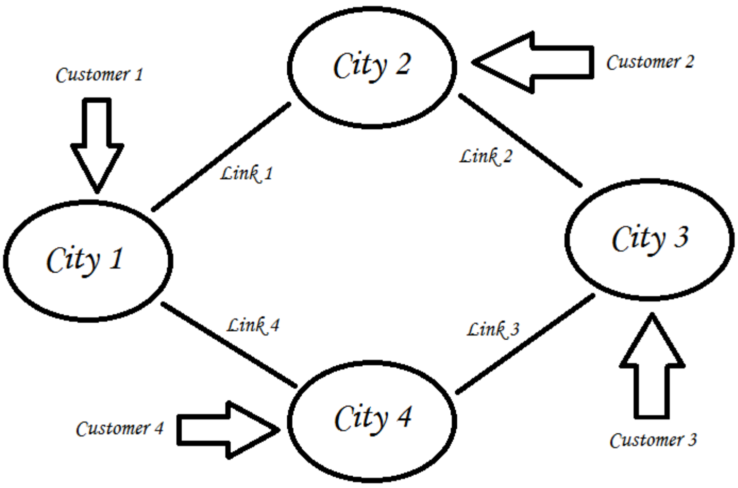
Eindhoven University of Technology

2MMS40 Stochastic Networks

Four City Problem



Supervisor:
Prof. dr. J.S.H. van Leeuwaarden

Authors:
Krishna Kalluri
Student number
s.k.kalluri@student.tue.nl

Tim Meeles
0809731
t.meeles@student.tue.nl

15th May 2018

Contents

1 Introduction

As an extension of the three city problem, in this project a non-typical four city problem is treated. A sketch of this problem is given below in the figure.

As one can see, the four cities are connected, but they are not all connected to each other. This means that some customers, or alternatively jobs, will travel from one city to another through other cities. Or, as one could argue, jobs travel through different network centres, where also other jobs can be added.

The more classical approach of customers travelling from one city to another, with different inflows and different destinations, called different class customers, can be extended to many applications in the digital world. Therefore, the results that are given can be interpreted in many ways and for many applications.

2 Problem Description

The four city problem, as treated in this report, consists of four cities, called City 1, 2, 3 and 4. All cities have their own class of customer, named Class 1, 2, 3 and 4 customer(s). Note that every class has a specific routing through the network. For example, a customer arriving at city 1, so a Class 1 customer, travels from City 1, through City 2 to City 3. The following matrix is given to show which classes of customers take which routes, so ultimately which links it takes during its travel:

$$b_{kl} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

The rows of this matrix show the different classes of customers, the columns denote the 4 links in the network.

Furthermore, every class of customers i has its own arrival rate. This is called λ_i. A Class i customer has a service time of 1 time unit, meaning that once the customers enters the system, 1time unit later it has left. This can be put into a formula as follows: $\mu_j = 1$ for all j.

Also, it is assumed that every link has a capacity, namely c_l for link l. Once this capacity is reached by the customers of any type, the following customer that needs to use this link in his route will be blocked. The goal of the assignment is to determine the blocking probabilities of the links and using that the blocking probabilities of the customers, using both an exact solution as the Erlang Fixed Point-method. The latter one is used to give an approximation of the blocking probabilities.

In this project the blocking probabilities for both the links as the classes of customers are used as performance measures.

3 Exact product-form solution

While Erlang-Fixed-Point method gives approximate result, the approximation is not always good, particularly in case of large dependency between links. In this section Exact product-form solution is calculated for the given network. However, even though this method is accurate it requires high computation time and isn't often suitable for large networks.

3.1 Method

The given network consists of four classes. Hence the state space of the network can be represented as $S := \{n\,\epsilon N^4\}$. However the network also has limits regarding maximum capacity on each link. Henceforth, the reduced state space satisfying the capacity constraints can be represented as

$$S_L := \{n\,\epsilon N^K : \sum_{k=1}^{K} b_{kl} n_k \leq C_l\, for\, all\, l\epsilon L\}$$

where $K = 4$ and $L = \{1, 2, 3, 4\}$ for the given network. For the above set of states, corresponding probabilities can be computed as a product form of each class present in the state as follows.

$$\pi(n) = G^{-1} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}$$

where G is the normalization constant of the system, computed as sum of product-form's of each state.

$$G = \sum_{n\epsilon S_L} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}$$

Each link starts to block once it reaches it's capacity limit. Therefore blocking probability of a link $\tilde{B}_l$ can be computed as sum of probabilities of states where the link reaches it's capacity limit.

$$\tilde{B}_l = \sum_{n\epsilon S_L, \sum_{k=1}^{K} b_{kl} n_k = C_l} \pi(n)$$

Thus blocking probability of each class from the blocking probabilities of links for the given network can be computed as follows.

$$B(1) = 1 - (1 - \tilde{B}_1) \cdot (1 - \tilde{B}_2) \tag{2}$$

$$B(2) = 1 - (1 - \tilde{B}_2) \cdot (1 - \tilde{B}_3) \tag{3}$$

$$B(3) = 1 - (1 - \tilde{B}_3) \tag{4}$$

$$B(4) = 1 - (1 - \tilde{B}_1) \cdot (1 - \tilde{B}_4) \tag{5}$$

The implementation in Matlab can be found in Appendix A.

4 Erlang Fixed Point Method

As the exact solution can become very expensive to calculate due to the enormous state space when considering either many states/cities or high capacities on the links, and therefore higher arrival rates, the Erlang Fixed Point Method is used to determine approximations for the blocking probabilities of links. This is done in the following way:

Consider the transition matrix as given in the problem description, see matrix 1. Then the approximations of the blocking probabilities of links ($\tilde{B}$) are given by taking the Erlang-B-formula (notation $B(.,.)$) with as first parameter the capacity of the link for which one is determining the blocking probability and as a second parameter the sum of the parameters of the class of customers that uses this specific link, and the parameters multiplied with the probability that the other link(s) that that specific class of customer also needs to use are free, if applicable. (This is not applicable when a class of customers only uses one link. This results into the following set of equations for our problem:

$$\tilde{B_1} = B(c_1, \lambda_1(1 - \tilde{B_2}) + \lambda_4(1 - \tilde{B_4})) \tag{6}$$

$$\tilde{B_2} = B(c_2, \lambda_1(1 - \tilde{B_1}) + \lambda_2(1 - \tilde{B_3})) \tag{7}$$

$$\tilde{B_3} = B(c_3, \lambda_3 + \lambda_2(1 - \tilde{B_2})) \tag{8}$$

$$\tilde{B_4} = B(c_4, \lambda_4(1 - \tilde{B_1})) \tag{9}$$

By iterating the recursive relations, one will find the final approximation of the blocking probabilities of links, $\tilde{B}$. After that, one can find the blocking probabilities of a specific class of customers by:

$$B(1) = 1 - (1 - \tilde{B_1}) \cdot (1 - \tilde{B_2}) \tag{10}$$

$$B(2) = 1 - (1 - \tilde{B_2}) \cdot (1 - \tilde{B_3}) \tag{11}$$

$$B(3) = 1 - (1 - \tilde{B_3}) \tag{12}$$

$$B(4) = 1 - (1 - \tilde{B_1}) \cdot (1 - \tilde{B_4}) \tag{13}$$

The implementation in Matlab can be found in Appendix A.

5 Results

5.1 Situation 1, equal arrival rates, large capacities

Simulations have been performed using both Erlang-Fixed point method and Exact-Product-form Solution method for the following network configuration.

$$\lambda_k = \begin{pmatrix} 0.8 & 0.8 & 0.8 & 0.8 \end{pmatrix}$$

$$\mu_k = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C_l = \begin{pmatrix} 3 & 2 & 4 & 5 \end{pmatrix}$$

$$b_{kl} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Following results have been observed.

5.1.1 Erlang-Fixed-Point Method

In this method the recursive iterations are performed until accuracy of 0.0001 is reached.

Blocking Probabilities of links

$$\tilde{B}_l = \begin{pmatrix} 0.1123 & 0.3065 & 0.0367 & 0.0007 \end{pmatrix}$$

Blocking Probabilities of classes

$$B_k = \begin{pmatrix} 0.3843 & 0.3319 & 0.0367 & 0.1129 \end{pmatrix}$$

5.1.2 Exact-Product-Form Method

Blocking Probabilities of links

$$\tilde{B}_l = \begin{pmatrix} 0.1084 & 0.3147 & 0.0310 & 0.0000 \end{pmatrix}$$

Blocking Probabilities of classes

$$B_k = \begin{pmatrix} 0.3890 & 0.3359 & 0.0310 & 0.1084 \end{pmatrix}$$

5.1.3 Conclusion

AS one can see, the blocking probabilities calculated using the EFP method are quite close to the actual exact solution of this problem. It can be concluded that the solutions of the blocking probabilities on links differ at maximum 0.005, though the blocking probabilities on the customers differ maximum 0.01. For an approximation that assumes independence between the different links (which is not the case in our small network), it approaches the real solution very well. Note that in this case, the arrival rates are the same for all customers and the capacities are relatively large. This raises the question what happens with different arrival rates and small capacities.

5.2 Situation 2, different arrival rates, small capacities

In the previous section, equal arrival rates for the different classes were treated. In this section, different arrival rates are discussed. The following network configuration is considered:

$$\lambda_k = \begin{pmatrix} 0.1 & 0.8 & 0.5 & 0.6 \end{pmatrix}$$

$$\mu_k = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C_l = \begin{pmatrix} 1 & 2 & 1 & 2 \end{pmatrix}$$

$$b_{kl} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Following results have been observed.

5.2.1 Erlang-Fixed-Point Method

In this method the recursive iterations are performed until accuracy of 0.0001 is reached.

Blocking Probabilities of links

$$\tilde{B}_l = \begin{pmatrix} 0.4001 & 0.0573 & 0.5564 & 0.0455 \end{pmatrix}$$

Blocking Probabilities of classes

$$B_k = \begin{pmatrix} 0.4345 & 0.5818 & 0.5564 & 0.4274 \end{pmatrix}$$

5.2.2 Exact-Product-Form Method

Blocking Probabilities of links

$$\tilde{B}_l = \begin{pmatrix} 0.3881 & 0.2310 & 0.4092 & 0.1780 \end{pmatrix}$$

Blocking Probabilities of classes

$$B_k = \begin{pmatrix} 0.5294 & 0.5457 & 0.4092 & 0.4971 \end{pmatrix}$$

5.2.3 Conclusion

As one can see, with different arrival rates for the different classes, and capacities that stress the network more than in situation 1, there is a larger difference between the approximation of the EFP method en the exact product form solution. It is found that the blocking probabilities of links differ more than the blocking probabilities of classes, in contrast to the first situation. In this case, the blocking probabilities on links differ maximally 0.2, though the blocking probabilities on classes differ maximum 0.15; a small difference. As one can see, the EFP method performs less accurately on stressed networks, probably because the independence assumption is violated on small stressed networks. However, this does still raise the question whether this depends on the relatively small capacity or on the different arrival rates.

5.3 Situation 3, different arrival rates, large capacities

In the previous section, equal arrival rates for the different classes were treated. In this section, different arrival rates are discussed. The following network configuration is considered:

$$\lambda_k = \begin{pmatrix} 0.1 & 0.8 & 0.5 & 0.6 \end{pmatrix}$$

$$\mu_k = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C_l = \begin{pmatrix} 5 & 5 & 5 & 5 \end{pmatrix}$$

$$b_{kl} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Following results have been observed.

5.3.1 Erlang-Fixed-Point Method

In this method the recursive iterations are performed until accuracy of 0.0001 is reached.

Blocking Probabilities of links

$$\tilde{B}_l = \begin{pmatrix} 0.0007 & 0.0019 & 0.0084 & 0.0004 \end{pmatrix}$$

Blocking Probabilities of classes

$$B_k = \begin{pmatrix} 0.0026 & 0.0103 & 0.0084 & 0.0010 \end{pmatrix}$$

5.3.2 Exact-Product-Form Method

Blocking Probabilities of links

$$\tilde{B}_l = \begin{pmatrix} 0.0007 & 0.0015 & 0.0084 & 0.0003 \end{pmatrix}$$

Blocking Probabilities of classes

$$B_k = \begin{pmatrix} 0.0022 & 0.0099 & 0.0084 & 0.0010 \end{pmatrix}$$

5.3.3 Conclusion

It is straightforward to see that the blocking probabilities are small, as the capacities are very high, compared to the arrival rates. However, one can also see that the probabilities as returned by the EFP method are very similar as the exact solution. This concludes that the performance of the EFP method depends on the stress on the network, i.e. whether the capacity is high or low in comparison with arrival rate.
It has been investigated whether similar arrival rates with small capacities gave large differences in the results of the EFP method and the exact product form solution. This is indeed the case.

6 Discussion

As one can see, there is a clear difference in the performance of the Erlang-Fixed-Point method in the two situations as treated in Section 5. Further research point out that this does not depend on the difference in arrival rate of the different classes in the network, but on the capacity of the links.
Once the capacity of the links is low compared to the arrival rates of the different customers, the network gets more stressed. Once the network is stressed, the links appear to get more dependent, as the EFP method differs more from the exact solution than in the situation where the links are not stressed.
The EFP method assumes independence of the links; which is not straightforward in small networks; as there are only few classes of customers that use multiple (similar) links. At the moment that the capacities of these links that these few different classes of customers use, the independence cannot be assumed, because of which the EFP method deviates from the exact solution.

The computation time of Erlang Fixed-Point-Method can be approximated to $O(KL/\epsilon)$ (ϵ is the percentage of required accuracy). On the contrary Exact-Product Form solution requires $O(L^K)$ computation time. Hence Exact-Product-Form Solution is easy to compute only for networks of small size where as Erlang-Fixed Point is preferred for larger networks.

A Matlab code

In this appendix, the Matlab code that is used for this project is added.

```
%Declaration of variables


lambda = [0.8 0.8 0.8 0.8];
C = [3 2 4 5];


B = [0.5 0.5 0.5 0.5]; %Initial guess
Ac = 0.0001; %Accuracy
Ae = 1000; %Actual error
n=0; %Iteration counter
Bold = B;
ServerUsage=[1 1 0 0;0 1 1 0; 0 0 1 0;1 0 0 1];%bkl matrix
%ServerUsage=[1  0 0 0;0 1 0 0; 0 0 1 0;0 0 0 1];% bkl matrix

base=max(C);
totalcapacity=sum(C);
l=size(ServerUsage);
for i=1:l(1)
    Bk(i)=sum(ServerUsage(i,:));
end

%start-Erlang Fixed-Point Method
while Ae > Ac

    B(1) = erlangb(C(1), lambda(1)*(1-B(2)) + lambda(4)*(1-B(4)));
    B(2) = erlangb(C(2), lambda(1)*(1-B(1)) + lambda(2)*(1-B(3)));
    B(3) = erlangb(C(3), lambda(3) + lambda(2)*(1-B(2)));
    B(4) = erlangb(C(4), lambda(4)*(1-B(1)));
    maxerror = [abs(B(1)-Bold(1)) abs(B(2)-Bold(2)) abs(B(3)-Bold
        (3)) abs(B(4)-Bold(4))];
    Ae = max(maxerror)*100;
    Bold = B;
    n = n+1;
end

%Compute blocking probability of individual class from blocking
    probability
%of links
b(1)=1-(1-B(1))*(1-B(2));
b(2)=1-(1-B(2))*(1-B(3));
b(3)=1-(1-B(3));
b(4)=1-(1-B(1))*(1-B(4));
```

```
B %Blocking probability of links -Erlang Fixed Point
n% Number of iterations to solve
b%Blocking Probability of classes-Erlanf Fixed point

%End-Erlang Fixed-Point Method




%Start- Brute force calclulation

p=0;
%Initialization of all possible State Space (Valid + Invalid
    Configuration)
for i=0:base
    for j=0:base
        for k=0:base
            for l=0:base
                p=p+1;
                SP(p)=SysState;
                SP(p)=SP(p).Initialize([l,k,j,i],base);%
                    Iniatilization of State Objaect
                SP(p)=SP(p).SetTraffic(lambda);%Configuring tarffic
                     values in the State Object
                SP(p)=SP(p).SetCapacity(C);%Configuring capacity
                    values in the State Object
                SP(p)=SP(p).SetServerUsage(ServerUsage);%
                    Configuring Bkl matrix in the State Object
                SP(p)=SP(p).ComputeValidity;%compute validity of
                    current configuration
                SP(p)=SP(p).ComputeContribution;% Compute product
                    form for each class
                SP(p)=SP(p).computeLinkSet;

            end
        end
    end
end

%Computation of Normalization constant from Valid Set
NormalizationConstant=0;
for i=1:length(SP)
    if(SP(i).Valid)
        NormalizationConstant=NormalizationConstant+SP(i).
            Contribution;
    end
```

```
end

%Computiing Probability of each state from individual contributions and
%from Normalization Constant computed above for all valid sates.
for i=1:length(SP)
    SP(i)= SP(i).SetNormalizationConstant(NormalizationConstant);
    SP(i)= SP(i).ComputeProbability;

end
sumP=0;



%Check if total probability of all valid states is 1.
for i=1:length(SP)
    if(SP(i).Valid)
        sumP=sumP+SP(i).Probability;
    end
end

sumP % Assertion that total probability is 1.
p=0;


%Seperating Valid Statespace
for i=1:length(SP)
    if(SP(i).Valid)
        p=p+1;
        SPnew(p)=SP(i);%for storing only valid configurations
    end
end




Bl=zeros(1,4);%Initilizing Blocking probabilities of each link to 0
for i=1:length(C)
    Bsum=0;
    for j=1:length(SPnew)

        if(SPnew(j).LinkSet(i)==C(i))
            Bsum=Bsum+SPnew(j).Probability; % Summing probailitis of states which reach blocking capcity of the link to find Blocking probability of the corresponding link.
        end
```

```
    end
    Bl(i)=Bsum;
end

%Compute blocking probability of individual class from blocking
    probability
%of links

bl(1)=1-(1-Bl(1))*(1-Bl(2));
bl(2)=1-(1-Bl(2))*(1-Bl(3));
bl(3)=1-(1-Bl(3));
bl(4)=1-(1-Bl(1))*(1-Bl(4));


Bl %Blocking probability of links -Brute Force
bl %Blocking Probability of classes-Brute Force

for j=1:length(SPnew)

    SetV(j,:)=SPnew(j).Set;% Valid set
end
for j=1:length(SPnew)

    SetP(j)=SPnew(j).Probability;% Probalities of valid sets.
end

%End- Brute force calclulation
```

```
%File-SysState.m
%Contains properties and methods of SysState class.(Represents
    state of the system)

classdef SysState
    properties
        Number; %state number j,class number-k
        Value; %Sum of number of server used by each class( nj=sum(
           nk(j) )
        Set;% set of servers used by each class in the current
           state ( Sj=SET{nk(j)} )
        Probability;%Probability of current state pi(nj)
        Valid;%to find if the configuration of current set is valid
            with respect to the capacity constrints
        base; %for storing maximum capacity
        NormalizationConstant; %for storing normalixation constant
           of the whole system
        traffic;% for storing lamda/u array
        Capacity;%for storing capacity array
        ServerUsage;%for storing bkl matrix
        Contribution; % Contribution of current state to the whole
           system
        LinkSet;%occupation on each link
    end
    methods

        %Constructor for creating object
        function S= SysState

            S.Number=0;
            S.Value=0;
            S.base=4;
            S.Valid=0;
            S.Probability=1;
            S.NormalizationConstant=1;
            S.Contribution=1;
        end



        %Initialization and updating prameters
        function S= Initialize(S,set,base)

            S.Number=0;
            S.Set=set;
            S=S.UpdateStateNum;
            S.base=base;
```

```
            S.Value=sum(set);


        end


        function S=UpdateStateNum(S)
            set=S.Set;

            for i=1:length(set)
                S.Number=S.Number+(S.base)^(i-1)*set(i);%compute unique statenumber for current configuration
            end
            S.Number=S.Number+1;
        end

        function S= SetTraffic(S,traffic)
            S.traffic=traffic;
        end

        function S= SetCapacity(S,Capacity)
            S.Capacity=Capacity;
        end

        function S= SetServerUsage(S,ServerUsage)
            S.ServerUsage=ServerUsage;
        end

        function S= computeLinkSet(S)
            S.LinkSet=S.Set*S.ServerUsage;
        end




        %Compute if the curernt state  configuration satisfies the Capacity
        %constraints
        function S= ComputeValidity(S)
            S.Valid=1;
            Nclass=length(S.Set);
            Nlinks=length(S.Capacity);

            for i=1:Nlinks
            sum=0;
            for j=1:Nclass
                sum=sum+S.ServerUsage(j,i)*S.Set(j);
```

```
            end
            if(sum>S.Capacity(i))
                S.Valid=S.Valid*0;
            else
                 S.Valid=S.Valid*1;
            end

            end


        end

        %Compute contribution as a product form of each class
        function S= ComputeContribution(S)
            S.Contribution=1;
            for i=1:length(S.Set)
                S.Contribution=S.Contribution*(S.traffic(i) ^ S.Set
                    (i)/factorial(S.Set(i)));
            end

        end


        %Update NormalizationConstant computed from the statespace
        function S= SetNormalizationConstant(S,
            NormalizationConstant)
            S.NormalizationConstant=NormalizationConstant;
        end


        %Compute Probability of current state from the contribution
            and NormalizationConstant

        function S= ComputeProbability(S)

            S.Probability=S.Contribution/S.NormalizationConstant;
        end


    end

end
```

```
function B = erlangb(N, A)
% ERLANGB Erlang-B blocking probability of a telecommunications systems
% with n servers (channels) and a traffic intensity a
% where ...
% a is lambda * d.
% lambda is average call arrival rate (call/s).
% d is average call duration (s/call). (note: most textbooks talk in terms of mu = 1/d)
%
% elements of a must be real and positive
% n must be a scalar positive integer. However, n = 0 yields the
% (trival) result of 100% blocking irrespective of traffic intensity.
% Reference: "Telecommunications Networks", by Mischa Schwartz, ISBN 0-201-16423-X
%
% Rule of thumb #1: Subscribers can tolerate a busy hour (i.e. peak) blocking
% probability of no more than 2% (landline) to 5% (mobile)
% example:
% lambda=0:0.0001:0.0065; % mean arrival rate (calls per second)
% d=200; % mean duration (seconds per call)
% a = lambda.*d;
% b = erlangb(4, a); % n=4 channels
% plot(a, b)
% The plot shows 2% blocking occurs at approx a=1.1
% The number of subscribers supported is a / m,
% where m is the busy hour intensity for per subscriber.
%
% Rule of thumb #2:  m = ~20% for land lines and ~4% for mobile
% example: b=0.02, n=4 (==> a=1.1), m=0.05 supports ~22 subscribers
%
% Rule of thumb #3:  Average intensity (~proportional to metered revenue) is
% one fifth of busy hour intensity i.e. 0.2*a.
% In the USA, local loops avg ~60 MOU (minutes of use) per day per subscriber.
% However, usage is increasing with popularity of the Internet.
% In the USA, wireless access avg ~12 MOU per day per subscriber,
% again increasing as the price/min drops.
% SEE ALSO INVERLANGB ("inverse" Erlang B) to calculate intensity for given n and b

% This code is freeware as defined by Free Software Foundation "copyleft" agreement. (C)2000 Colin Warwick
% Comments, bugs, MRs to cwarwick@home.com. Offered "as is". No
```

```
    warranty express or implied.
% version 2000-Sep-10
if (length(N)~=1) | (fix(N) ~= N) | (N < 0)
  error('N must be a scalar positive integer');
end
% TODO: test that elements of A are real and positive here?
esum = zeros(size(A));
for ii=0:N
        esum = esum + A .^ ii ./ factorial(ii);
end
B = A .^ N ./ (factorial(N) .* esum);
```