvector_Centrality
Nodes_to_protect                          -0.28
Nodes_to_protect_2                         0.26
Degree_Centrality                          0.91
Betweenness_Centrality                     0.85
Closeness_Centrality                       0.88
Eigenvector_Centrality                     1.00
```

Once again, Nodes_to_protect seems to be useless. However, Nodes_to_protect_2 shows a better behavior, being mostly correlated to Closeness centrality. It does also remain true that the correlation between Nodes_to_protect_2 and Betwenness centrality is the minimum between all metrics. The prediction that the correlation between Nodes_to_protect_2 and all other metrics was going to decrease as the size of the graph increases has been verified in this case (different graph structures might give different results).

Here Nodes_to_protect has almost no information, while nodes_to_protect_2 behaves quite properly, probably because of how many interconnections the graph has.

6/ (Bonus) Try to design and implement a new ranking measure. You have to first:

- Define the objective of our ranking measure (robusteness, importance in a network, etc.);
- Implement your ranking measure;
- Compare your measure with the other ones.

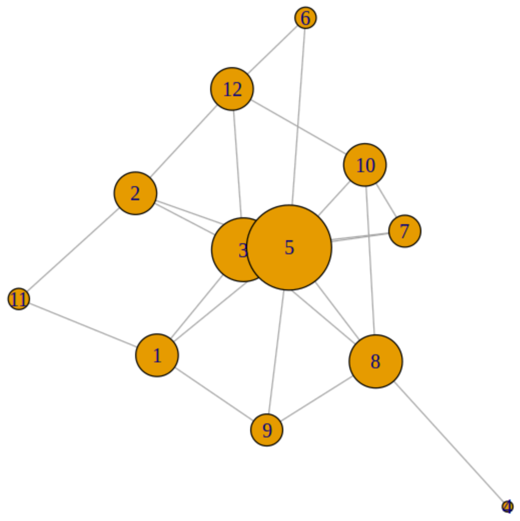
We will try to define a centrality based on a Markov Chain modelling of the graph. Let's suppose that node i will change of state with a probability $\frac{1}{d_i}$ where d_i is the degree of node i. Then, the transition matrix P can easily be obtained from the adjacency matrix A (just divide the ith row by d_i). Once this is done, the stationary distribution of the markov chain can be obtained, which can be used as a measure of the centrality of each node. For this last point, it is necessary to solve $\pi = \pi P$, where π is the stationary distribution.

This type of centrality could be used to model a network where each node behaves symmetrically towards all of its neighbours.

Looking it more in depth, this ranking measure equals the eigenvalue centrality. This is quite logical given the parallelism between the eigenvalue decomposition and Markov Chains.

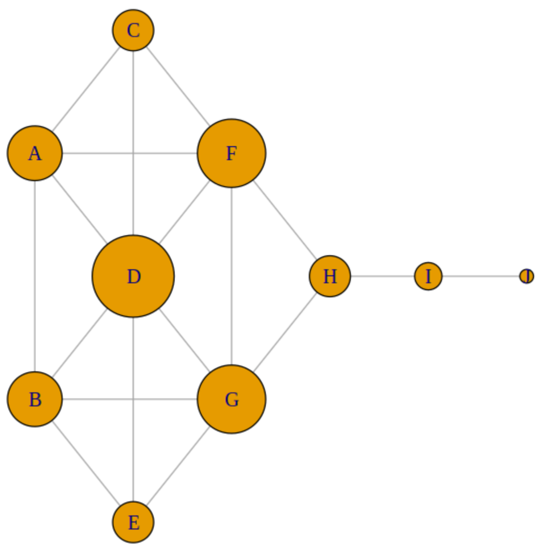
In [32]:
```
library(MASS)

Markov.stationary.distribution <-function(g){
    A <- as.matrix(as_adjacency_matrix(g))
    degrees <- colSums(A)
    l <-dim(A)[1]
    P <- A / degrees

    # https://stephens999.github.io/fiveMinuteStats/markov_chains_discrete_s
    # Get the eigenvectors of P, note: R returns right eigenvectors
    r=eigen(P)
    rvec=r$vectors
    # left eigenvectors are the inverse of the right eigenvectors
    lvec=ginv(r$vectors)
    # The eigenvalues
    lam<-r$values
    # Stationary distribution
    pi <-lvec[1,]/sum(lvec[1,])
    return(data.frame(1:l, Re(pi)))
}
```

In [33]:
```
plot(g, vertex.size = Markov.stationary.distribution(g)[['Re.pi.']]*200)
```

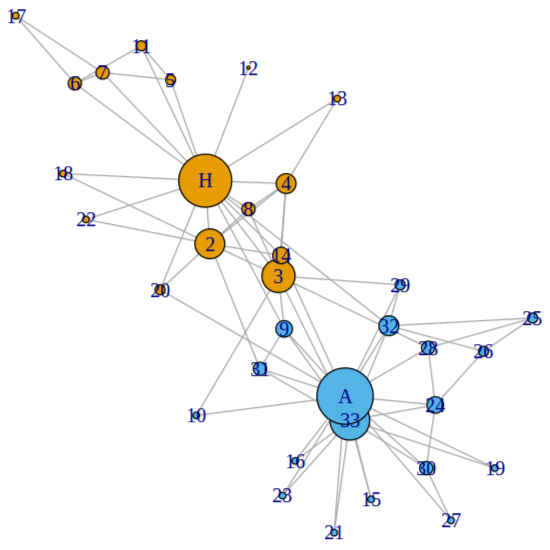


Note that this result is equivalent to the obtained in eigenvalue equality of the previous point.

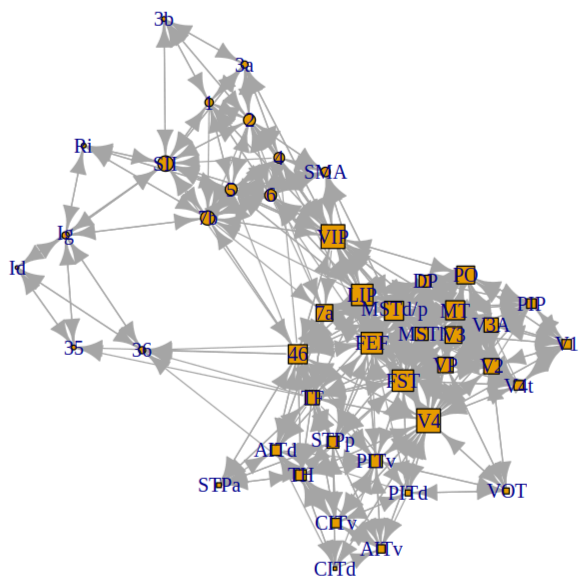
In [34]:
```
plot(kite, vertex.size = Markov.stationary.distribution(kite)[['Re.pi.']]*20
```



```
In [35]: plot(karate, vertex.size = Markov.stationary.distribution(karate)[['Re.pi.']
```

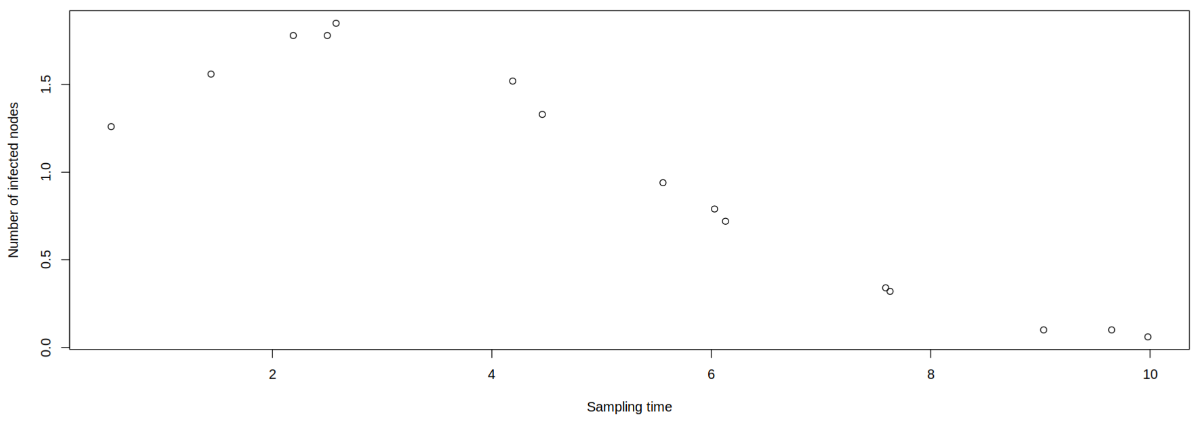


```
In [36]: plot(macaque, vertex.size = Markov.stationary.distribution(macaque)[['Re.pi.
```



Bass model

```
In [12]: df<-data.frame(sampling.time=c(0.53, 1.44, 2.19, 2.5, 2.58, 4.19, 4.46, 5.56
                    Res=  c( 1.26, 1.56, 1.78, 1.78, 1.85, 1.52, 1.33, 0.94, 0.79
         plot(df$sampling.time,df$Res,xlab="Sampling time",ylab = "Number of infecte
```



7/ We define:

- $F(t)$= ratio of people (out of the total population) that has been infected *by* time t;
- $f(t)$= ratio of people (out of the total population) that has been infected *at* time t.

In the Bass model, the probability that a person is infected at time t, is given by $p + qF(t)$.

Parameters: p captures the effect of direct infections and q captures the effect of social network. The Bass model is described by the equation:

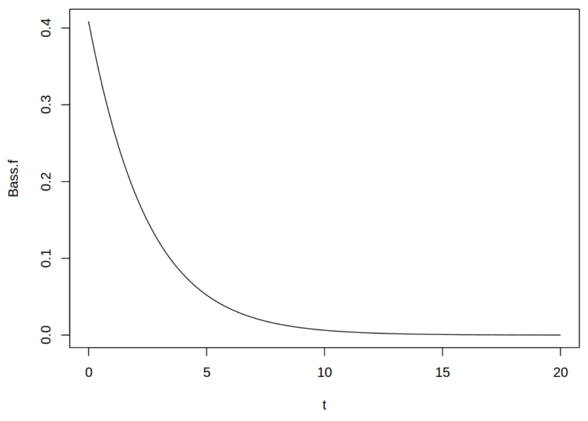
$$\frac{f(t)}{1 - F(t)} = p + qF(t), F(0) = 0.$$

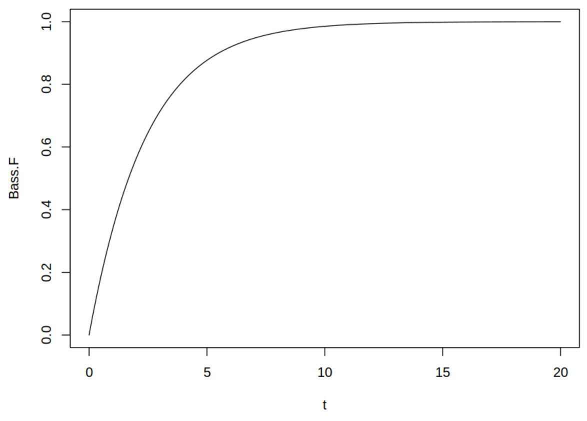
What is the solution of this differential equation? Implement the function *Bass.model(p,q,t.max)* which will plot $f(t)$ and $F(t)$ for a given p, q, and $s \in [0, t.max]$. Plot the two curves with $p = 0.016$, $q = 0.409$ and $t.max = 20$

The solution of this differential equation, where $f(t) = dF(t)/dt$, is given by

$$F(t) = \frac{1 - e^{-(p+q)t}}{1 + pe^{-(p+q)t}/q}$$

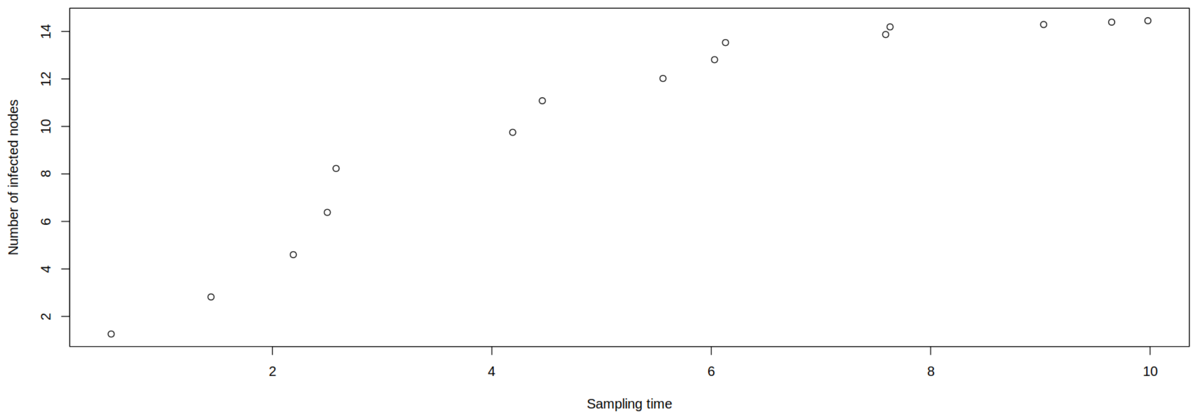
In [13]:
```
Bass.model <- function(p,q,t.max) {
    delta_t <- 0.01
    t <-seq(from=0,to=t.max,by=delta_t)
    Bass.F <- (1 - exp(-(p + q) * t)) / (1 + p * exp(-(p + q) * t) / q)
    Bass.f <- c(diff(Bass.F) / delta_t, NA)  # NA for the last point

    options(repr.plot.width = 15, repr.plot.height = 6)
    par(mfrow=c(1,2))
    plot(t, Bass.f, type="l");
    plot(t, Bass.F, type="l")
}
Bass.model(0.016, 0.409, 20)
```





8/ Estimate parameters of the Bass model using *nls()* function for the data in the data frame *df*. To extract the value of p and q you can use the function *coef()*. Finally compare the actual and the fitted values using ggplot function.

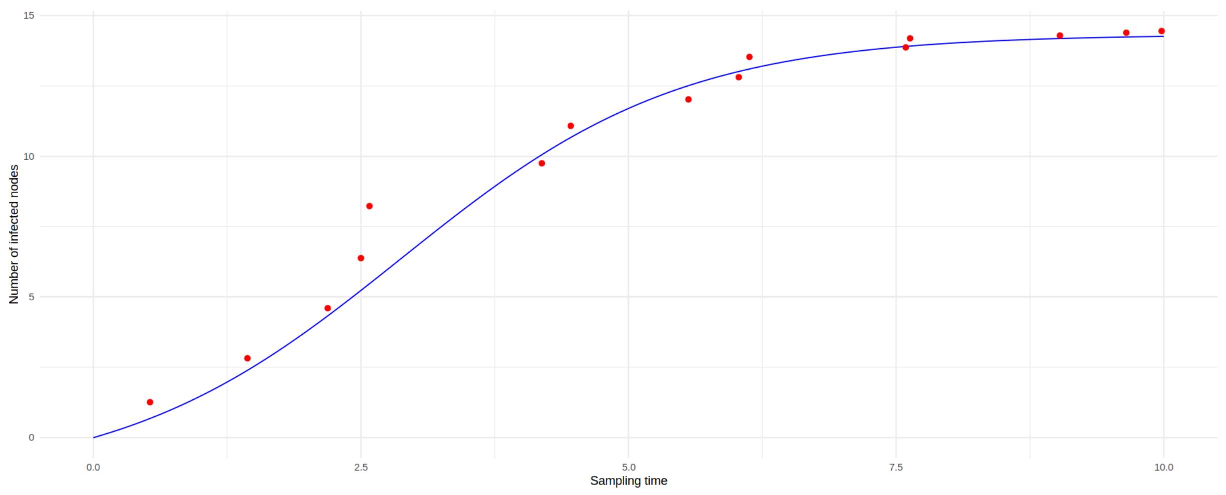
In [25]:
```
# Obtain F
dF<-data.frame(sampling.time=df$sampling.time,
               Res=  cumsum(df$Res))

plot(dF$sampling.time,dF$Res,xlab="Sampling time",ylab = "Number of infected
```



In [57]:
```
# Non-linear least squares fit using nls()
fit <- nls(Res ~ (A*exp((p + q) * sampling.time) - 1) / (exp((p + q) * sampl
           data = dF,
           start = list(A=14, p = 0.4, q = 0.1))

# Extracting parameter values
coefficients <- coef(fit)

# Accessing individual parameter values
A <- coefficients["A"]
p <- coefficients["p"]
q <- coefficients["q"]

print(fit)
```

```
Nonlinear regression model
  model: Res ~ (A * exp((p + q) * sampling.time) - 1)/(exp((p + q) * samplin
g.time) +     p/q)
   data: dF
       A        p        q
14.32787  0.68678  0.07646
 residual sum-of-squares: 6.479

Number of iterations to convergence: 6
Achieved convergence tolerance: 4.254e-06
```

In [67]:
```
library(ggplot2)
t <-seq(from=0,to=10,by=0.01)
Bass.F <- A*(1 - exp(-(p + q) * t)) / (1 + p * exp(-(p + q) * t) / q)

# Create a data frame with the values
fit_df <- data.frame(sampling.time = t, Bass_F = Bass.F)

# Create the ggplot with both lines and scatter plot
ggplot() +
  geom_line(data = fit_df, aes(x = sampling.time, y = Bass_F), color = "blue
  geom_point(data = dF, aes(x = sampling.time, y = Res), color = "red", size
  labs(x = "Sampling time", y = "Number of infected nodes") +
  theme_minimal()
```



The non linear square error method fits the data as expected.