# RoboSurv v1.0 Evaluation, Optimization on Markov Chains

Han Wang

Version 1.0 of Nov. 2019

## Contents

**Abstract**

We develop open source Matlab & Julia software toolbox RoboSurv for calculating and optimizing a bunch of quantities and metrics related to Markov chains

# 1 Introduction of RoboSurv

RoboSurv is intended to calculate and optimize a bunch of quantities and metrics that are related to Markov chains. It is motivated by the use of Markov chains in robotic applications where one or a group of robots randomly move on a graph to perform surveillance tasks.These stochastic surveillance strategies for quickest detection of anomalies or intelligent intruders in network environments. To solve this problem, different algorithms have been proposed. In this package we implement five different algorithms to calculate and optimize related quantities and metrics: mixing time[1], hitting time probability[2], Kemeny constant[2], entropy rate[3] and return time entropy[4].

In the optimization part, for solving non-convex optimization problems, we use fmincon[5] with sqp solver in Matlab and JuMP[6] with Ipopt solver in Julia. Users also are provided with to solve semidefinite programming (SDP) with CVX[7] in Matlab and Convex.jl[8] in Julia. For details of the optimization solver, the users are referred to the above references.

# 2 Installation

RoboSurv is a software toolbox which has both Matlab and Julia versions. Source code can be found at

`https://github.com/HanWang99`

The installation and using method are for Matlab and Julia, and we will introduce the usage respectively.

## 2.1 Matlab Installation

The package has been packed up into a toolbox file, named **Robot Surveillance.mtblx**, so that the users just need to download that into their own workspace and double click it, then matlab will automatically install the toolbox into toolspace. It can be downloaded here:

`https://github.com/HanWang99/RoboSurv/tree/master/Robot-Surveillance-Matlab`

For more details about how to use and manage the toolboxes, reader is referred to

`https://www.mathworks.com/help/matlab/matlab_prog/`
`create-and-share-custom-matlab-toolboxes.html`

The package contains several m-files, user can also download all these files and add them into workspace to use the package.

### 2.2 Julia Installation

The package has also been packed up, readers just need to use e.g. Julia's command

```
Pkg.add("https://github.com/HanWang99/RoboSurv.git")
```

or enter "]" into package mode, then use e.g. Julia's command

```
add https://github.com/HanWang99/RoboSurv.git
```

Then the package can be automatically added to Julia work path. It should be noted that compared with Matlab, Julia doesn't have preinstalled packages, e.g. linear algebra package, optimization package and so on. It may take a long time for users without these necessary packages to install our toolbox because of the installing dependencies process. After successfully installing the package, user need to enter e.g. Julia's command to use

```
using MarkovChian
```

Now all the functions can be used.

## 3 Getting Started

Please type the command

>>test

In Matlab to run the basic example that follows.

To calculate the Kemeny constant[3] of a probability transition matrix, which is defined by

$$K_W = \pi^T (P \circ W) \mathbb{1}_n K, \tag{1}$$

where $K_W$ is weighted Kemeny constant, and $K$ is defined by

$$K = Tr[(I - \Pi^{1/2} P \Pi^{-1/2} + qq^T)^{-1}], \tag{2}$$

where $P$ is the input probability transition matrix, using $\pi$ to represent the stationary distribution of the probability transition matrix, then

$$\Pi = diag[\pi], \tag{3}$$

and

$$q^T = (\sqrt{\pi_1}, ..., \sqrt{\pi_n}), \tag{4}$$

in the test file, weighted matrix and probability transition matrix is defined as:

```
>>W =[0   2   2   6   5   0   0   5;
      7   8   7   4   0   8   8   7;
     10  10   0  10   0   0   4   8;
```

Table 1: Options of Computation and Optimization

| | Computation | Optimization |
|---|---|---|
| Mixing Time | MixingTime | MixingTimeOp |
| Kemeny Constant | Kemeny | KemenyOp |
| Hitting Time | HittingTime | HittingTimeOp |
| Return Time Entropy | ReturnTimeEntropy | ReturnTimeEntropyOp |
| Entropy Rate | EntropyRate | EntropyRateOp |

```
    10     8    10     6     4     8    10     3;
     5     0     0     3     3     5     0     4;
     0    10     0     6     6     0     6     9;
     0     9     2     7     0     9     0     0;
     4     4     6     5     6     5     0     4;]

>>P =[-0.0000  0.1000  0.1000  0.1000  0.1000  0.000  0.0000  0.6000;
     0.1000  0.1000  0.1000  0.4000  0.0000  0.1000  0.1000  0.1000;
     0.6000  0.1000  0.0000  0.1000  0.0000  0.0000  0.1000  0.1000;
     0.1000  0.3000  0.1000  0.1000  0.1000  0.1000  0.1000  0.1000;
     0.1000  0.0000  0.0000  0.1000  0.1000  0.6000  0.0000  0.1000;
     0.0000  0.1000  0.0000  0.1000  0.6000  0.0000  0.1000  0.1000;
     0.0000  0.7000  0.1000  0.1000  0.0000  0.1000  0.0000  0.0000;
     0.1000  0.1000  0.4000  0.1000  0.1000  0.1000  0.0000  0.1000;]
```

Then, the Kemeny constant of the graph is

```
>>F_Kem
F_Kem =

    51.3840
```

## 4 Mathematical Details and Usage

RoboSurv has two integrated function **MC_COMP(X,Option)** and **MC_OP(X,Option)** to call every calculation and optimization function. Available options are listed as follow: It should be noted that, for Matlab, the format of function calling is

F=MC_OP(X, 'Option')

and for Julia, the format is

F=MC_OP(X,"Option")

The differences lie between ″ and "".

## 4.1 Kemeny Constant

### 4.1.1 Kemeny Constant Evaluation

For this part, mathematical details are referred to part three. The function prototype is defined as

Kemeny(P,W)

and then, the users just need to use MC_COMP(P,W,'Kemeny') in Matlab or MC_COMP(P,W,"Kemeny") in Julia. W has a default value of one correspond to non-zero entries in P and zero, otherwise. Return value of the function is a float number.

### 4.1.2 Kemeny Constant Optimization

To optimize Kemeny constant for given adjacent matrix, weighted matrix and stationary distribution, the problem can be formulated by

$$\begin{aligned} \min \quad & (\pi^T(P)\mathbb{1}_n)(Tr[(I-\Pi^{1/2}P\Pi^{-1/2}+qq^T)^{-1}]) \\ \text{subject to} \quad & \sum_{j=1}^{n} P_{ij} = 1, \text{for each} i \in 1,...,n \\ & \pi_i p_{ij} = \pi_j p_{ij}, \text{for each}(i,j) \in E \\ & 0 \le p_{ij} \le 1, \text{for each}(i,j) \in E \\ & p_{ij} = 0, \text{for each}(i,j) \notin E \end{aligned} \tag{5}$$

This problem is a non-convex optimization problem, we first convert the constraint into formal equality constraint, inequality constraint and non-linear constraint.

$$\begin{aligned} A_1 P \le b_1, A_{eq} P = b_e q \\ nonlcon1(P) \le 0 \end{aligned} \tag{6}$$

Using Rushabh_eva(x,PI,W) to calculate the object value which is to be minimized, and then, using fmincon to solve this problem in Matlab. The function prototype is defined as

kemenyOp(A, PI,W)

and then, user just need to use MC_COMP(A,PI,W,'Kemeny') in Matlab or MC_COMP(A,PI,W,"Kemeny") in Julia. W has a default value of one correspond to non-zero entries in P and zero, otherwise. Return value of the function is optimized probability transition matrix and maximum Kemeny constant.

```
[y,f] = fmincon(@(x)Rushabh_eva(x,PI,W),x0,A1,b1,
A_eq1,b_eq1,[],[],@nonlcon1,options);

using JuMP to solve this problem in Julia

model=Model(with_optimizer(Ipopt.Optimizer))
@variable(model,P[i=1:n^2])
register(model, :Rushabh_eva, n^2, Rushabh_eva; autodiff = true)
JuMP.register(model, :nonlcon1, n^2, nonlcon1, autodiff=true)
@objective(model,Min,Rushabh_eva(P...))
@constraint(model,A_eq*P.==b_eq)
@constraint(model,A1*P.<=b1)
@NLexpression(model,my_expr,nonlcon1(P...))
@NLconstraint(model,my_constr,nonlcon1(P...)==0)
JuMP.optimize!(model)
```

## 4.2 Mixing Time

### 4.2.1 Mixing Time Evaluation

Mixing time of a graph is the second largest absolute engine value of the Laplacian matrix, which can be written as

$$\mu(P) = \max_{i=2,\ldots,n} |\lambda_i(P)| = \max\{\lambda_2(P), -\lambda_n(P)\} \tag{7}$$

The function prototype is defined as

```
MixingTime(P)
```

and then, user just need to use MC_COMP(P,'MixingTime') for Matlab or MC_COMP(P,"MixingTime") for Julia. W has a default value of one correspond to non-zero entries in P and zero, otherwise.

### 4.2.2 Mixing Time Optimization

For given adjacent matrix, we can calculate a correspond probability transition matrix with maximum second largest absolute engine value, the problem can be formulated by

$$\begin{aligned} \min \quad & \mu(P) = ||P - (1/n)11^T||_2 \\ \text{subject to} \quad & P \geq 0, P1 = 1, P = P^T \\ & P_{ij} = 0, (i,j) \notin \mathbb{E} \end{aligned} \tag{8}$$

The optimization object is non-convex, but the problem can be rewritten in SDP form as follow

$$\begin{aligned} \text{minimize} \quad & s \\ \text{subject to} \quad & -sI \leq P - (1/n)11^T \leq sI, \\ & P \geq 0, P1 = 1, P = P^T \\ & P_{ij} = 0, (i,j) \notin \mathbb{E} \end{aligned} \tag{9}$$

To solve this kind of problem, CVX in Matlab and Convex.jl in Julia are perfect tools. Actually those are both developed from Prof. Stephan Boyd's group. The function prototype is defined as

MixingTimeOp(A)

and then, user just need to use MC_COMP(A,'MixingTimeOp') for Matlab or MC_COMP(A,"MixingTimeOp") for Julia. W has a default value of one correspond to non-zero entries in P and zero, otherwise. Return value of the function is optimized probability transition matrix and maximum second largest absolute engine value.

## 4.3 Return Time Entropy

### 4.3.1 Return Time Entropy Computation

Return Time Entropy of a graph is defined as

$$\mathbb{J}(P) = \sum_{i=1}^{n} \pi_{ij} \mathbb{H}(T_{ii}) \tag{10}$$

where$\pi_i$ is the $i-th$ entry of stationary distribution, $\mathbb{H}(T_{ii})$ is return time entropy of state $i$, formula is given as

$$\begin{aligned} \mathbb{H}(T_{ii}) &= -\sum_{k=1}^{\infty} \mathbb{P}(T_{ii} = k) \log \mathbb{P}(T_{ii} = k) \\ &= -\sum_{k=1}^{\infty} F_k(i,i) \log F_k(i,i) \end{aligned} \tag{11}$$

$T_{ii}$represents the first time the random walk reaches node $j$ starting from node $i$, that is

$$T_{ij} = \min\{\sum_{k'=0}^{k-1} w_{X_{k'}, X_{k'+1}} | X_0 = i, X_k = j, k \geq 1\} \tag{12}$$

The function prototype is defined as

ReturnTimeEtropy(P,W,yeta)

and then, user just need to use MC_COMP(P,W,yeta,'ReturnTimeEntropy') for Matlab or MC_COMP(P,W,yeta,"ReturnTimeEntropy") for Julia. Weighted matrix W has a default value of one correspond to non-zero entries in P and zero, otherwise. It should be noted that in simulation, we cannot calculate the entropy with $k \to +\infty$, thus, a truncation accuracy parameter $\eta$ is introduced to help calculation. $\eta$ is user defined, but it must lie between 0 and 1.

#### 4.3.2 Return Time Entropy Optimization

For given weighted matrix(or just adjacent matrix), stationary distribution, truncated parameter and lower bound of probability transition matrix, we can calculate optimal probability transition matrix with maximum return time entropy. The problem can be formulated by

$$\begin{aligned} \text{maxmize} \quad & \mathbb{J}_{trunc,\eta}(P) \\ \text{subject to} \quad & P \in P^{\varepsilon}_{\mathbb{G},\pi} \end{aligned} \tag{13}$$

where$\mathbb{J}_{trunc,\eta}(P)$is return time entropy with truncated parameter, which is defined as

$$\begin{aligned} N_\eta &= \lceil \frac{w_m ax}{\eta \pi_{min}} \rceil - 1 \\ \mathbb{J}_{trunc,\eta}(P) &= -\sum_{i=1}^{n} \pi_i \sum_{k=1}^{N_\eta} F_k(i,i)_k(i,i) \end{aligned} \tag{14}$$

and $P^{\varepsilon}_{\mathbb{G},\pi}$ is constraint set of probability transition matrix, that is

$$\begin{aligned} P^{\varepsilon}_{\mathbb{G},\pi} = \{P \in \mathbb{R}^{n\times n} | & p_{ij} \geq \varepsilon \ if(i,j) \in \mathbb{E} \\ & p_{ij} = 0 \ if(i,j) \notin \mathbb{E} \\ & P\mathbb{1}_n = \mathbb{1}_n, \pi^T P = \pi^T\} \end{aligned} \tag{15}$$

We use fmincon with sqp solver in matlab to solve this problem, and JuMP with Ipopt solver in Julia, respectively. The function prototype is defined as

ReturnTimeEntropyOp(A, PI ,W, epsilon , yeta)

and then, user just need to use MC_COMP(A,PI,W,epsilon,yeta,'ReturnTimeEntropyOp') for Matlab or MC_COMP(A,PI,W,epsilon,yeta,"ReturnTimeEntropyOp") for Julia. Weighted matrix W has a default value of one correspond to non-zero entries in P and zero, otherwise. Return value of the function is optimized probability transition matrix and maximum return time entropy.

User should be noticed that, if a very small $\eta$ is chosen, the optimization time can be extremely long. An acceptable value is usually bigger than 0.01.

### 4.4 Entropy Rate

#### 4.4.1 Entropy Rate Computation

Entropy rate of a probability transition matrix is defined as

$$H_\pi(P) = -\sum_{i,j=1}^{n} \pi_i p_{ij} \log p_{ij} \tag{16}$$

Where $\pi\_i$ is the i-th value of stationary distribution and $p\_ij$ represents the entry of probability transition matrix P. The function prototype is defined as

EntropyRate(P)

and then, user just need to use MC_COMP(P,'EntropyRate') for Matlab or MC_COMP(P,"EntropyRate") for Julia.

### 4.4.2 Entropy Rate Optimization

For given adjacent matrix A and stationary distribution PI, we can calculate optimal probability transition matrix P with maximum entropy rate. The problem can be modeled as

$$\begin{aligned} \max \quad & H_\pi(P) \\ \text{subject to} \quad & P \geq 0 \\ & p_{ij} = 0, if\ a_{ij} = 0 \\ & P\mathbb{1}_n = \mathbb{1}_n \\ & \pi^T P = \pi^T \end{aligned} \tag{17}$$

We use fmincon with sqp solver in matlab to solve this problem, and JuMP with Ipopt solver in Julia, respectively. The function prototype is defined as

EntropyRateOp(A, PI)

and then, user just need to use MC_COMP(A,PI,'EntropyRateOp') for Matlab or MC_COMP(A,PI,"EntrpyRateOp") for Julia. Weighted matrix W has a default value of one correspond to non-zero entries in P and zero, otherwise. Return value of the function is optimized probability transition matrix and maximum entropy rate.

## 4.5 stationary Distribution

The stationary distribution is the eigenvector corresponding to the eigenvalue 1 of the probability transition matrix. The function prototype is defined as

Stadis(P)

and then, user just need to use MC_COMP(P,'Stadis') for Matlab or MC_COMP(P,"Stadis") for Julia. For Julia, the return value is a **vector**, but not a **one-column matrix**!

# 5 Technical Details and Check Function

There are several parameter check functions in MC_COMP and MC_OP. We use some technical method to realize successful optimization and accelerate code speed. Details can be found in following subsections.

Table 2: Check Functions

| Function | Julia | Matlab |
|---|---|---|
| Legal Markov Chain | ✓ | ✓ |
| Irreducible | ✓ | ✓ |
| Legal Option | ✓ | ✓ |
| Default Weighted Matrix | ✓ | ✓ |
| Symmetric Matrix (some case) | ✓ | ✓ |
| Dimension Match | ✓ | ✓ |
| Legal stationary Distribution | ✓ | ✓ |
| Integer and non-negative(duration) | ✓ | ✓ |

### 5.1 Variable and Method Definition

While solving optimization problem with JuMP in Julia, the variables are declared as "VariableRef" type, and in Julia, every entries in a matrix must be in same type. However, while solving our optimization problem, constant and variable always coexist in a same matrix. Actually, our variable should be a matrix which is not supported in JuMP. We use the following skills to solve this problem.
1.Split matrix into column vector.

```
for i=1:n*n
    if rem(v_place[i+1],n)!=0
P_opt[convert(Int64,rem(v_place[i+1],n)),convert(Int64,floor(v_place[i+1]/n))+1]=x[
    else
        P_opt[n,convert(Int64,floor(v_place[i+1]/n))]=x[i]
    end
end
```

Then, we use @NLObjective macro in JuMP, which can accept user-defined objective function

```
@NLobjective(model,Max,myfun(X...))
```

We use X... as input parameter because that the dimension of input matrix is not fixed.
2.Define **Supervariable** type.
Supervariable is a combination of VairableRef and Float, so it can be used to declare a matrix with mixed types of data.

```
SuperVariable=Union{Real,VariableRef}
```

Then, we need to overload basic corresponding function with our data type.

```
Base.zero(::Type{SuperVariable})=0
Base.zero(::Type{VariableRef})=0
```

Now, code can operate successfully. The reason for the overload of Base.zero() is this function will operate during optimization. Without it, an error will throw out.

3.Use sparse matrix.

In our problem, some matrix are actually sparse with a high dimension. Thus, we convert some matrix into sparse matrix to reduce code run time. That is why SparseArrays package exits in the dependencies of our package in Julia.

### 5.2 Legal Markov Chain

We provide a function named Markov_or_not() to check whether the input probability transition matrix has all sum of rows equal 1 or not.

Matlab test:

```
>> P=[1/2 1/2 0;
1/3 1/3 1/3;
1/4 1/4 1/4;]

P =

    0.5000    0.5000         0
    0.3333    0.3333    0.3333
0.2500    0.2500    0.2500
>> Markov_or_not(P)
ERROR: Markov_or_not (line 5)
the matrix you input is not an illegal probability transition matrix
```

Julia test

```
julia> P=[1/2 1/2 0;1/3 1/3 1/3;1/4 1/4 1/4]
3 3 Array{Float64,2}:
 0.5       0.5       0.0
 0.333333  0.333333  0.333333
 0.25      0.25      0.25
```

```
julia> Markov_or_not(P)
ERROR: the matrix you input is an illegal probability transition matrix
Stacktrace:
 [1] Markov_or_not(::Array{Float64,2}) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\src\Markov_or_not.jl:16
 [2] top-level scope at none:0
```

### 5.3 Irreducibility

We provide a function named Irreducible_or_not(P) to check whether the input probability transition matrix is irreducible or not. To realize this function, we use the concept reachability.

$$A_{reach} = \sum_{i=1}^{n} A^i \tag{18}$$

If every entries of A_reach is bigger than 0, then there must exist one path between every two nodes for graph G(A),, which means that adjacent matrix A is irreducible.
Matlab test:

```
>> A=[1 1 0;0 1 0;1 1 1]

A =

     1     1     0
     0     1     0
     1     1     1

>> Irreducible_or_not(A)
ERROR: Irreducible_or_not (line 14)
the matrix you have input is reducible
```

Julia test

```
julia> A=[1 1 0;1 1 0;1 1 1]
3 3 Array{Int64,2}:
 1 1 0
 1 1 0
 1 1 1

julia> Irreducible_or_not(A)
ERROR: the matrix you have input is reducible!
```

```
Stacktrace:
 [1] Irreducible_or_not(::Array{Int64,2}) at C:\Users\hp
\.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\
src\Irreducible_or_not.jl:28
 [2] top-level scope at none:0
```

### 5.4 Legal Option

We provide option check for MC_COMP and MC_OP, if user has input a non-exist option, one error will throw out.
Matlab test:

```
>> MC_OP([1 1;1 1],'WrongOption')
ERROR: MC_OP (line 63)
please input legal options!
```

Julia test:

```
julia> MC_OP([1 1;1 1],"WrongOption")
ERROR: please input legal options!
Stacktrace:
 [1] MC_OP(::Array{Int64,2}, ::Vararg{Any,N} where N) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\src\MC_OP.jl:94
 [2] top-level scope at none:0
```

### 5.5 Default Weighted Matrix

As mentioned before, we support a default weighted matrix for MC_COMP and MC_OP. This is realized by count the number of input parameter.
Matlab test:

```
>> P=[0 1/2 1/2;1/2 0 1/2;1/2 1/2 0]

P =

         0    0.5000    0.5000
    0.5000         0    0.5000
0.5000    0.5000         0
>> MC_COMP(P,'Kemeny')

ans =
```

```
2.3333

>> MC_COMP(P,[0  1  1;1  0  1;1  1  0],'Kemeny')

ans =

2.3333
```

Julia test:

```
julia> P=[0  1/2  1/2;1/2  0  1/2;1/2  1/2  0]
3  3   Array{Float64,2}:
 0.0   0.5   0.5
 0.5   0.0   0.5
 0.5   0.5   0.0
julia> MC_COMP(P,"Kemeny")
2.3333333333333335
julia> MC_COMP(P,[0  1  1  ;1  0  1;1  1  0],"Kemeny")
2.3333333333333335
```

### 5.6 Symmetric Matrix

In some cases, the input probability transition matrix and adjacent matrix are required to be symmetric, e.g. mixing time computation.
Matlab test:

```
>> P=[1/2  1/2  0;1/3  1/3  1/3;1/4  1/2  1/4]

P =

    0.5000    0.5000         0
    0.3333    0.3333    0.3333
    0.2500    0.5000    0.2500

>> MC_COMP(P,'MixingTime')
ERROR: MC_COMP (line 51)
please input a symmetric probability transition matrix
```

Julia test:

```
julia> P=[1/2  1/2  0;1/3  1/3  1/3;1/4  1/2  1/4]
```

```
3 3  Array{Float64,2}:
 0.5       0.5       0.0
 0.333333  0.333333  0.333333
 0.25      0.5       0.25

julia> MC_COMP(P,"MixingTime")
ERROR: please input a symmetric probability transition matrix
Stacktrace:
 [1] MC_COMP(::Array{Float64,2}, ::Vararg{Any,N} where N) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\src\MC_COMP.jl:75
 [2] top-level scope at none:0
```

### 5.7 Dimension Match

We provide dimension check between dimension check between probability transition matrix and weighted matrix for MC_COMP as well as between adjacent matrix and weighted matrix for MC_OP
Matlab test:

```
>> P=[1/2  1/2;1/2  1/2]

P =

    0.5000    0.5000
    0.5000    0.5000

>> W=[1  2  3;4  5  6;7  8  9]

W =

     1     2     3
     4     5     6
     7     8     9

>> MC_COMP(P,W,'Kemeny')
ERROR: MC_COMP (line 127)
the dimension of probability transition matrix and weighted
matrix doesnt match
```

Julia test:

```
julia> P=[1/2  1/2;1/2  1/2]
2  2   Array{Float64,2}:
 0.5   0.5
 0.5   0.5

julia> W=[1  2  3;4  5  6;7  8  9]
3  3   Array{Int64,2}:
 1   2   3
 4   5   6
 7   8   9

julia> MC_COMP(P,W,"Kemeny")
ERROR: the dimension of probability transition matrix and weighted
matrix doesnt match
Stacktrace:
 [1] MC_COMP(::Array{Float64,2}, ::Vararg{Any,N} where N) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain
 \wA3Hc\src\MC_COMP.jl:173
 [2] top-level scope at none:0
```

### 5.8 Legal stationary Distribution

Input stationary distribution must fulfill:
1. Sum equals to one
2. Every entry is bigger than zero
3. Column vector
Matlab test:

```
>> PI=[1/2  1/3  1/4]

PI =

    0.5000    0.3333    0.2500

>> A=[1  1  1;1  1  1;1  1  1]

A =
```

```
     1     1     1
     1     1     1
     1     1     1
>> MC_OP(A,PI,'EntropyRateOp')
ERROR MC_OP (line 83)
the stationary distribution must be a column vector

>> PI=[1/2;1/3;1/4]

PI =

    0.5000
    0.3333
    0.2500

>> MC_OP(A,PI,'EntropyRateOp')
ERROR MC_OP (line 86)
please input legal stationary distribution
```

Julia test:

```
julia> PI=[1/2 1/3 1/4]
3-element Array{Float64,2}:
 0.5
 0.3333333333333333
 0.25

julia> A=[1 1 1;1 1 1;1 1 1]
3 3  Array{Int64,2}
 1  1  1
 1  1  1
 1  1  1

julia> MC_OP(A,PI,"EntropyRateOp")
ERROR: the stationary distribution must be a column vector
Stacktrace:
 [1] MC_OP(::Array{Int64,2}, ::Vararg{Any,N} where N) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\src\MC_OP.jl:123
```

```
 [2] top-level scope at none:0
julia> PI=[1/2;1/3;1/4]
3-element Array{Float64,1}:
 0.5
 0.3333333333333333
 0.25

julia> MC_OP(A,PI,"EntropyRateOp")
ERROR: please input legal stationary distribution
Stacktrace:
 [1] MC_OP(::Array{Int64,2}, ::Vararg{Any,N} where N) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\src\MC_OP.jl:126
 [2] top-level scope at none:0
```

### 5.9 Integer and non-negative(duration)

The duration should be a positive integer.
Matlab test:

```
>> tau=-10

tau =

   -10

>> MC_OP(A,W,tau,'HittingTimeOp')
ERROR: MC_OP (line 155)
the duration must be a non-negative integer
```

Julia test:

```
julia> tau=-10
-10

julia> MC_OP(A,W,tau,"HittingTimeOp")
ERROR: the duration must be a non-negative integer
Stacktrace:
 [1] MC_OP(::Array{Int64,2}, ::Vararg{Any,N} where N) at C:\Users\hp
 \.juliapro\JuliaPro_v1.1.1.1\packages\MarkovChain\wA3Hc\src\MC_OP.jl:216
 [2] top-level scope at none:0
```

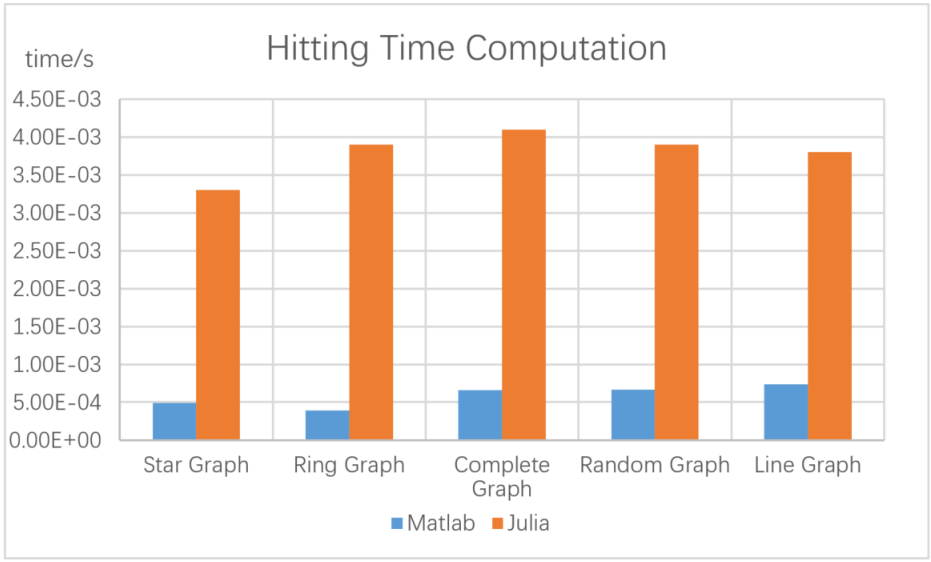


Figure 1: Hitting Time Computation

## 6 Efficiency Computation

Code operating time can be found in Fig.1-10.

## 7 Reference

[1] Boyd S, Diaconis P, Xiao L. Fastest mixing Markov chain on a graph[J].SIAM review, 2004, 46(4): 667−689.
[2] Patel R, Agharkar P, Bullo F. Robotic surveillance and Markov chains with minimal weighted Kemeny constant[J]. IEEE Transactions on Automatic Control, 2015, 60(12): 3156−3167.
[3] George M, Jafarpour S, Bullo F. Markov chains with maximum entropy for robotic surveillance[J]. IEEE Transactions on Automatic Control, 2018, 64(4): 1566−1580.
[4] Duan X, George M, Bullo F. Markov chains with maximum return time entropy for robotic surveillance[J]. IEEE Transactions on Automatic Control, 2019.
[5] https://www.mathworks.com/help/optim/ug/fmincon.html
[6] http://www.juliaopt.org/JuMP.jl/v0.14/
[7] http://cvxr.com/cvx/
[8] https://www.juliaopt.org/Convex.jl/dev/

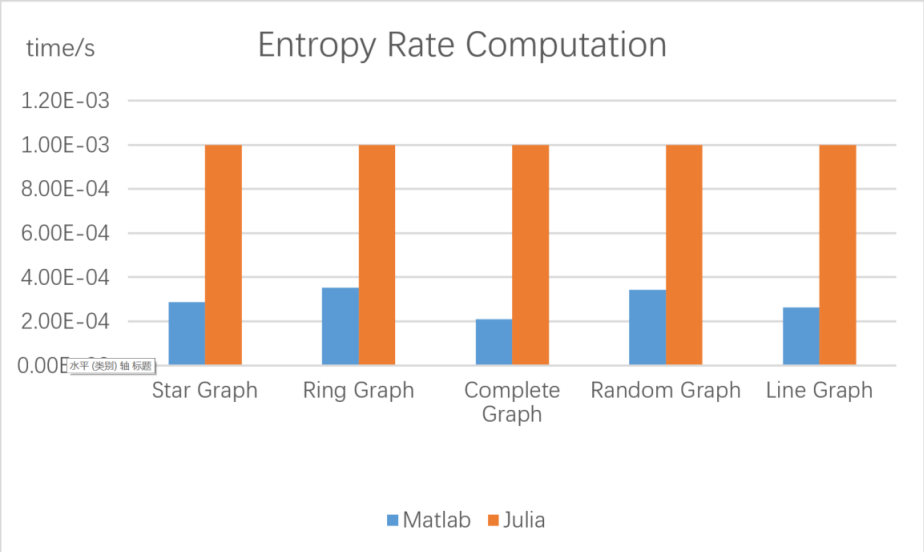


Figure 2: Entropy Rate Computation

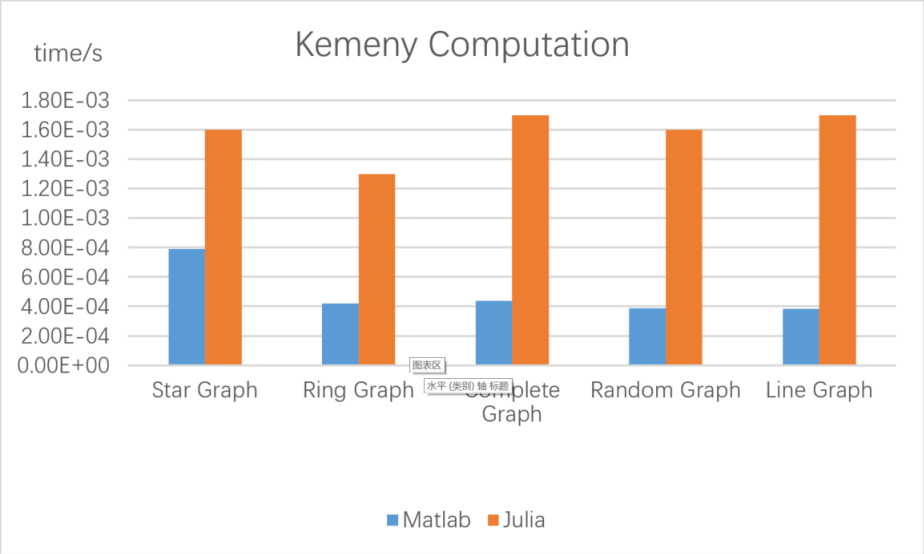


Figure 3: kemeny Constant Computation

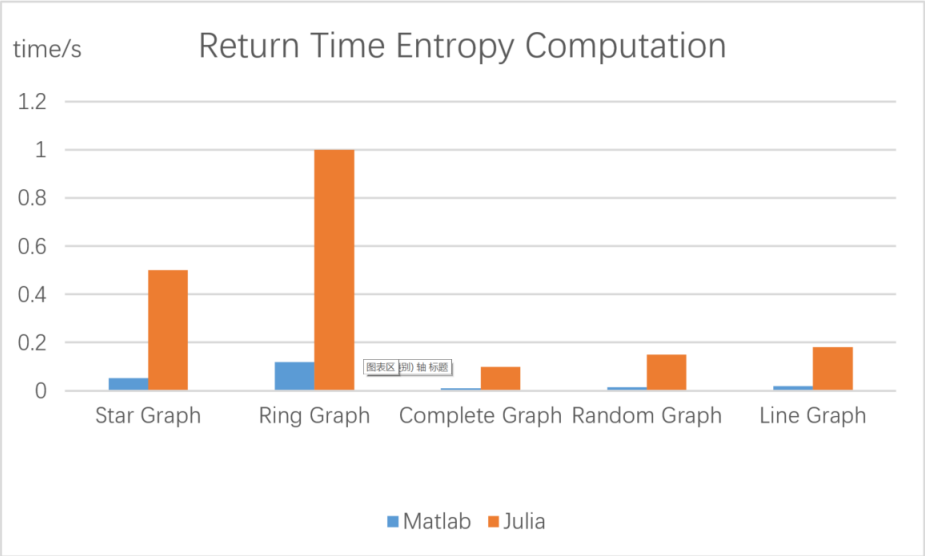


Figure 4: Return Time Entropy Computation

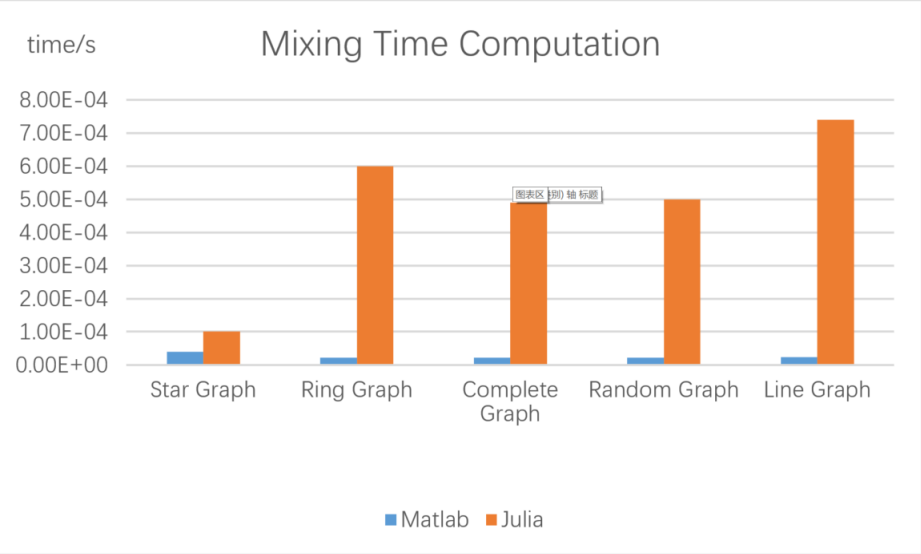


Figure 5: Mixing Time Computation

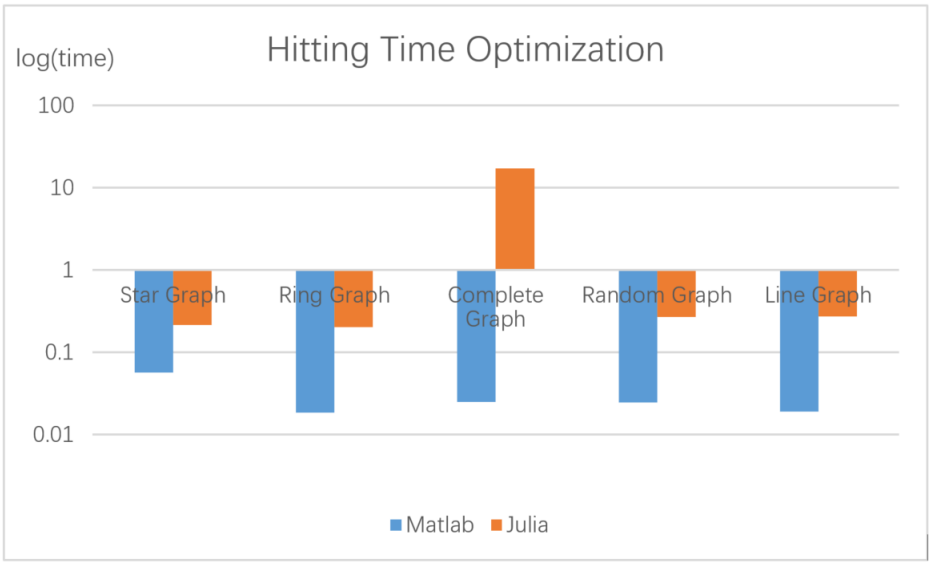


Figure 6: Mixing Time Optimization

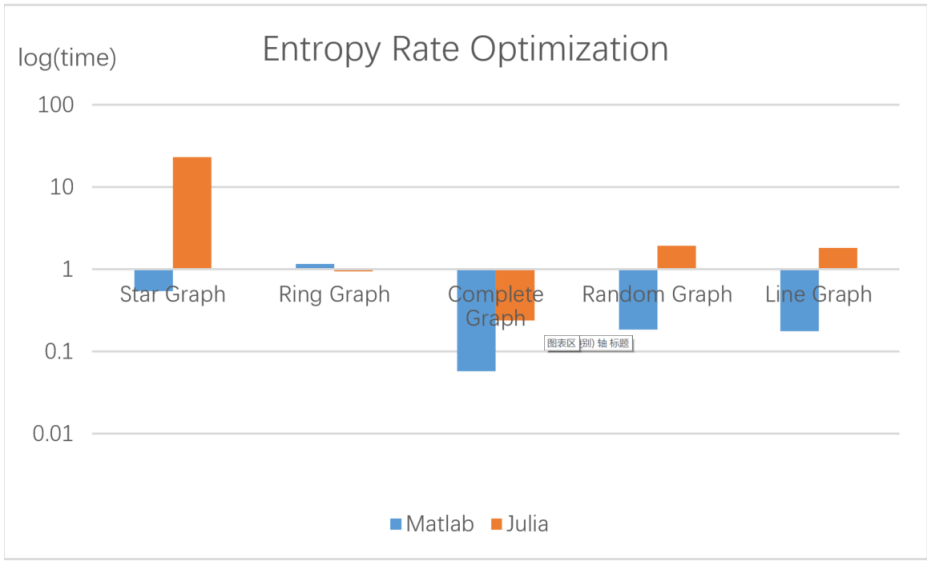


Figure 7: Entropy Rate Optimization

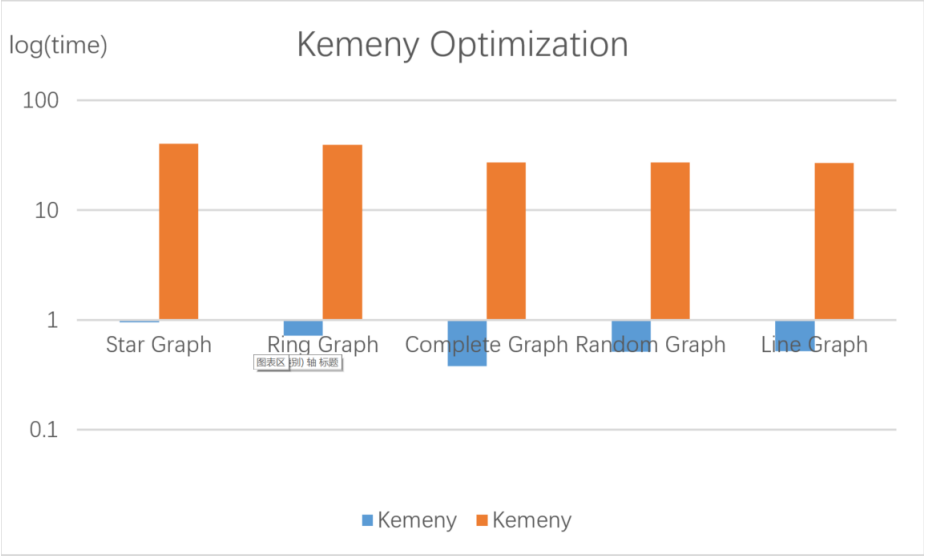


Figure 8: Kemeny Constant Optimization

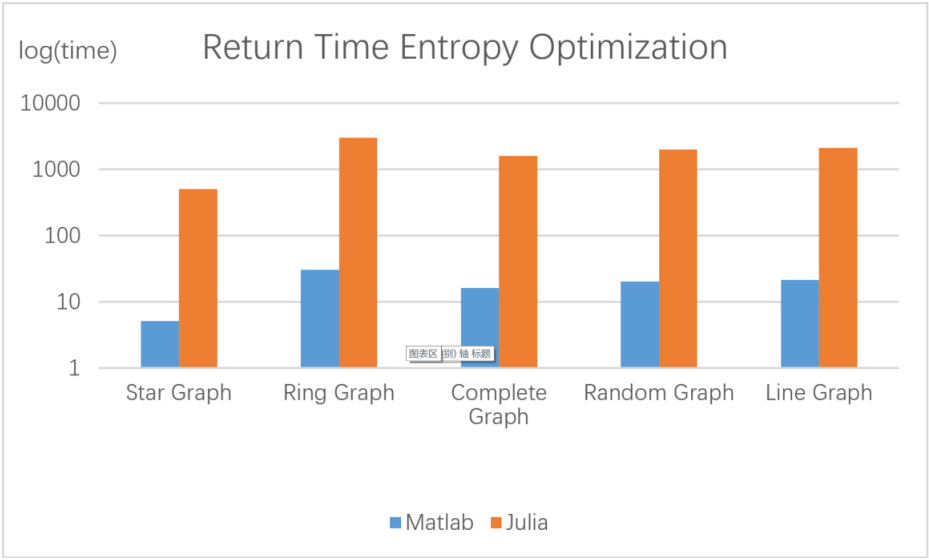


Figure 9: Return Time Entropy Optimization

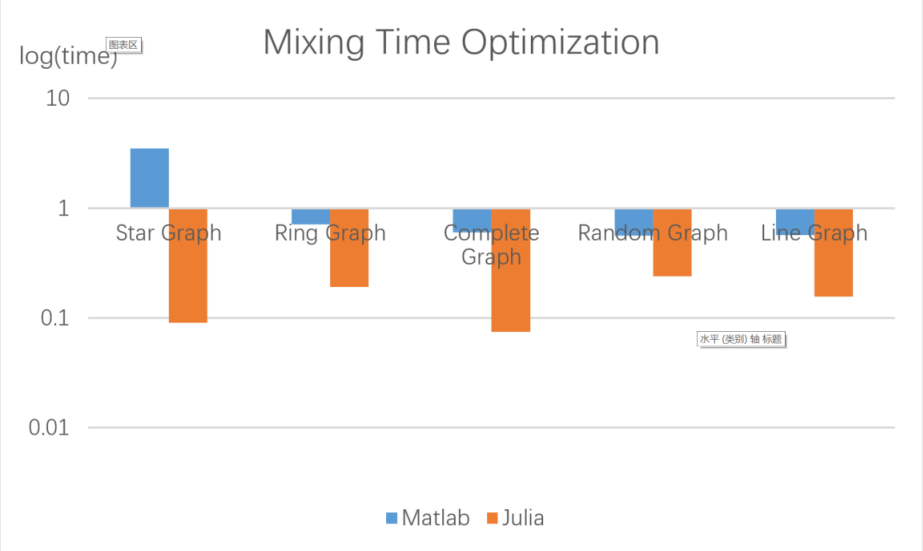


Figure 10: Mixing Time Optimization