Catene di Markov

iNDiCE

iNTRODUZiONE

Le catene di Markov sono una classe di processi randomici(stocastici) , cioè processi dove il comportamento e lo stato vengono individuati da variabili aleatorie(più comunemente viene individuato come processo stocastico una famiglia di variabili aleatorie caratterizzate da certe proprietà), chiameremo le variabili aleatorie {Xt} collegate alle catene di Markov come stati.

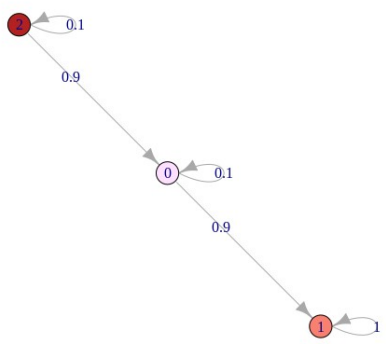
in particolare le catene di Markov sono caratterizzate dall'importantissima proprietà per cui lo stato futuro considerato {Xt} dipende soltanto dallo stato presente $\{X_n\}$ e non dagli stati precedenti a quello corrente.
Formalmente:
Proprieta' di Markov:Dato lo stato corrente X_t , la distribuzione di ogni
stato futuro X_y (con $y > t$), non dipende dalla sua storia passata, $\{X_u : u < t\}$, ma solo da
quello presente.
Concentrerò la mia implementazione ed analisi sulle catene di Markov a tempo discreto di primo ordine, dove uno stato X_t dipende soltanto dallo stato precedente X_{t-1}.
Forniro anche lo pseudo-codice per trovare certe proprietà delle catene di Markov e l'intuizione che sta dietro l'implementazione.

Durante questa relazione, tento di visitare il concetto delle Markov chain e implementarle in un linguaggio molto usato per la rappresentazione di dati e l'analisi dei dati(R), includerò pseudocodice e linkero le implementazioni e librerie che uso durante il processo.

Verso la fine della relazione presenterò come esempio una situazione attuale (lo sviluppo e il contagio di virus, in particolare visiterò il concetto di Corona Virus) e tenterò(molto accennato) di stabilire un framework di costruzione di modello , e accennerò come è possibile ricavare un modello tramite inferenza sui simboli(dati reali) generati usando il modello di learning HMM.

1. PRESENTAZiONE E RAPPRESENTAZiONE DELLE MARKOV CHAiN

La rappresentazione grafica delle catene di Markov è importante per capire alcune proprietà della catena e studiare talvolta le caratteristiche del grafo(ricerca di motif, cioè isomorfismi, analisi di hub, significatività statistica delle proprietà con creazione di modelli di markov randomici, etc...), può essere semplicemente implementata tramite un automa a stati finiti(nel caso in cui il numero di stati della nostra catena di markov sia finito), dove i passaggi di stato sono le probabilità di transizione.
Ovviamente come già accennato noi ci concentreremo sullo studio di catene a stati preferibilmente finiti e a tempo discreto.
Quindi rappresentiamo la catena come un grafo, di seguito riporto un esempio:



Nel grafo che rappresenta la matrice di transizione abbiamo tre stati S={1,2,3}.
Le probabilità di transitare da uno stato all'altro sono rappresentate dagli archi direzionati e pesati da un nodo all'altro(che può essere lo stesso o no), ovviamente la somma dei pesi degli archi uscenti da uno stato(nodo) deve essere uguale a 1 altrimenti non sarebbe un modello di markov valido.
Ovviamente possiamo pure rappresentare le probabilità degli stati iniziali.

La rappresentazione della catena di markov è fatta attraverso una libreria per la trattazione di grafi chiamata igraph, il codice per la rappresentazione di seguito:

```
g <- graph(edges = c("2","2" ,"2","0" ,"0","0" ,"0","1" ,"1","1"))
E(g)$weight <- c(0.1,0.9,0.1,0.9,1)
plot(g,edge.label = E(g)$weight)
```

Per essere indipendenti dai dati ed evitare ridondanze useremo la funzione di libreria plot overloadata per la rappresentazione di markov_chain.

1.1 iMPLEMENTAZiONE DELLA MATRiCE Di TRANSiZiONE, iNSiEME DEGLi STATi E STATi iNiZiALi:

Una markov chain è identificata univocamente dalla tripla(S,Mtrans,stat_iniziali), quindi dobbiamo trovare un modo di rappresentarla nel codice.
Molto semplicemente possiamo rappresentare gli stati come un vettore o una lista di stati, la matrice di transizione ovviamente come una matrice quadrata semplice(oppure come una lista di adiacenza data l'analogia con i grafi, ma useremo principalmente la matrice di adiacenza poichè la usano nella maggior parte degli algoritmi), e le probabilità di stati iniziali come un vettore dove le componenti devono essere comprese tra [0,1] e la somma delle componenti deve essere 1.

1.2 iMPLEMENTAZiONE DEL CALCOLO DELLO STATO AD UN TEMPO STABiLiTO
Come già detto, siamo interessati a markov_chain a tempo discreto (t ϵ N), la proprietà di markov viene definita formalmente come:

$P(X_{n+1} = j|X_0 , X_1, ..., X_n) = P(X_{n+1} = j|X_n)$.

inoltre durante il nostro studio consideriamo soltanto catene temporalmente omogenee dove la probabilità di transizione da uno stato ad un altro non dipende dal tempo in cui avviene, quindi.

$q_{i,j}(n) = P(X_{n+1} = j|X_n = i) = q_{i,j}$ *i, j* stati e *n* il tempo in cui avviene la transizione

è per questo che possiamo rappresentare la matrice di transizione come quadrata, se mancasse l'omogeneità dovremmo considerare una matrice per ogni tempo (matrice multidimensionale a 3 dimensioni)

quindi come già detto rappresentiamo la matrice di transizione, le probabilità di occupazione degli stati iniziali come vettori , e gli stati come liste o come vettori.

Di seguito l'esempio già visto sopra di implementazione in R con igraph:

```
transition.matrix2 <-matrix(c(0.1,0.9,0,0,0.1,0.9,0,0,1),nrow = 3,byrow = TRUE)

states <- colnames(transition.matrix2) <- rownames(transition.matrix2) <- c(2,0,1)

initialstate <- c(0.6,0.3,0.1)
```

Come già specificato la somma degli elementi di una riga deve essere uguale a 1, formalmente:

$\sum_{j=0,...\infty} q_{i,j} = 1$

quindi durante l'implementazione dobbiamo tener conto di questo dettaglio, costruendo un costruttore che controlli la somma di ogni riga, pseudocodice

```
markovchain(matrix transition_matrix,vector initial_state,list states){
        if(dim(states) != rows_dim(transition_matrix)  && ...){
                //gestione dell'errore ed error handler, per avvisare dell'errore
        }

        for(ogni riga i della transition matrix){
                let cont=0;
                for(ogni colonna j della riga){
                        cont += transition_matrix[i][j];
                }
                if(cont !=1)  {
                        //gestione dell'errore per la riga i
                }
         }
        //controllo degli stati iniziali e inizializzazione degli attributi della markov chain
        //tramite i parametri
}
```

Ovviamente queste non sono le uniche implementazioni, per esempio potremmo sfruttare le proprietà intrinseche di parallelismo tra i controlli e i dati, ma questo studio differisce di molto dall'implementazione primaria di base che stiamo portando avanti.

Una possibile implementazione in R è la seguente:

```
setClass("Markov_chain", slots=list(transition_matrix="matrix", initial_state="numeric", states="character"))
Markov_chain <- function(transitmatrix, initstate,stat){
 for(i in 1:nrow(transitmatrix)){
  cont <- 0
  for(j in 1:ncol(transitmatrix)){
   cont <- cont + transitmatrix[i,j]
  }
  if(cont <= 0.99 || cont >=1.01)stop("la matrice non è stocastica")
 }
 cont <- 0
 for(i in 1:length(initstate)){
  cont <- cont + initstate[i]
 }
 if(cont <= 0.99 || cont >=1.01)stop(paste("gli stati iniziali non sommano a 1 ma a " ,as.character(cont),sep = " "))
 new("Markov_chain",transition_matrix = transitmatrix, initial_state = initstate, states = stat)
}
```

1.3 CALCOLO DI LEGGI CONGIUNTE

Per sapere quale è la probabilità di transizione partendo da uno stato i ad uno stato *j* dopo un tempo *t* possiamo utilizzare vari metodi di ottimizzazione della moltiplicazione tra matrici, come la scomposizione spettrale o metodi algoritmici che ottimizzano, oppure sfruttare il parallelismo del calcolo dei singoli valori della matrice ed utilizzare metodi di calcolo su GPU o parallel processing, per esempio un algoritmo parallelo sarebbe il seguente (naive parallelism senza ottimizzazioni).

indici dei processori che lavorano parallelamente vanno da (0,0) a (dim(row),dim(col))

```
Kernel matrix_multi(out matrix product,in matrix A,in matrix B,dimcolsA,dimrowsA){
        //indice dell'elemento da calcolare cioè processing element lavorerà su (dim1,dim2)
        let cont=0
        for(ogni colonna i della matriceA riga dim1){
                for(ogni riga j della matricheB colonna dim2 ){
                        cont += A[dim1][i]*B[j][dim2]
                }
        }
        product[dim1][dim2]=cont
}
```

e come già accennato questa è una implementazione naive senza ottimizzazioni, quali accesso alla memoria locale, vettorizzazione, o logiche di velocizzazione del calcolo.

Una semplice implementazione in R è la seguente:

```
legge_congiunta <- function(markovch,temp = 2,istate=0,jstate=0){
 if(class(markovch)!="Markov_chain")stop("devi passare una markov_chain")
 tempmat <- markovch@transition_matrix
 for(i in 1:temp){
  tempmat <- tempmat %*% markovch@transition_matrix
 }
 if(istate==0 || jstate==0){
  tempmat
 }
 else{
  tempmat[istate,jstate]
 }
}
```

1.4 EVOLUZiONE TEMPORALE

Come già sappiamo la legge congiunta e la catena di markov sono univocamente determinate sapendo l'occupazione degli stati iniziali.
Dalla teoria sappiamo che al tempo n = 0 la variabile aleatoria X 0 può assumere i valori (1,2,...,m) con probabilità
$v_1 = P(X_0 = 1)$,
$v_2 = P(X_0 = 2)$,
.....,
$v_m = P(X_0 = m)$

Affinche' questa sia una legge di probabilita' dovranno essere soddisfatte le condizioni
$v_k \geq 0 \; \forall k =, 1, 2....$
$\sum_{k \in E} v_k = 1$
queste condizioni sono soddisfatte dalla mia implementazione del costruttore.
Per calcolare le probabilità di occupazione ad un tempo stabilito n dobbiamo definire formalmente:
$w_k = P(X_n = k) = \sum_{h \in E} P(X_n = k | X_0 = h) P(X_0 = h) = \sum_{h \in E} q^{(n)}_{hk} \; v_h$
cioè i due vettori di occupazione stato a tempo n e occupazione stati iniziali sono legati dalla seguente legge:
$w = vQ^{(n)} = vQ^n$

Lo pseudocodice della legge congiunta aggiornata è il seguente:

```
legge_congiunta_occupazione(class Markov_chain markovch,temp ,istate,jstate){
 Let tempmat = markovch.transition_matrix
 for(i in 1:temp){
  tempmat = matrix_multy(tempmat , markovch.transition_matrix)
 }
 if(istate==0 || jstate==0){
  Return matrix_multy(markovch.initial_state , tempmat)
 }
 else{
  Return (matrixmulty(markovch.initial_state , tempmat))[jstate]
 }
}
```

Possiamo aggiornare la nostra implementazione di legge congiunta nel seguente modo:

```
legge_congiunta_occupazione_stati <- function(markovch,temp = 2,istate=0,jstate=0){
 if(class(markovch)!="Markov_chain")stop("devi passare una markov_chain")
 tempmat <- markovch@transition_matrix
 for(i in 1:temp){
  tempmat <- tempmat %*% markovch@transition_matrix
 }
 if(istate==0 || jstate==0){
  markovch@initial_state %*% tempmat
 }
 else{
  (markovch@initial_state %*% tempmat)[jstate]
 }
}
```

quindi la probabilità di occupazione al tempo n (opzionalmente per un singolo stato) è stata implementata.

Se volessimo sapere la probabilità congiunta di più stati consecutivi temporalmente(cioè volessimo sapere la probabilità della sequenza degli stati ai tempi stabiliti) definita formalmente come segue:

$$\mathbb{P}(X_{n_1} = i_1, X_{n_2} = i_2, ..., X_{n_k} = i_k) = \sum_k v_k q_{ki_1}^{(n_1)} q_{i_1 i_2}^{(n_2 - n_1)} ....q_{i_{k-1} i_k}^{(n_k - n_{k-1})}$$

dove n1...nk sono i tempi temporalmente crescenti. Per calcolare questa probabilità sviluppiamo un algoritmo, lo pseudocodice è il seguente:

```
legge_congiunta_occupazioneN(markov_chain markovch,vector stati,vector tempi,numstati){
        let sum = 0
        for(k in 1:numstati){
                let mult=0
                mult+=stati[k]*legge_congiunta(markovch,tempi[1],k,stati[1])
                for(j in 2:numstati){
                        mult*=legge_congiunta(markovch,tempi[j]-tempi[j-1],stati[j-1],stati[j])
                }
                sum+=mult
        }
        Return sum
}
```

2. CARATTERiSTiCHE DELLE CATENE Di MARKOV

Ci sono molte caratteristiche che possono tornare utili durante l'analisi e lo sviluppo di catene di markov, nei seguenti sottoparagrafi tento di costruire degli algoritmi per il calcolo e la visione di queste proprietà.

2.1 Comunicazione tra due stati

si dice che due stati comunicano se esiste $i,j \in E$ (ovvero $i \rightarrow j$) se $\exists\ n > 0 \mid q_{i,j}^{(n)} > 0$.

cioè nel grafo delle transizioni esiste un cammino che porta dallo stato i allo stato j. Questa proprietà non è simmetrica.

Per trovare un algoritmo possiamo intraprendere due strade, quella statistica e quella algoritmica. Usando quella statistica possiamo calcolare la matrice di transizione finchè non troviamo che il valore di $q_{i,j}^{(n)} > 0$. sia maggiore di 0, oppure dire che i due stati non comunicano(se arriviamo al tempo t, dove t sono gli stati, e ancora non siamo arrivati allo stato desiderato)

Usando un approccio algoritmico, e pensando alla matrice di transizione come un grafo, dobbiamo semplicemente trovare un cammino dal nodo i al nodo j, cioè il nodo j sia raggiungibile dal nodo i (usando algoritmi come BFS o DFS).

i due approcci sono equivalenti quindi possiamo usare o uno o l'altro.
Un possibile algoritmo per vedere al comunicazione tra due stati è il seguente:

```
communication(markovch,istate,jstate){
        for(i in 1:dim(markovch.states)){
                if(legge_congiunta(markovch,i,istate,jstate) > 0) return TRUE
        }
        return FALSE
}
```

il codice in R è il seguente:

```
communication <- function(markovch,istate,jstate){
 for(i in 1:length(markovch@states)){
  if(legge_congiunta(markovch,i,istate,jstate) > 0) return (TRUE)
 }
 FALSE
}
```

2.2 Sottoinsieme di stati **chiuso**

Un sottoinsieme di stati C si dice chiuso se gli stati di C non comunicano con gli stati che stanno fuori di C.

in pratica dobbiamo vedere se gli stati in un sottoinsieme non comunicano con gli stati nel suo complementare all'interno della catena di markov, una possibile implementazione è la seguente:

```
closure_subset(Markov_chain markovch,subset){
        for(i in subset){
                for(j in markovch.states){
                        if(j not in subset){
                                if(communication(markovch,i,j)==TRUE) return FALSE
                        }
                }
        }
        return TRUE
}
```

una possibile implementazione in R è la seguente

```
closure_subset <- function(markovch,subset){
 for(i in subset){
  for(j in markovch@states){
   if(!(j %in% subset)){
    if(communication(markovch,i,j)==TRUE) return(FALSE)
   }
  }
 }
 TRUE
}
```

2.3 insieme irriducibile di stati

Un sottoinsieme di stati chiuso C si dice irriducibile se tutti i suoi stati comunicano tra loro.

Un possibile algoritmo per calcolare l'irriducibilità è il seguente:

```
irreducible_subset(Markov_chain markovch, subset){
        if(!closure_subset(markovch,subset)) return false//il sottoinsieme di stati non è chiuso
        for(i in markovch.states){
                for(j in markovch.states){
                        if(i!=j){
                                if(communication(markovch,i,j)== FALSE) Return false
                        }
                }
        }
        return TRUE
}
```

una possibile implementazione in R è la seguente:

```
irreducible_subset <- function(markovch,subset){
 if(! closure_subset(markovch,subset))stop("il sottoinsieme non è chiuso")
 for(i in subset){
   for(j in subset){
     if(i!=j){
       if(communication(markovch,i,j)== FALSE) return(FALSE)
     }
   }
 }
 TRUE
}
```

Una catena di markov si dice irriducibile se tutti gli stati comunicano tra di loro, ovvero E(insieme di tutti gli stati della catena) è una classe irriducibile

2.4 Stato assorbente

Uno stato si dice assorbente se da solo costituisce una classe irriducibile, cioè non comunica con altri stati nella catena di markov.

Algoritmicamente dobbiamo soltanto controllare che $q_{i,i}=1$ per lo stato i

```
absorb(Markov_chain markovch,state){
        //calcolo dello state index
        if(markovch.transition_matrix[state_index][state_index]==1){
                return TRUE
        }
        return FALSE
}
```

implementazione in R:

```
absorbing_state <- function(markovch,state){
  index <- get_index_state(markovch,state)
  if((markovch@transition_matrix[index,index])!=1){FALSE}
  else {TRUE}
}
```

2.5 Stato ricorrente

Un stato i di una catena di Markov si dice ricorrente se il sistema essendo stato una volta in i, la catena ritornera' in i con probabilita' uno.

Algoritmicamente vogliamo vedere se uno stato è raggiungibile dal nodo in cui stiamo testando la ricorrenza, allora deve esistere un percorso inverso che riporta al nodo, se non esiste questo percorso significa che lo stato è transitorio.

Un algoritmo per calcolare la ricorrenza di un nodo è il seguente:

```
recurrent_state(markovch,state){
   for(i in markovch.states){
     if(communication(markovch,state,i)){
       if(!communication(markovch,i,state))return FALSE
     }
   }
  Return TRUE
}
```

il codice in R è il seguente:

```
recurrent_state <- function(markovch,state){
   for(i in markovch@states){
     if(communication(markovch,state,i)){
       if(!communication(markovch,i,state))return(FALSE)
     }
   }
  TRUE
}
```

2.6 Stato periodico

Un stato j di una catena di Markov si dice periodico con periodo m >1, se i ritorni consecutivi allo stato j avvengono solamente con multipli di m passi, cioè

$$P(X_{n+s} = j | X_n = j) = 0$$

per s >1 , s=mk , k>=1

algoritmicamente siamo interessati a vedere quando nella matrice di transizione la probabilità di passare dallo stato testato per essere periodico allo stesso in un tempo definito sia maggiore di 0.

Di seguito l'algoritmo per il calcolo del periodo e per vedere se uno stato è periodico o no:

```
periodic_state_period(markovch,state){
  for (i in 1:length(markovch.states)) {
    if(legge_congiunta(markovch,i,state,state)) return i
  }
  Return -1
}

periodic_state(markovch,state){
  if(periodic_state_period(markovch,state)<=1)return(FALSE)
  TRUE
}
```

Una possibile implementazione in R è la seguente:

```
periodic_state_period <- function(markovch,state){
  for (i in 1:length(markovch@states)) {
    if(legge_congiunta(markovch,i,state,state))return(i)
  }
  -1
}

periodic_state <- function(markovch,state){
  if(periodic_state_period(markovch,state)<=1)return(FALSE)
  TRUE
}
```

Se non esistono stati periodici allora la catena si dirà aperiodica.

2.7 Matrici di transizioni regolari

Definiamo legge di probabilità stazionaria su E una legge per cui valga la seguente proprietà:

v=vQ

quindi la legge di probabilità $w=vQ^n$ coinciderà comunque con quella iniziale.

Definiamo matrice di transizione regolare una matrice di transizione ad un tempo m>0 tale che

$q_{ij}^{(m)} > 0 \quad , \quad \forall i, j \in E$

La catena risultante dalla matrice regolare è detta catena regolare, in una catena regolare tutti gli stati comunicano tra di loro e quindi la catena è irriducibile e tutti gli stati sono ricorrenti

Un'algoritmo semplice per vedere la regolarità di una catena sta nel controllare tutte le probabilità di transizione per vari tempi di m, un'algoritmo che riesca a vedere la regolarità è il seguente:

```
regularity_chain(Markov_chain markovch){
        let temp = markovch.transition_matrix
        for(i in length(markoch.state)){
                let count=0;
                for(row in length(markovch.state)){
                        for(col in length(markovch.state)){
                                if(temp[row][col]>0)count+=1
                        }
                }
                if(count>=length(state)^2)return TRUE
                temp *= markovch.transition_matrix
        }
        Return FALSE
}
```

Una implementazione in R è la seguente

```
regularity_chain ← function(){
 tempmat <-  markovch@transition_matrix
 for(i in 1:length(markovch@states)){
   count <- 0;
   for(row in 1:length(markovch@states)){
     for(col in 1:length(markovch@states)){
       if(tempmat[row,col]>0){count <- count + 1}
     }
   }
   if(count>=length(markovch@states)^2)return(TRUE)
   tempmat = tempmat %*% markovch@transition_matrix
 }
 return(FALSE)
}
```

2.8 Stati stazionari

Vogliamo studiare una catena di Markov per tempi molto grandi. Quando l'istante di osservazione e' molto lontano dal punto iniziale, la probabilita' di trovare la catena in uno stato j, p_j, non dipende dallo stato iniziale

$$\lim_{n\to\infty} \mathbb{P}(X_n = j|X_0 = i) = \lim_{n\to\infty} q_{i,j}^{(n)} = p_j \quad , j = 0, 1..$$

Quando le probabilita' limite p_j esistono (e la loro somma fa uno), esse si chiamano distribuzione di equilibrio o distribuzioni dello stato stazionario di una catena di Markov.

Per il calcolo di distribuzioni di stato stazionario ci affidiamo principalmente al teorema di Markov per cui se la catena di markov è regolare, allora esiste ed è unica la probabilità invariante per cui

$\lim_{n\to\infty} q_{ij}^{(n)} = p_j$ dove p è la probabilità invariante

Conseguenze di questo teorema sono che la legge di probabilità associata per n che tende a +∞ è

$$\mathbb{P}(X_n = j) = (vQ^n)_j = \sum_{i\in E} v_i q_{ij}^{(n)} \to \left(\sum_{i\in E} v_i\right) p_j = p_j$$

e quindi qualunque sia la distribuzione iniziale v la X_n converge in legge alla distribuzione invariante p.

quindi per calcolare la distribuzione limite abbiamo due possibilità (tre in realtà):

1) Possiamo calcolare la probabilità di occupazione per $n \to +\infty$, cioè $p=vQ^n$ cioè approssimare per valori di n abbastanza grandi

2) Possiamo notare la similarità tra la legge di probabilità stazionaria v=vQ e la teoria di algebra lineare che riguarda gli eigenvalues(autovalori) e gli eigenvector ($Mv=\lambda v$, nel nostro caso trasponiamo la matrice di transizione e cerchiamo gli eigenvettori con eigenvalue=1) ,e prendere l'eigenvettore di eigenvalue=1 che rispetta sia l'equazione di bilancio $p_j = \sum_{i\in E} p_i q_{i,j}$, per ogni sua componente che quella di normalizzazione $\sum_{i\in E} p_i = 1$

La seconda possibilità è la più veloce poichè non dobbiamo calcolare un numero di moltiplicazioni matriciali troppo alto. Ovviamente abbiamo pure la possibilità di risolvere il sistema lineare formato dalle equazioni di bilancio per ogni componente e dall'equazione di normalizzazione, ma è quasi equivalente al calcolo degli eigenvector.

Gli algoritmi per i due metodi presentati sono :

```
stationary_state_matrixmult(Markov_chain markovch,n){
  if(!regularity_chain(markovch))exit
  matrix_multy(markovch.initial_state,(matrix_pow(markovch@transition_matrix, n))
}

stationary_state_eigenvect(Markov_chain markovch){
  if(!regularity_chain(markovch))exit
  Let eigenvect  = eigen(t( markovch.transition_matrix))
  Let eigenval = eigenvalues(t(markovch.transition_matrix))
  for(i in 1:ncol(eigenvect)){
    If( Re( eigenval[i]) == 1){
      normalizer <- 1/( rep(1,nrow(eigenvect)) %*% Re(eigenvect[,i]) )
      return(Re(eigenvect[,i])%*% normalizer)
    }
  }
  Return c(0,0,0)
}
```

il codice in R è:

```
stationary_state_matrixmult <- function(markovch){
  if(!regularity_chain(markovch))stop("la catena non è regolare")
  library(expm)
  markovch@initial_state%*%(markovch@transition_matrix %^% 50)
}


stationary_state_eigenvect <- function(markovch){
  if(!regularity_chain(markovch))stop("la catena non è regolare")
  eigenvect <- eigen(t( markovch@transition_matrix))[[2]]
  eigenval <- eigen(t(markovch@transition_matrix),only.values = TRUE)[[1]]
  for(i in 1:ncol(eigenvect)){
    if(Re( eigenval[i])>= 1-0.001 && Re( eigenval[i])<= 1+0.001){
      normalizer <- 1/( rep(1,nrow(eigenvect)) %*% Re(eigenvect[,i]) )
      return(Re(eigenvect[,i])%*% normalizer)
    }
  }
  c(0,0,0)
}
```

3. DiSCUSSiONE FiNALE

Come già stabilito nell'introduzione, in questa parte tenterò di iniziare una dissertazione verso un esempio reale e tenterò di costruire una infrastruttura di controllo per la creazione di un modello di Catena di Markov.

incomincio la discussione parlando di processi di espansione, continuo tentando di stabilire un modello di analisi sulla base delle catene di Markov, per poi derivare accennare per inferenza la catena vera e propria usando come principio algoritmi come learning HMM

PROCESSi Di ESPANSiONE

importante e di fondamentale comprensione è la parte della teoria che studia i processi e le dinamiche di espansione e propagazione(spreading) di fenomeni, modelli compartimentale come suddivisione in stati caratteristici, cioè ogni nodo appartiene ad un compartimento.

Per esempio nel modello di espansione SiR un nodo può appartenere a tre compartimenti differenti, suscettibile, infetto e recovered(guarito), i passaggi da un compartimento ad un altro avvengono tramite funzioni che tengono conto del tempo di contatto tra i vari compartimenti).

Le principali caratteristiche di un virus sono il tasso di infezione ed il tasso di mortalità, sono queste caratteristiche che permettono la costruzione del nostro modello.

Per esempio potremmo costruire il modello pensando agli stati di una persona(sano, infetto, guarito-immune, morto), e alle probabilità di passare da uno stato all'altro come i tassi stabiliti(mortalità, infezione, guarigione).

Un altro possibile modello che potremmo costruire in contemporanea della catena di infezione è quello dello sviluppo di un virus, che segue diversi stati aggiungendo nuove caratteristiche, infatti potremmo riuscire a studiare l'evoluzione temporale del virus vedendo le caratteristiche che ha sviluppato fino a quel momento, per poi costruire un modello di predizione dell'evoluzione virale(infatti per questa modellizzazione ci si può affidare ad una HMM e trattare le caratteristiche del virus come simboli generati da stati)