D Appendix 1

D.1 R- Code

```
# Code for the conversion of matrices
# Input parameters:
#       A... matrix that needs to be converted (original matrix)
#       p... defined model cycle length
#
# Methods
#       M1... Sonnenberg method
#       M2... Schur-Pade method
#               M21... regularized Schur-Pade method with ER
#               M22... regularized Schur-Pade method with ERE
#       M3...   Eigenvalue method
#               M31... regularized Eigenvalue method with ER
#               M32... regularized Eigenvalue method with ERE
#               M33... regularized Eigenvalue method with Q
#
# Output:
#       Solution... matrix with cycle length p

convertTP <- function(A,p){

# Load packages
install.packages("lattice")
install.packages("Matrix")
install.packages("expm")
install.packages("xlsx")
library(expm)
library(lattice)
library(Matrix)

# Define different methods
# M1 - Sonnenberg method

Sonnenberg <- function(A,p){
n = dim(A)[1]
```

```
ii = c(1:n)
for (i in ii){
        for (j in ii){
                if(j!=i){A[i,j] = 1-(1-A[i,j])^(p)}
                }

A[i,i] = 1- sum(A[i,(1:n)])+A[i,i]
}

return(A)
}

# M2 - Schur-Pade method

SchurPade <- function(A,p){

# Basics

maxsqrt = 36
n = dim(A)[1]
nsq = 0
m = 0

# Compute a (complex) Schur decomposition A = QTQ*

AS = Schur(A)
Q = AS$Q
T = AS$T

# Eigenvalues of A
diagT = diag(T)

# Check eigenvalues of A and return warnings for matrices with
   nonpositive
# eigenvalues

if (nnzero(diagT) != n){
```

```
        warning("Matrix power may not exist for singular matrix"
            )
}

if (any(Im(diagT)==0 & Re(diagT)<=0)){
        warning("Principal matrix power is not defined for A
            with nonpositive eigenvalues.
        A non-principal matrix power is returned")
}

# Handle special case of diagonal T
if (identical(T,diag(diagT,n))){
X = Q%*%diag(diagT^p,n)%*%t(Q)
return(X)
}

##################FUNCTIONS FOR FINAL EVALUATION

####POWERM2BY2
powerm2by2 = function(A,p){
# POWERM2BY2    Power of 2-by-2 upper triangular matrix.
#   POWERM2BY2(A,p) is the pth power of the 2-by-2 upper
#    triangular matrix A, where p is an arbitrary real number.

a1 = as.complex(A[1,1])
a2 = as.complex(A[2,2])

a1p = a1^p
a2p = a2^p

loga1 = log(a1)
loga2 = log(a2)

X = diag(c(a1p, a2p))

if (a1 == a2){
        X[1,2] = p*A[1,2]*a1^(p-1)
        return(X)
```

```
}
else if (abs(a1) < 0.5*abs(a2) || abs(a2) < 0.5*abs(a1)){
        X[1,2] = A[1,2]*(a2p-a1p)/(a2-a1)
        return(X)
}
else
        ####UNWINDING
        unwinding = function(z,k){
        #   UNWINDING(Z,K) is the K'th derivative of the
        #   unwinding number of the complex number Z.
        #   Default: k = 0.
        #MISSING NARGIN
        if (k==0){
                u = ceiling((Im(z)-pi)/(2*pi))
                return(u)
        }
        else u = 0
        return(u)
        }

        w = atanh((a2-a1)/(a2+a1))+1i*pi*unwinding(loga2-loga1,k
            )
        dd = 2*exp(p*(loga1+loga2)/2)*sinh(p*w)/(a2-a1)
        X[1,2] = A[1,2]*dd
return(X)
}

####SQRTM_TRI

sqrtm_tri = function(T){
# Upper triangular square root of upper triangular matrix

n = dim(T)[1]

# change format to complex

for (i in 1:n){
        for (j in 1:n){
```

```
                T[i,j]=as.complex(T[i,j])
        }
}

R = diag(0,n)

for(j in 1:n){
        R[j,j]=sqrt(T[j,j])
        if(j>1){
                for(i in seq(j-1,1,-1)){
                        R[i,j]=(T[i,j]-R[i,(i+1):(j-1)]%*%R[(i
                            +1):(j-1),j])/(R[i,i]+R[j,j])
                }
        }
}
return(R)
}

####COEFF

coeff = function(p,i){
# Compute coefficients for Pade approximation
if (i == 1){
        c = -p
        return(c)
}
else
        g = i/2
        if (g == round(g)){
                c = (-g+p)/(2*(2*g-1))
                return(c)
        }
        else
                g = floor(g)
                c = (-g-p)/(2*(2*g+1))
                return(c)
}
```

```
####POWERM_CF

powerm_cf = function(Y,p,m){
# POWERM_CF   Evaluate Pade approximant bottom up.
# POWERM_CF(Y,p,m) computes the [m/m] Pade approximant of the
# matrix power (I-Y)^p by evaluating a continued fraction
# representation in bottom-up fashion.

k <<- 2*m
n = dim(Y)[1]

S = coeff(p,k)*Y
#f = seq(k-1,1,-1)
#d = (k-1):1
#for (i in 1:(k-1)){
#d = d[i]
#}
d = c((k-1):1)
for(i in d){
        z = coeff(p,i)
        S = z*solve((diag(1,n)+S))%*%Y
}
S = diag(1,n)+S

return(S)
}

#### Main function
####POWERM_TRIANG

powerm_triang<-function(T,p,maxsqrt){
# POWERM_TRIANG   Power of triangular matrix by Pade-based
   algorithm.
# X = POWERM_TRIANG(T,P,MAXSQRT) computes the P'th power X of
# the upper triangular matrix T, for an arbitrary real number P
   in the
# interval (-1,1),and T with no nonpositive real eigenvalues, by
    a
```

```
# Pade-based algorithm. At most MAXSQRT matrix square roots are
    computed.
# [X NSQ,M] = POWERM_TRIANG(T,P) returns the number NSQ of
# square roots computed and the degree M of the Pade approximant
     used.

n=dim(T)[1]
nsq=0
if(n==1){
        X=T^p #scalar power
        m=0
        return(X)
}
if(n==2){
        X=powerm2by2(T,p)#compute power directly
        m=0
        return(X)
}
T_old=T
nsq=0
q=0

# Max norm(X) for degree m Pade approximant to (I-X)^p

xvals =         c(1.512666672122460e-005,                  # m = 1
                  2.236550782529778e-003,                  # m = 2
                  1.882832775783885e-002,                  # m = 3
                  6.036100693089764e-002,                  # m = 4
                  1.239372725584911e-001,                  # m = 5
                  1.998030690604271e-001,                  # m = 6
                  2.787629930862099e-001,                  # m = 7
                  3.547373395551596e-001,                  # m = 8
                  4.245558801949280e-001,                  # m = 9
                  4.870185637611313e-001,                  # m = 10
                  5.420549053918690e-001,                  # m = 11
                  5.901583155235642e-001,                  # m = 12
                  6.320530128774397e-001,                  # m = 13
                  6.685149002867240e-001,                  # m = 14
```

```
                        7.002836650662422e-001,          # m = 15
                        7.280253837034645e-001,          # m = 16
                        9.152924199170567e-001,          # m = 32
                    9.764341682154458e-001)          # m = 64

W = diag(0,n)
while(1){
        M = Mod(as.matrix(T-diag(1,n)))
        normdiff = norm(M,"1");
        if(normdiff <= xvals[7]){

                        q = q+1
                        j1 = which(xvals[3:7]>=normdiff)
                        j1 = j1[1]+2
                        j2 = which(xvals[3:7]>= normdiff/2)
                        j2 = j2[1]+2
                        if(j1-j2<=1 || q == 2){
                                m = j1;
                                break
                        }
                }

                if (nsq == maxsqrt){
                                m=16;
                                break
                }
                T = sqrtm_tri(T)                    #Schur method
                nsq = nsq + 1;
}
X = powerm_cf(diag(1,n)-T,p,m)
#Squaring phase, with directly computed diagonal and
   superdiagonal

for (s in 0:nsq){
        if(s != 0){
                X = X%*%X
        }
        for(i in 1:(n-1)){
```

```
                    Tii = T_old[i:(i+1),i:(i+1)]
                    Si = powerm2by2(Tii, p/(2^(nsq-s)))
                    X[i:(i+1),i:(i+1)] = Si;
          }
}
return(X)
}
####FINAL EVALUATION

X = powerm_triang(T,p,maxsqrt)
X = Q%*%X%*%t(Q)

return(X)
}

# M3 - Eigenvalue method

Eigenvalue<-function(A,p){

E = expm(logm(A, "Eigen")*p)

return(E)
}

# Define regularization algorithms

# Relative Entropy

RegRE <- function(P){
d=dim(P)[1]
P=Re(P)
z = which(P<0)
P[z] = 0
for(i in 1:d){
          n = sum(P[i,1:d])
          P[i,1:d]=P[i,1:d]/n
}
return(P)
```

```
}

# Extreme relative entropy algorithm
# Restriction: only possible if max+min <= 1

RegERE <- function(P){

d=dim(P)[1]
for(i in 1:d){
        if(sum(P[i,1:d])-1<10^(-16) && min(P[i,1:d])>=0){
        break
        # P[i,1:d]=P[i,1:d]
        }
        else
        z = which(P[i,1:d]<0)
        P[i,z] = 0

        maxP = max(P[i,1:d])
        w = which(P[i,1:d]!=0)
        minP = min(P[i,w])
                if(maxP+minP >1 | maxP==minP){
                warning("max + min > 1 or min = max; this
                    algorithm cannot be applied")
                break
                }
        delta = 1-maxP-minP
        n = sum(P[i,1:d])-maxP-minP
        for(j in 1:d){
                if(P[i,j]!=maxP && P[i,j]!= minP){
                        P[i,j]=(P[i,j]/n)*delta
                }
        }
}
return(P)
}

# Algorithm for the regularization of Q matrices
```

```
RegQ <- function(A,p){
R = logm(A, "Eigen")
d = dim(R)[1]
g=matrix(0,d,1)
b=matrix(0,d,1)

for (i in 1:d){
        for (j in 1:d){
                if(j!=i){
                g[i]=g[i]+max(R[i,j],0)
                b[i]=b[i]+max(-R[i,j],0)
                }
        }
}
for(i in 1:d){
        g[i]=abs(R[i,i])+g[i]
}

for (i in 1:d){
        for (j in 1:d){
                if(j!=i && R[i,j]<0){
                        R[i,j]=0
                }
                if(g[i]>0){
                        R[i,j]=R[i,j]-b[i]*abs(R[i,j])/g[i]
                }
                if(g[i]==0){
                        R[i,j]=R[i,j]
                }
        }

}

A1 = expm(R*p)
return(A1)
}
```

```
# Define parameters for the computation
error = matrix(0,8,2)
error[,1]=c('M1','M2','M21','M22','M3','M31','M32','M33')

# Compute Matrices using methods M1, M2 and M3

M1 = Sonnenberg(A,p)
M2 = SchurPade(A,p)
M3 = Eigenvalue(A,p)

# Assure non-complexity and stochasticity of matrices
# If matrices are stochastic compute corresponding error using
   Frobenius norm

error[1,1]='M1'
error[1,2]= norm(A-M1%^%(1/p),"f")/norm(A,"f")

M2 = Re(M2)
M3 = Re(M3)

# Check if matrices are stochastic - if not use regularization
   methods
# Additionally compute relative error using frobenius norm

d=dim(A)[1]

if(sum(M2)-d<10^(-16) && min(M2)>=0){
        error[2,2]= norm(A-M2%^%(1/p),"f")/norm(A,"f")
        error[3,2]= 'NA'
        error[4,2]='NA'
        }
        else{
                M21<- RegRE(M2)
                M22<- RegERE(M2)
                if(sum(M22)-d<10^(-16) && min(M22)>=0){
                        error[2,2]='NA'
                        error[3,2] = norm(A-M21%^%(1/p),"f")/
                            norm(A,"f")
```

```
                    error[4,2] = norm(A-M22%^%(1/p),"f")/
                        norm(A,"f")
                }
                else{ error[2,2] = 'NA'
                    error[3,2] = norm(A-M21%^%(1/p),"f")/
                        norm(A,"f")
                    error[4,2] = 'NA'
                }

        }



if(sum(M3)-d<10^(-16) && min(M3)>=0){
        error[5,2]= norm(A-M3%^%(1/p),"f")/norm(A,"f")
        error[6,2]='NA'
        error[7,2]='NA'
        error[8,2]='NA'
        }
        else{ M31 <- RegRE(M3)
                M33 <- RegQ(A,p)
                M32 <- RegERE(M3)
                if(sum(M32)-d<10^(-16) && min(M32)>=0){
                    error[5,2] = 'NA'
                    error[6,2] = norm(A-M31%^%(1/p),"f")/
                        norm(A,"f")
                    error[7,2] = norm(A-M32%^%(1/p),"f")/
                        norm(A,"f")
                    error[8,2] = norm(A-M33%^%(1/p),"f")/
                        norm(A,"f")
                }
                else {error[5,2] = 'NA'
                    error[6,2] = norm(A-M31%^%(1/p),"f")/
                        norm(A,"f")
                    error[7,2] = 'NA'
                    error[8,2] = norm(A-M33%^%(1/p),"f")/
                        norm(A,"f")
                }
```

```
        }

# Determine best fit according to smallest error

print(error)
b = which.min(error[1:8,2])
BestFit = error[b,]

# Print the method and error of the final result
print(error[b,])

# Find the corresponding matrix
Solution = get(error[b,1])

# Output
return(Solution)
# return(error)
}
```

E Appendix 2

E.1 Additional results

The transition matrix with the best performance in the case example

- treatment pattern 1 (T1):

$$TM_{monthly} = \begin{pmatrix} 0.9952969 & 0.0005349666 & 0.001475188 & 0.002692947 \\ 0.0000000 & 0.9924437433 & 0.005640548 & 0.001915709 \\ 0.0000000 & 0.0000000000 & 0.959251288 & 0.040748712 \\ 0.0000000 & 0.0000000000 & 0.000000000 & 1.000000000 \end{pmatrix}$$

- treatment pattern 2 (T2):

$$TM_{monthly} = \begin{pmatrix} 0.9961704 & 0.0002661867 & 0.0007337746 & 0.002829682 \\ 0.0000000 & 0.9924437433 & 0.0056405480 & 0.001915709 \\ 0.0000000 & 0.0000000000 & 0.9592512878 & 0.040748712 \\ 0.0000000 & 0.0000000000 & 0.000000000 & 1.000000000 \end{pmatrix}$$

- treatment pattern 3 (T3):

$$TM_{monthly} = \begin{pmatrix} 0.9967719 & 0.0002648875 & 7.564649e-07 & 0.002962420 \\ 0.0000000 & 0.9924437433 & 5.640548e-03 & 0.001915709 \\ 0.0000000 & 0.0000000000 & 9.592513e-01 & 0.040748712 \\ 0.0000000 & 0.0000000000 & 0.000000000 & 1.000000000 \end{pmatrix}$$

E.2 Definitions

Definition E.1. *Nonsingular matrix [[27],34] A square matrix A is said to be invertible or nonsingular if there exists a matrix B such that*

$$A \cdot B = B \cdot A = I$$

where I is the identity matrix. Such a matrix B is unique. That is, if $A \cdot B_1 = B_1 \cdot A = I$ and $A \cdot B_2 = B_2 \cdot A = I$, then

$$B_1 = B_1 \cdot I = B_1 \cdot (A \cdot B_2) = (B_1 \cdot A) \cdot B_2 = I \cdot B_2 = B_2$$

We call such a matrix B the inverse of A and denote it by A^{-1}.

Definition E.2. *Unitary matrix [[27],38] A complex matrix A is unitary if $A^H \cdot A^{-1} = A^{-1} \cdot A^H = I$ that is, if $A^H = A^{-1}$. A^H is the conjugate transpose of A.*

Definition E.3. *Stochastic matrix [[28],34] A stochastic matrix over a field F is a square matrix with entries form F with the property that the entries in each of its columns add up to 1.*

If F is an ordered field, than we may define a positively stochastic matrix to be a stochastic matrix with non-negative entries—this is what is usually called a stochastic matrix, in the case where F is the field of real numbers.

Definition E.4. *Accessibility [[29],204] State j is said to be accessible from state i if $P_{ij}^n > 0$ for some $n \geqslant 0$. Note that this implies that state j is accessible from state i if and only if, starting in i, it is possible that the process will ever enter state j.*

Definition E.5. *Eigenvalues and Eigenvectors [[27],296] Let A be any square matrix. A scalar λ is called an eigenvalue of A if there exists a non-zero (column) vector υ such that*

$$A\upsilon = \lambda\upsilon$$

Any vector satisfying this relation is called an eigenvector of A belonging to the eigenvalue λ.

An example for the computation of eigenvalues and the corresponding eigenvectors can be found in ([8]).

E.3 Selected theorems on generators or intensity matrix Q

(i) Existence

Theorem E.1 ([14])**.** *Let P be a transition matrix and suppose that*

(i) $\det(P) \leq 0$ *or*

(ii) $\det(P) > \prod_i p_{ii}$ *or*

(iii) there are states i and j such that j is accessible from i, but $p_{ij} = 0$.

Then there does not exist an exact generator for P.

(ii) Uniqueness

Theorem E.2 ([17])**.** *Let P be a transition matrix*

(i) [14] If $\det(P) > 1/2$*, then* P *has at most one generator.*

(ii) [14] If $\det(P) > 1/2$ *and* $\|(P - I) < 1/2\|$ *(using any matrix norm), then the only possible generator for* P *is* $\log(P)$

(iii) [30], [31] If P *has distinct eigenvalues and* $\det(P) > \exp(-\pi)$*, then the only possible generator for* P *is* $\log(P)$

(iv) [15] If P *has real, positive, distinct eigenvalues, then the only real matrix* Q *such that* $\exp(Q) = P$ *is* $\log(P)$

However, if more than one solution exists for the desired transition matrix, it should be clarified if more information about the underlying process is available to help select a relevant transition matrix [15].

E.4 Advanced regularization methods for P_n

If the algorithm to calculate the $n-th$ root of the transition matrix does not provide a valid stochastic transition matrix for cycle length n P_n, Charitos, et al. [18], propose two different regularization methods: relative entropy (RE) and extreme relative entropy (ERE).

For simplicity replace P_n^* (matrix resulting from removing the imaginary parts of all entries of P_n) with P. Denote by p_i^+ the vector that results by replacing each negative entry of p_i with zero and as m^+ the number of positive entries left.

– **Distance measure: relative entropy (RE)** ([18] + Cover 1991)

$$p_i^r := \arg\min_{p_i'} \sum_{j:p_{ij}^+>0} p'_{ij} \ln \frac{p'_{ij}}{p_{ij}^+} \tag{20}$$

Adding the Lagrangian multiplier λ, using the constraint $\sum_{j=1}^{m^+} p'_{ij} = 1$, setting the derivative equal zero and summing over j to get λ, we find for those j where $p_{ij}^+ > 0$ that $p_{ij}^r = p_{ij}^+ / \sum_{j=1}^{m^+} p_{ij}^+$. The optimal solution is the normalization of p_i^+.

– **Distance measure: extreme relative entropy (ERE)** [18] This method confines the RE to the values that are between the extreme values of p_i^+, $p_{il}^+ = \max_i p_{ij}^+$ and $p_{ih}^+ = \min_i p_{ij}^+$. Therefore p_i^r must satisfy $p_{il}^r = p_{il}^+$, $p_{ih}^r = p_{ih}^+$ and

$$p_i^r := \arg\min_{p_i'} \sum_{j:p_{ij}^+>0;j\neq l,h} p_{ij}' \ln \frac{p_{ij}'}{p_{ij}^+} \tag{21}$$

Adding the Lagrangian multiplier λ, using the constraint $\sum_{j=1,\neq h,l}^{m^+} p_{ij}' = \delta$, setting the derivative equal zero and summing over j to get λ, we find for those j where $p_{ij}^+ > 0$ that $p_{ij}^r = (p_{ij}^+ / \sum_{j=1\neq h,l}^{m^+} p_{ij}^+)\delta$. This method is only applicable if $\min_i p_{ij}^+ + \max_i p_{ij}^+ \leq 1$ holds. The rationale for using the ERE is to resemble the potential transition probabilities exactly for the two extremes.

Example
The annual transition probability matrix for treatment strategy 3 can be transformed into a monthly transition probability matrix:

- **monthly transition probability matrix using Eigenvalues method with negative entry in the first row:**

$$T3_{monthly,E} = \begin{pmatrix} 0.9967768 & 0.0002652894 & -8.807312e-06 & 0.002966715 \\ 0.0000000 & 0.9924437433 & 0.005640548 & 0.001915709 \\ 0.0000000 & 0.0000000000 & 0.959251288 & 0.040748712 \\ 0.0000000 & 0.0000000000 & 0.000000000 & 1.000000000 \end{pmatrix}$$

- **monthly transition probability matrix using Eigenvalues method and RE without a negative entry:**

$$TM_{monthly,E-RE} = \begin{pmatrix} 0.996768 & 0.0002652871 & 0 & 0.002966688 \\ 0.0000000 & 0.9924437433 & 0.005640548 & 0.001915709 \\ 0.0000000 & 0.0000000000 & 0.959251288 & 0.040748712 \\ 0.0000000 & 0.0000000000 & 0.000000000 & 1.000000000 \end{pmatrix}$$

- **monthly transition probability matrix using Eigenvalues method and ERE without a negative entry:**

$$TM_{monthly,E-ERE} = \begin{pmatrix} 0.9967768 & 0.0002652894 & 0 & 0.002957907 \\ 0.0000000 & 0.9924437433 & 0.005640548 & 0.001915709 \\ 0.0000000 & 0.0000000000 & 0.959251288 & 0.040748712 \\ 0.0000000 & 0.0000000000 & 0.000000000 & 1.000000000 \end{pmatrix}$$

F Appendix 3

F.1 Results random matrix

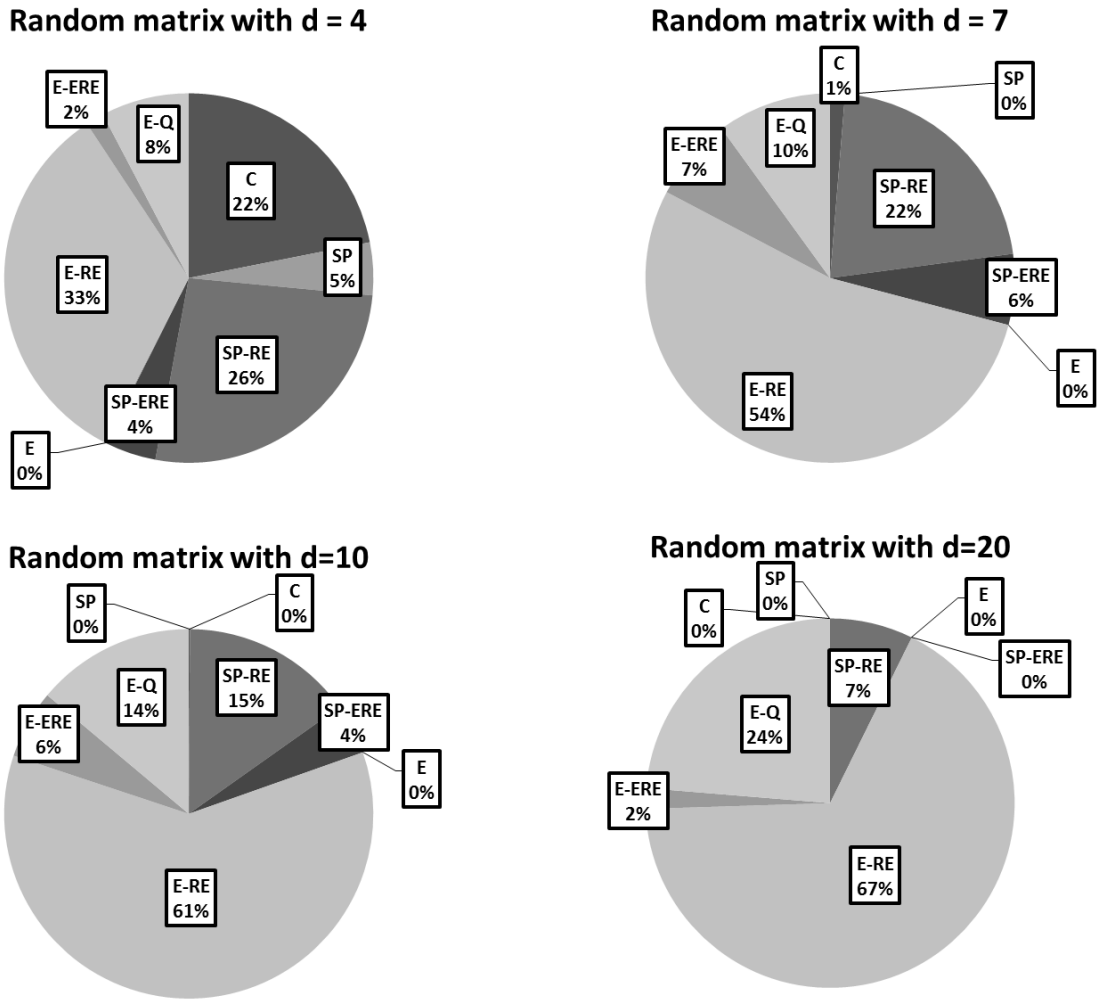

Figure 3: Graphical representation of the best performing method for different dimension of matrices (E eigenvalue, SP Schur-Padé, C common, E-Q eigenvalue with Q-regularization, E-ERE eigenvalue with extreme relative entropy regularization, E-RE eigenvalue with relative entropy regularization, SP-ERE Schur-Padé with extreme relative entropy, SP-RE Schur-Padé with relative entropy regularization), d dimension of the transition matrix meaning number of health states

G Appendix 4

G.1 Case example relative differences outcomes

Treatment Strategy	Outcome	Method	Relative Differences
T1	LYs	C	-0.452
		SP	0.000
		E	0.000
T1	QALYs	C	-0.073
		SP	-0.012
		E	-0.012
T1	Costs	C	-10.646
		SP	0.333
		E	0.333
T2	LYs	C	-0.236
		SP	-0.005
		E	-0.005
T2	QALYs	C	-0.044
		SP	-0.011
		E	-0.011
T2	Costs	C	-5.634
		SP	0.162
		E	0.162

Table 6: Relative differences between point estimates of different treatment patterns calculated with the original matrix and with matrices corresponding to the transformation methods (E eigenvalue, SP Schur-Padé, C common, QALYs quality-adjusted life-years, LYs - life years)

Treatment Strategy	Outcome	Method	Relative Differences
T3	LYs	C	-0.041
		SP-RE	-0.042
		SP-ERE	0.008
		E-RE	-0.042
		E-ERE	0.008
		E-Q	-0.020
T3	QALYs	C	-0.024
		SP-RE	-0.052
		SP-ERE	-0.003
		E-RE	-0.052
		E-ERE	-0.003
		E-Q	-0.031
T3	Costs	C	-0.517
		SP-RE	0.276
		SP-ERE	0.324
		E-RE	0.276
		E-ERE	0.324
		E-Q	0.313

Table 7: Relative differences between point estimates of different treatment patterns calculated with the original matrix and with matrices corresponding to the transformation methods (E eigenvalue, SP Schur-Padé, C common, E-Q eigenvalue with Q-regularization, E-ERE eigenvalue with extreme relative entropy regularization, E-RE eigenvalue with relative entropy regularization, SP-ERE Schur-Padé with extreme relative entropy, SP-RE Schur-Padé with relative entropy regularization, QALYs quality-adjusted life-years, LYs life years), T treatment strategy