Raul Ossada

Mestrado:
Epidemiologia Experimental Aplicada às Zoonoses, Faculdade de Medicina Veterinária e Zootecnia, USP.

Título da tese de mestrado:
"Modelagem da dinâmica de doenças infecciosas em redes de movimentação de animais".

Orientador: Marcos Amaku.

Links Interessantes
Laboratório de Epidemiologia e Bioestatística
Videos com exercícios de Bioestatística

Meus Exercícios

Link para a página com os meus exercícios resolvidos exec

Proposta de Trabalho Final

Principal

Uma função que simula o espalhamento de uma doença em uma rede estática. Pode-se simular um modelo Suscetível-Infectado (SI) ou Suscetível-Infectado-Suscetível (SIS), dependendo dos parâmetros fornecidos pelo usuário.

Parâmetros da função:
- Matriz de adjacência (Matriz que representa como os nós estão conectados na rede) [M_adj]
- Vetor com o tempo que cada nó ficará infectado (Condição inicial) [tI]
- Probabilidade de um nó ser infectado por outro [pspread]
- Tempo mínimo de infecção [tImin]
- Tempo máximo de infecção [tImax]
- Tempo de simulação [tSim]

Retorna uma matriz onde cada coluna representa os nós que estavam infectados no tempo t(x), x=1,... .

Aplicações dos resultados:
- Cálculo da prevalência num dado momento t(x)
- Número de vezes que um certo nó foi infectado durante a simulação

OBS.: Apesar de descrito como espalhamento de doenças (AIDS, Vírus de Computador, etc), a função também pode ser usada em outras áreas, como por exemplo, para simular o espalhamento de idéias

em redes sociais. Utilizando-se da interpretação do pesquisador.

Comentários da Proposta Principal

Legal, é generalizável. Você pode plotar seu output num gráfico para observar o espalhamento também. Parece factível no prazo.

Gabriel

Comentários do autor sobre a versão final

Acabo de postar a versão final de minha função.
Nela, eu tentei colocar um pouco de cada conceito visto nas aulas da disciplina(desde funções pré-prontas até programação mais bruta).
E atendendo ao comentário do Gabriel eu inclui umas opções mais gráficas. Que poderiam muito bem ser funções separadas da função final. Deu um pouco de trabalho, mas tudo bem, digamos que foi uma promoção pague 1 função e leve 3 hehehe.

Raul

Página de Ajuda

```
simulationSIS                package:bie5782_2011           R Documentation

Simulates the spread of diseases on a static directed network

Description:

     Simulates the spread of diseases on a static directed network,
returning the simulation matrix for user analysis.
     Optionally, the function returns some analysis of the simulation and an
animation representing step by step the
     simulation of the spread on the network, for a more visual analysis.

Usage:

     simulationSIS (M_adj, tI, pspread, tImin, tImax, tSim, analysis=TRUE,
animation=TRUE)

Arguments:

     M_adj:     A numeric matrix. Adjacency matrix which represents the
network. Where M_adj(i,j)=1 if there is a
                conection between two nodes, or 0 with no conection exists.
     tI:        A numeric vector. Vector of the initial 'infected' nodes.
```

```
Where each position represents the t times
                steps that a node will be 'infected' from the beginning of
the simulation.
     pspread:   Numeric value (0<=pspread<=1). The probability that a node
will get 'infected'.
     tImin:     Numeric value (0<=tmin<=tmax). The minimum time that a new
'infected' node will be in this state.
     tImax:     Numeric value (0<=tmin<=tmax). The maximum time that a new
'infected' node will be in this state.
     tSim:      Numeric value (tSim>=1). The t time steps to simulate.
     analysis:  Logical. If TRUE, shows the network and the prevalence; also
returns a list with the simulation matrix,
                the proportion of 'infected' at each time step t, total time
that each node was 'infected', nodes that
                most stayed 'infected', nodes that least stayed 'infected'.
Requires the igraph package.
     animation: Logical. If TRUE, opens the web-browser and shows the
simulation of spread on the network. Requires the
                animation and igraph packages.

Details:

      Although described as a function to simulate the spread of a disease
on a network, its usage can be expanded to
      other areas like social analysis when one would be interested in how
an idea spread on a social network.
      The function requires that the user inputs the network where the
simulation will be done.
      The initial state of each node (in time steps of the 'infected'
state).
      The probability of a node becoming 'infected'. Used in the expression:
      prob_i = 1 - (1-pspread)^(Number of 'infected' nodes conected to node
i)
      The range of the 'infected' state, expressed in time steps [tImin,
tImax]
      The number of times steps to simulate.
      If the option analysis=TRUE, the function automaticaly return some
analysis that might interest the user.
      If the option animation=TRUE, the function generates an animation of
the simulation for visual inspection. (See Warning[*])
Value:

     By default, the function returns:
       M_Sim: Simulation matrix where each entrie represents for how long
will a node be 'infected', at a time step t.

     If analysis=TRUE, returns a list with:
       M_Sim:                  See above.
       proportion_tT (vector): Proportion of 'infected' nodes at a time step
t.
       total_tI (vector):      Sum of the total time that each node was in
```

```
the 'infected' state.
       max_total_tI (vector):  Nodes that stayed the most at the 'infected'
state.
       min_total_tI (vector):  Nodes that stayed the least at the 'infected'
state.
      and prints on screen the network representantion and a graphic with
the proportion of 'infected' nodes at each time step simulated.

Warning:

     This function supposes that you have the animation and the igraph
packages installed.
     Also it considers the input adjacency matrix as a representation of a
directed network, what shouldn't be a problem
     if your input represents an undirected one.
     This function was developed disconsidering autoloops and multiple edges
in the network. So, check your network
     before using this function on it.
     [*] If animation=TRUE, after each simulation it is strongly recommended
that the user cleans his actual working directory, so that no previous
simulation results disturbes the next ones. Also it is recommended
deactivating the animation option for long times of simulation (tSim>100)
which will decrease the processing time.

Author(s):

     Raul Ossada
     raul.ossada@usp.br

References:

     Ossada, R.; Amaku, M.; Grisi-Filho J.H.H.; Ferreira, F. - Modelagem da
dinâmica de doenças infecciosas em redes complexas - Resumos do XXXIII
CNMAC, 2010. p.39
     http://ecologia.ib.usp.br/bie5782/doku.php
     http://www.leb.fmvz.usp.br/

Example:

     # Defining the arguments/parameters.
     M_adj = matrix(data=c(0,1,0,0,0, 0,0,0,0,1, 1,0,0,0,0, 0,1,0,0,1,
1,0,0,0,0), nrow=5, ncol=5);
     tI = c(0, 5, 0, 0, 2);
     pspread = 1;
     tImin = 1;
     tImax = 10;
     tSim = 10;
     analysis=TRUE;
     animation=TRUE;
```

```
     # Executing the simulation.
     TEST = simulationSIS (M_adj, tI, pspread, tImin, tImax, tSim,
analysis=TRUE, animation=TRUE);
     TEST;
```

Código da Função

```
simulationSIS <- function(M_adj, tI, pspread, tImin, tImax, tSim,
analysis=TRUE, animation=TRUE)
{
        # Load the packages igraph and animation
    library(igraph);
    library(animation);

        # Builds the simulation matrix where each row represents a node and
each column represents a time step of the simulation
    M_Sim = matrix(data=NA, nrow=nrow(M_adj), ncol=(tSim+1) );

        # For each time step...
    for( i in 1:tSim )
    {
                # ... assembles the tI vector to the corresponding column of
the simulation matrix
        M_Sim[,i] = tI;

                # Identify the nodes that are at the 'infected' and
'susceptible' state
        I = as.numeric(tI>0);
        S = as.numeric(!I);

                # Multiply the 'infected' vector by the adjacency matrix,
generating the potential infection vector
        Pvector = I %*% M_adj;

                # Vector with the probability of each node getting infected
        Pnode = 1 - (1-pspread)^Pvector;
                # Generating a vector with random numbers based on a uniform
distribution [0,1]
        Prand = runif(n=length(M_adj[1,]), min=0, max=1);

                # If Prand(i) <= Pnode(i), the node becames 'infected'
        Pinfection = as.numeric(Prand<=Pnode);

                # Creating the new 'infected' vector: S=(0,1) AND
Pinfection=(0,1)
        Inew = as.numeric(S & Pinfection);

                # Update the 'tI vector' decreasing 1 for each element in
it, denoting the passage of 1 time step
```

```
        temp_tI = tI-1;
                # Check if any node got a negative time duration and if that
happened turn it back to zero
        tInew = as.numeric(temp_tI<0) + temp_tI;

                # Give a random duration to the new infected nodes
        aux_tI = rep( 0, length(Inew) );
                # Check which model is been simulated (SIS or SI)
        if( tImin != tImax )
        {
            aux_tI[ Inew>0 ] = sample(x=(tImin:tImax), size=length(
which(Inew!=0) ), replace=TRUE);
        }
        else
        {
            aux_tI[ Inew>0 ] = rep(x=tImax, times=length( which(Inew!=0) )
);

        }
                # Merge the new infected nodes with the new ones
        tI = tInew + aux_tI; tI
    }
        # Inputs the results of the last time step of the simulation, in the
simulation matrix
    M_Sim[,i+1] = tI;

        # Assigns the simulation matrix to the variable to be returned, case
no options were selected
    answer = M_Sim;

        # If any optional was selected, make the preparations
    if(analysis==TRUE || animation==TRUE)
    {
                # Name the rows and columns of the adjacency matrix
        rownames(M_adj) = 1:nrow(M_adj)
        colnames(M_adj) = 1:ncol(M_adj)
                # Transforms the adjacency matrix into an igraph network
object
        igraph_network = graph.adjacency(M_adj);
                # Generates coordinates for plotting the network
        coordinates = layout.kamada.kawai(igraph_network);
                # Makes a matrix where each non-zero position of the
simulation matrix is equal to 1 and 0 otherwise.
        M_Result = matrix(data=as.numeric(M_Sim>0), nrow=nrow(M_Sim),
ncol=ncol(M_Sim) );
    }
    if(analysis==TRUE)
    {
                # Plot the network, for visual inspection of the user
```

```
        plot( x=igraph_network, layout=coordinates, vertex.size=10,
vertex.label=rownames(M_adj), edge.arrow.size=0.6, main="Network Plot" )
                # Calculates the proportion of 'infected' nodes at each time
step
        infected_tT = apply(X=M_Result, MARGIN=2, FUN=sum);
        proportion_tT = infected_tT/nrow(M_Sim);
                # Opens a new device and plots the proportion of the
'infected' nodes at each time step
        x11();
        plot(proportion_tT, pch=16, cex=0.5, xlab="Time (t)",
ylab="Proportion", main="Proportion x Time" );
        lines(x=1:(tSim+1), y=proportion_tT);

                # Calculates for how long each node stayed 'infected'
        total_tI = apply(X=M_Result, MARGIN=1, FUN=sum);
                # Gets the nodes that stayed 'infected' longer and less
        max_total_tI = which(total_tI==max(total_tI));
        min_total_tI = which(total_tI==min(total_tI));

                # Assigns the previous results on a list
        analysis_list = list( M_Sim, proportion_tT, total_tI, max_total_tI,
min_total_tI );
        names(analysis_list) = c( "Simulation Matrix", "Proportion of
'infected' at time(t)", "Total time that each node got 'infected'", "Nodes
that most stayed 'infected'", "Nodes that least stayed 'infected'" );

                # Assigns the list to be returned to the user
        answer = analysis_list;
    }

    if(animation==TRUE)
    {
                # Assigns the colors to be used on the animation 2-
red=infected, 3-green=susceptible
        M_Sim_color = -1 * M_Result + 3;

                # Save the images for the animation on a web-browser
        saveHTML(
        {
                        # Plot the network simulations
            for( i in 1:(tSim+1))
            {
                plot( x=igraph_network, layout=coordinates, vertex.size=10,
vertex.label=rownames(M_adj), vertex.color=M_Sim_color[,i],
edge.arrow.size=0.6, main="Animation of the spread on the network" )
                ani.pause();
            }

        }, ani.width=800, ani.height=800, outdir=getwd(),
imgdir="network_dir", img.name="network_plot",
htmlfile="network_animation.html", title="Animation of the spread on the
```

```
network", verbose=FALSE, autobrowse=TRUE, autoplay=FALSE );
    }
        # Return the answer asked by the user
    return( answer );
}
```

Arquivo da Função

simulationSIS.r