Mathematica Notebooks for CS5024

CS6104 Symbolic Computation - Final Project

Michael Duckett and Ted Qian

Table of Contents

1 Mathematica Notebooks for CS5204

1.1. Overview

1.1.1. Abstract

The department of Computer Science at Virginia Polytechnic Institute and State University offers three core classes at the graduate level which every student is required to take. They are CS5014 - Introduction to Experimental Methods, CS5024 - Models and Analysis, and CS5034 - Computational Models. All courses require students to perform heavy computation. In particular, various problems in CS5024 become computationally complex very quickly as the size of the problem increase. It is our goal to build a series of Mathematica notebooks which will off-load the computational cumbersomeness of large problems from students. Therefore students will be able to analyze more complex problems, without the worries of complex computation.

We developed Mathematica notebooks in three areas of CS5024: Markov chains, queuing systems, and Jackson networks. The notebooks are designed to be companions for the students and are not intended to solve problems from scratch.

1.1.2. General Use

The system consists of three independent notebooks: Markov.ma, queue-theory.ma, and Jackson.ma. Each notebook implements problem solving techniques found in the course lecture notes. Therefore, each notebook assumes the user has some foreknowledge of the material contained within the notebooks.

In order to use the notebooks, the user must have all the notebooks and all supporting files in the current directory. This allows Mathematica to load any external rules or definitions that it may need to complete a computation. Documentation on how to use the various functions are contained within the notebooks.

The purpose of the Markov Chain notebook is to allow the user to input a transition probability matrix. The notebook then computes the following:

- Generate a graph representation the of chain
- Calculate reducibility and reachability
- Calculate stationary probability distribution
- Calculate convergence rates.

The function of the Queuing Theory notebook is to let Mathematica know about various models of queues. The student enters various parameters depending on the model from a set of the following models:

- M/M/1
- M/M/1/K
- M/M/c

- M/M/c/c
- M/M/∞
- M/M/1/K/K.

Mathematica does the following:

- Compute L_q, L_s, W_q, W_s, L, W, p_0, p_n
- Show plots of how variance in parameters such as μ, ρ, and a affect queuing system in terms of above parameters
- Produce results for Erlang's Loss and Delay Formulas when appropriate.

When using the Jackson Networks notebook, the user is to input the network connectivity matrix, external inputs to the system, departure rates and the number of servers for each queue. Mathematica then performs the following tasks:

- Calculate the effective arrival rates
- Compute L_q, L_s, W_q, W_s, L, W, p_0, p_n
- Compute the probabilitie of one customer in the system.

1.1.3. Support Files

The following support files are included in the package and are needed primarily for the graphics routines of the notebooks. It is important to note that Mathematica must be spawned from the directory which contains these support files. Otherwise, Mathematica will not be able to find the support files.

- Arrow.m - This package comes from the Mathematica Math Sources home page. It contains the primitives for generating arrows in the Markov Chain and Jackson network graphs.
- Digraph.m - This package consists of all the functions necessary to implement the drawing of the Markov Chain graphs and the Jackson Network graphs.
- Queue.m - This package contains the necessary functions to implement the plotting routines. These routines are used to display the various plots of L, L_q, etc. The function to calculate Erlang's delay formuala and Erlang's loss formula are found within this package too.

1.2. Known Bugs

During the implementation of the notebooks, we found several bugs with unix version 2.2 of the Mathematica notebook interface. The notebook interface sometimes does not display all of the graphics primitives for the

Markov chain graph or the Jackson network graph. This problem is corrected by re-executing the functions which display these graphs.

2 Markov Chains

In[1]:=
```
*)
<<digraph.m
(*
```

2.1. Input Transition Probability Matrix

A Markov chain with stationary transition probability can be represented by a transition probability matrix. Each element(i,j) in the matrix is a constant, independent of time, and represents the one step transition probability from state i to state j.

2.1.1. Example 1 -

Below is an example of a transition probability matrix with probabilities expressed in numerical values.

In[1]:=
```
*)
transprobability = { { 0.15, 0.40, 0.45, 0.},
                     { 0.50, 0.,   0.50, 0.},
                     { 0.,   0.,   0.,   1.},
                     { 0.80, 0.,   0.20, 0.} };
MatrixForm [ transprobability ]

(*
```

Out[1]=
```
0.15   0.4   0.45   0.

0.5    0.    0.5    0.

0.     0.    0.     1.

0.8    0.    0.2    0.
```

2.1.2. Example 2 -

Below is an example of a transition probability matrix with probabilities expressed in symbolic expression.

In[1]:=
```
transprobability = {{1-p0,p0},{p1,1-p1}};

MatrixForm[transprobability]
```

Out[1]=
```
1 - p0   p0

p1       1 - p1
```

2.1.3. User Input

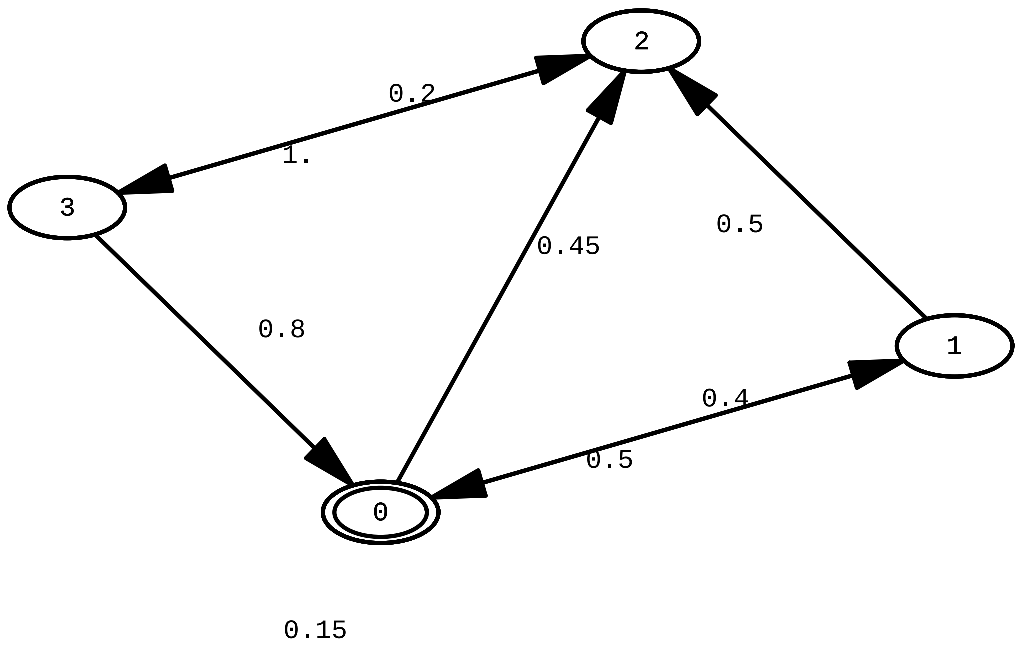
In[1]:=
```
ClearAll[r,p];
(*r = 1/2
p = 1/10*)
transprobability = ( ((1-r)^2, 2 r (1-r), r^2), ((1-r)p, (1-r)(1-p) + r p,
                    r(1-p)),(p^2,2 p (1-p), (1-p)^2));

MatrixForm [ transprobability ]
```

```
Out[1]=
               2                                          2
        (1 - r)               2 (1 - r) r                r

        p (1 - r)             (1 - p) (1 - r) + p r      (1 - p) r

         2                                                       2
        p                     2 (1 - p) p                (1 - p)
```

2.2. Visualization of the Markov chain

The function computegraph will produce a graphic image of the Markov chain from the transition probability matrix. An one step transition back to the same node in the Markov chain is represented by an inscribed circle at the node where the cycle is located. The Mathematica notebook interface seems to contain bugs in its graphical display functions. The computegraph function might require multiple execution to generate the correct graphical output.

```
In[1]:=  *)
         computegraph [ transprobability ];

         (*
```



2.3. Limiting Probabilities

Calclimitingprobabilities computes the limiting probabilities for the Markov Chain. The function assumes an irreducible, aperiodic and time homogenous Markov chain. Then by Thereom 4.4.4 [Allen], the limiting probabilities always exist and are independent of the initial state probability distribution. Thus the function computes the correct limiting probabilities for the input transition probability matrix. The function outputs the results in column form with p0 at the top and pn at the bottom.

The following cell contains the Mathematica code for calclimitingprobability. The cell is closed when the notebook is first opened.

In[1]:=
```
*)
calclimitingprobability[transprobability_]:=
Block[{pimatrix, i, b, c},
     pimatrix = transprobability;
     Do [ pimatrix [[ i, i ]] = pimatrix [[ i, i ]] - 1,
     { i, 1, Length[pimatrix] } ];
     pimatrix = Transpose [ pimatrix ];
     Do [ pimatrix [[ 1, i ]] = 1, { i, 1, Length[pimatrix] } ];

     b = Table[ 0, {i, 1, Length[transprobability]}];
     b [[ 1 ]] = 1;
     c = LinearSolve[pimatrix, b];
     c = MatrixForm[ Simplify [c] ];
     Print [ c ]
]
(*
```

In[2]:=
```
*)
calclimitingprobability [ transprobability ]
(*
```

```
0.330579

0.132231

0.268595

0.268595
```

■ 2.4. Probabilities of Recurrence(after n step transitions)

calcnstep calculates the probabilities of ever returning to the same state within stp step. If the probability transition matrix is numerical, plotrecurrence plots the probabilities of returning to the same state as a function of steps up to twenty steps for each state.

In[1]:=
```
*)
calcrecurrence[transprobability_,n_,ii_]:= Module[ {idx1, k, temptp, s, size },

    size = Length [ transprobability ];
     temptp = transprobability;

     Do[ temptp[[idx1, ii]] = 0,
         {idx1, 1, size}
       ];

     s = Sum[ MatrixPower[temptp, k] . transprobability,
              {k, 0, n-1}
            ];
     Return[Simplify[Chop[N[s[[ii,ii]]]]]];
]

calcnstep [ steps_, tp_ ] := Module [ {size},

  size = Length [ tp ];

  Do [ Print["f",Subscript[i-1],"= ",calcrecurrence[transprobability, steps, i]],
       {i,1,size}
     ]
]

plotrecurrence [ tp_ ] := Module [ {size, invalidflag,i,j, state1, step1},

  invalidflag = 0;
  size = Length [ tp ];

  Do [ If [ NumberQ [ tp[[i,j]] ],
            {},
            invalidflag = 1
          ],
       {i,1,size}, {j,1,size}
     ];

  If [ invalidflag == 1,
       Print [ "plotrecurrence only computes on numerical Markov chains." ],
       {
              Plot3D[calcrecurrence[tp,step1,state1],
                     {step1,1,20}, {state1,1,size},
                     PlotPoints -> { 20, size},
                     ViewPoint -> {-2,-2,2},
                      AxesLabel->{"Steps","Node","Probability of Recurrence"},
                      DefaultFont ->{"Helvetica", 10}, Axes->True]
       }
     ]
]


(*
```

In[2]:=
```
*)
stp = 2;
calcnstep [ stp, transprobability ];
plotrecurrence [ transprobability ]
(*
```

```
f = 0.35
 0
f = 0.2
 1
f = 0.2
 2
f = 0.2
 3
```

■ 2.5. Reducibility

Calcreducibility attempts to determine if the Markov chain is irreducible. It calculates up to two hundred steps to determine if a state is irreducible.

In[1]:=

```
*)
calcreducibility[transprobability_] :=

  Module [{counter, temptransprobability, maxcounter, invalidflag, size},

    invalidflag = 0;
    size = Length[transprobability];

     Do [ If [ NumberQ [ transprobability [[i,j]] ],
               {},
               invalidflag = 1
             ],
          {i,1,size}, {j,1,size}
        ];

     If [ invalidflag == 0,
          {
           Print["Original transition matrix: P",
                  Superscript["(1)"],"=", MatrixForm[transprobability]];

           counter = 1;
           maxcounter = 200;

           temptransprobability = transprobability;

           While[ Min[temptransprobability] == 0 && counter < maxcounter,
             {
             temptransprobability = temptransprobability . transprobability;
              counter = counter + 1
              }
            ]

       Π   If[ Min[temptransprobability] > 0,
               {
                Print["This Markov chain is irreducible."];
                 Print["A transition can be made between any two states in ",
                        counter," steps."];
                 Print["The multi-step positive matrix: P",Superscript["("],
                          Superscript[counter],Superscript[")"],"=",
                          MatrixForm[temptransprobability]]
                 },
                Print["There are states that cannot be reached with less than or equal to ",
                          maxcounter," steps."]
            ]
           },
           {
            Print [ "This function does not compute on symbolic transition probability" ];
            Print [ "matrices." ];
           }
     ]
]
(*
```

In[2]:=

```
*)
calcreducibility[transprobability];
(*
```

```
                              (1)
Original transition matrix: P    =0.15   0.4    0.45   0.

                                  0.5    0.     0.5    0.

                                  0.     0.     0.     1.

                                  0.8    0.     0.2    0.
This Markov chain is irreducible.
A transition can be made between any two states in 3 steps.
                                  (3)
The multi-step positive matrix: P    =

  0.423375   0.089      0.220125   0.2675

  0.51125    0.03       0.23375    0.225

  0.12       0.32       0.36       0.2

  0.338      0.048      0.254      0.36
```

2.6. Analyze Convergence Rate

calcconvergencerate function returns the rate at which deviation from its limiting probability approaches zero. This rate tell us how rapidly the Markov chain "mixes". dd[i] is the deviation at time 0 for state i and nn is the current time.

In[1]:=

```
*)
calcconvergencerate[localtransprobability_]:=

Block[{idx2, ss, dd, delta, nn, ff, size, invalidflag },

    size = Length[ localtransprobability ];
    invalidflag = 0;

    Do [ If [ NumberQ [ localtransprobability [[i,j]] ],
              invalidflag = 1,
              {}
            ],
            {i,1,size}, {j,1,size}
       ];

    If [ invalidflag == 0,
         {
            delta=Array[dd, size];

            ss[1] = - dd[1];
            ss[idx2_] := ss[idx2 - 1] - dd[idx2];

            reduceonedelta = dd[ size ] -> ss[size-1];

            ff = delta . MatrixPower[localtransprobability, nn];
            ff = ff /. reduceonedelta;
            ff = MatrixForm[Simplify[ff]];

            Print[ff]
          },
          {
            Print [ "This function only computes on symbolic transition probability"];
            Print [ "matrices."]
          }
      ]
]
(*
```

```
In[2]:=  *)

         calcconvergencerate[transprobability];
         (*
```

```
This function only computes on symbolic transition
  probability
matrices.
```

3 Queuing Theory

```
In[1]:=  *)
         <<queue.m
         (*
```

■ 3.1. M/M/1

■ 3.1.1. M/M/1 - Solution of Queue Parameters

To use the system, input the number of customers under rulen for probability of the desired number of customers in the system. Rule1 and rule2 are reserved for the given(known) values of the queuing system. At least two independent values are expected. For example, lambda and mu can be used to solve the queuing system.

```
In[1]:=  ClearAll[ a, k, p0, pk, lambdaa, lambda, mu, len, lens, lenq, ws, w, wq, n, pn];
         ClearAll[mylist,pn, n];
         rulen = n -> n;
         rule1 = rule1init -> len == 1;
         rule2 = rule2init -> wq==1/20;
```

```
In[2]:=
         mylist=Sort[Flatten [Simplify [ Solve [{
              rule1init /. rule1,
              rule2init /. rule2,
              len == (lambda/mu)/(1-(lambda/mu)),
              lens == lambda/mu,
              lenq == (lambda/mu)^2/(1-(lambda/mu)),
              ws == 1/mu,
              w == 1/(mu*(1-(lambda/mu))),
              wq == w - ws
              },
              {lambda,mu,ws,w,wq, len,lens,lenq },
              {lambda,mu,ws,w,wq, len,lens,lenq } ]]]];
              Print[TableForm[mylist]];
         pn = (1- mylist[[1,2]]/mylist[[5,2]])(mylist[[1,2]]/mylist[[5,2]])^n /. rulen;
         Print["The probability of ", n /. rulen, " customer in the system pn: ", pn];
```

■ 3.1.2. M/M/1 - Visualization

The function mm1lenplot will generate surface plots of L, Lq and Ls versus lambda and mu ranging from 0 to lambda-max and 0 to mu-max. The function mm1waitplot generates surface plots of W, Wq and Wq. The input values are the same for mm1waitplot.

mm1lenplot [lambda-max, mu-max]

mm1waitplot [lambda-max, mu-max]

```
In[1]:=  mm1lenplot [ 20, 20 ];
         mm1waitplot [ 20, 20 ];
```

■ 3.2. M/M/1/K

To use the system, input the limit of the number of customers under rulek and the number of customers under rulen for probability of the desired number of customers in the system. Rule1 and rule2 are reserved for the given(known) values of the queuing system. At least two independent values are expected. For example, lambda and mu can be used to solve the queuing system.

3.2.1. Example 1(lambda and mu are not equal.)

```
In[1]:= ClearAll[ a, k, p0, pk, lambdaa, lambda, mu, len, lens, lenq, ws, w, wq, n, pn];
        rulek = k -> 26;
        rulen = n -> n;
        rule1 = rule1init -> a == 0.88;
        rule2 = rule2init -> lambda == 4.;
```

3.2.2. Example 2(lambda and mu are equal.)

```
In[1]:= ClearAll[ a, k, p0, pk, lambdaa, lambda, mu, len, lens, lenq, ws, w, wq, n, pn];
        rulek = k -> 26;
        rulen = n -> n;
        rule1 = rule1init -> mu == 4;
        rule2 = rule2init -> lambda == 4.;
```

3.2.3. Formulas

```
In[1]:=  mylist = Sort[Flatten[Simplify[Solve[{
              rule1init /. rule1,
              rule2init /. rule2,
              a == lambda/mu,
              p0 == 1 / ( k + 1 ) /. rulek,
              lambdaa == ( 1 - p0 ) lambda,
              len == k/2 /. rulek,
              lens == len - lenq,
              lenq == len - ( 1 - p0 ),
              ws == 1/mu,
              w == len / lambdaa,
              wq == lenq / lambdaa
              },
              {lambda,a,mu,ws,w,wq, len,lens,lenq, lambdaa, p0 },
              {lambda,a,mu,rho,ws,w,wq, len,lens,lenq, lambdaa, p0  }]]]];


         If[mylist[[2,2]] == mylist[[7,2]],
              {
                   Print[TableForm[mylist]];
                   pn = 1/(k + 1) /. rulek;
              },
              {
                   mylist = Sort[Flatten[Simplify[Solve[{
                         rule1init /. rule1,
                        rule2init /. rule2,
                         a == lambda / mu,
                         pk == ( 1-a ) a^k / ( 1- a^(k+1) ) /. rulek,
                        p0 == ( 1 - a ) / ( 1 - a^(k+1) ) /. rulek,
                          lambdaa == ( 1 - pk ) lambda /. rulek,
                         len == a ( 1 - ( k+1 ) a^k + k a^(k+1)) / ( ( 1 - a ) ( 1 - a^(k+1) ) ) /. rulek,
                         lens == len - lenq,
                         lenq == len - ( 1 - p0 ),
                        ws == 1/mu,
                        w == len / lambdaa,
                        wq == lenq / lambdaa
                        },
                        {lambdaa,mu,a,ws,w,wq, len,lens,lenq, lambda, p0, pk},
                        {lambdaa,mu,a,ws,w,wq, len,lens,lenq, lambda, p0, pk}]]]];
                        Print[TableForm[mylist]];
                        pn = (1 - mylist[[1,2]]) (mylist[[1,2]])^(n /. rulen)  /(1 - (mylist[[1,2]])^(k + 1)) /. rulek
;
              }
           ];
         Print["The probability of ", n /. rulen, " customer in the system is pn: ", pn];
```

3.2.4. M/M/1/K - Visualization

The function mm1klenplot generates surface plots of L and Lq versus lambda and mu ranging from 0 to lambda-max and 0 to mu-max for the M/M/1/K queuing system given a value of k. The function mm1kwaitplot is similar, however it computes the plots for W and Wq.

mm1klenplot [lambda-max, mu-max, k]

mm1kwaitplot [lambda-max, mu-max, k]

```
In[1]:=  mm1klenplot[ 20,20,5 ];
         mm1kwaitplot [ 20,20,5 ];
```

3.3. M/M/c

The following function returns the probability of n customer in the system where j is the number of customer, c is the number of server, a is the traffic intensity, and pp0 is the probability of 0 customer in the system.

In[1]:=
```
*)
mmcp[j_, c_, a_, pp0_] := Module[{},
     If[!NumberQ[j],
         Return[{pp0 a^j /j!, pp0 a^j/c!/c^(j-c)}],
          If[i <= c,
               Return[{pp0 a^j /j! }],
               Return[{pp0 a^j/c!/c^(j-c)}]
            ]
       ];
]

(*
```

To use the system, input the number of severs available under rulec and the number of customers under rulen for probability of the desired number of customers in the system. When a symbolic expression is given to n, the probability of n customers in the system is a list The first element is for n <= c and the second one is for n>=c. rule1 and rule2 are reserved for the given(known) values of the queuing system. At least two independent values are expected. For example, lambda and mu can be used to solve the queuing system.

In[2]:=
```
ClearAll[rulec,rulen,p0,pn,mylist,lambda,mu]
ClearAll[ a, c, lambdaa, lambda,mu,len,lens,lenq,ws,w,wq,n,c,i];
rulec = c -> 3;
rulen = n -> n;
rule1 = rule1init -> a == 1/2;
rule2 = rule2init -> mu == 20;
```

In[3]:=
```
mylist = Sort [ Flatten [ Simplify [ Solve [ {
     rule1init /. rule1,
     rule2init /. rule2,
    a == lambda/mu,
     lenq == a cca[c,a] / ( c - a ) /. rulec,
     len == lambda w,
     lens ==len - lenq,
     ws == 1/mu,
     wq == lenq / lambda,
     w == wq + ws
     },
     { a, lambda, mu, lenq, len, lens, w, ws, wq },
     { a, lambda, mu, lenq, len, lens, w, ws, wq } ] ] ] ];
Print[TableForm[mylist]];
p0 = Simplify[1 /(mylist[[1,2]]^c/c!/(1-mylist[[1,2]]/c)+Sum[(mylist[[1,2]]^i)/(i!),{i,0,c-1}]) /. rulec];
Trace[pn = Simplify[mmcp[n /. rulen, c /. rulec, mylist[[1,2]], p0]]];
Print["The probability of 0 customer in the system p0: ", p0];
Print["The probability of ", n /. rulen, " customer in the system pn: ", pn];
Print[""];
Print["Note: pn is a list. The first element is the probability of n customer in the node"];
Print["when n is less than or equal to the number of server and second element is the"];
Print["probability of n customer in the node when n is greater than or equal to the number"];
Print["of server."];
```

3.3.1. M/M/c - Visualization

The function mmclenplot generates plots of L and Lq versus the number of servers and rho ranging from 0 to servers-max and 0 to 1 for rho. The input value gives the maximum number of servers. Function mmcwaitplot generates a surface plot of W and Wq. Input is the same as mmclenplot. The function ccaplot generates a surface plot of Erlang's Delay Formula (C[c,a]), where a runs from 0 to 1 and c is the maximum number of servers input.

mmclenplt [servers-max]

mmcwaitplt [servers-max]

ccaplot [servers-max]

```
In[1]:=  mmclenplot[ 10 ];
         mmcwaitplot [ 10 ];
         ccaplot [ 5 ];
```

■ 3.4. M/M/c/c - M/M/c loss system

```
In[1]:=  *)
         ClearAll[p]
         mmccp[n_, c_, a_] := Module[{},Return[Simplify[((a^n)/n!)/Sum[ a^i / i!, {i,0,c}]]]]
         (*
```

To use the system, input the number of severs available under rulec and the number of server under rulen for the probability of the desired number of server that are busy. rule1 and rule2 are reserved for the given(known) values of the queuing system. At least two independent values are expected. For example, lambda and mu can be used to solve the queuing system.

```
In[2]:=  ClearAll[ a, c, lambdaa, lambda,mu,len,lens,lenq,ws,w,wq,n,c,mylist];

         rulec = c -> 2;
         rulen = n -> n;
         rule1 = rule1init -> lambda == 10;
         rule2 = rule2init -> mu == 20;


In[3]:=  mylist = Sort[Flatten[Simplify[Solve[{
                        rule1init /. rule1,
                        rule2init /. rule2,
                        a == lambda / mu,
                        lambdaa == lambda ( 1 - bca[ c, a ] ) /. rulec,
                        len == a ( 1 - bca[ c, a ] ) /. rulec,
                        lens == len,
                        lenq == 0,
                        wq == 0,
                        ws == w,
                        w == len / lambdaa
                          },
                    {lambdaa,mu,a,ws,w,wq, len,lens,lenq, lambda },
                    {lambdaa,mu,a,ws,w,wq, len,lens,lenq, lambda }]]]];
         Print[TableForm[mylist]];
         Print["The probability of ", n /. rulen, " customer in the system is ", mmccp[n /. rulen,c /. rulec ,mylist[[1,2]]
         Print["Note: n=0,...,",c /. rulec];
         Print["The probability of all servers are busy is ",bca[c /. rulec, mylist[[1,2]]]];
```

■ 3.4.1. M/M/c/c - Visualization

The function mmcclenplot generates a plot of the total number in the system versus the numbers of servers and rho ranging from 0 to servers-max and 0 to 1 for rho. The input is the maximum number of servers. Since there is no queuing allowed, by definition Wq=0 and Lq=0. Thus L=Ls and W=Ws=1\mu. The plot for W can be seen in the M/M/1 queuing section and is not presented here. The function bcaplot generates a plot of Erlang's Loss Formula B[c,a] for a=0 to 1 and c the maximum number of servers.

mmcclenplt [servers-max] : (a=0->1)

bcaplot [servers-max] : (a=0->1)

In[1]:=
```
mmcclenplot [ 10 ];
bcaplot [ 5 ];
```

3.5. M/M/Infinity

To use the system, input the number of customers under rulen for probability of the desired number of customers in the system. rule1 and rule2 are reserved for the given(known) values of the queuing system. At least two independent values are expected. For example, lambda and mu can be used to solve the queuing system.

In[1]:=
```
ClearAll[mylist,pn, n];
rulen = n -> n;
rule1 = rule1init -> lambda == 5;
rule2 = rule2init -> mu == 6;
```

In[2]:=
```
mylist=Sort[Flatten [Simplify [ Solve [{
     rule1init /. rule1,
     rule2init /. rule2,
     a == lambda / mu,
      len == a,
     lens == len,
     lenq == 0,
     ws == w,
     w == len /lambda,
     wq == 0
     },
     {a,lambda,mu,ws,w,wq, len,lens,lenq },
     {a,lambda,mu,ws,w,wq, len,lens,lenq } ]]]];
     Print[TableForm[mylist]];
pn = ((mylist[[1,2]])^n / n! /. rulen)Exp[-mylist[[1,2]]];
Print["The probability of ", n /. rulen, " customer in the system pn: ", pn];
```

3.5.1. M/M/Infinity

The plots for the M/M/Infinity system are not shown. Since by definition, Wq=0 and Lq=0, W=Ws and L=lambda/mu. The plot for W=Ws can be seen in the M/M/1 section.

3.6. M/M/1/K/K - Machine Repair with One Repairman

In[1]:=
```
*)
mm1kkp[n_, p0_, z_, k_] := Module[{}, Return[((k!)/((k-n)!)) z^(-n) p0]]
(*
```

To use the system, input the number of machine available under rulek and the number of machines that are out of service under rulen for probability of the desired number of the machines that are out of service. rule1 and rule2 are reserved for the given(known) values of the queuing system. At least two independent values are expected. For example, Mean Time To Failure and Ws can be used to solve the queuing system.

In[2]:=
```
ClearAll[ a, c, rulek, rulen, lambdaa, lambda,alpha,mu,len,lens,lenq,ws,w,wq, rho, MTTF, p0, z, mylist]
rulek = k -> 2;
rulen = n -> n;
rule1 = rule1init -> MTTF == 20;
rule2 = rule2init -> ws == 10;
```

```
In[3]:=
        mylist = Sort[Flatten[Simplify[Solve[{
                  rule1init /. rule1,
                  rule2init /. rule2,
                  MTTF == 1 / alpha,
                  p0 == bca [ k, z ] /. rulek,
                  z == MTTF / ws,
                  rho == 1 - p0,
                  lambda == rho / ws,
                  w == k / lambda - MTTF /. rulek,
                  ws == 1 / mu,
                  wq == w - ws,
                  len == lambda w,
                  lenq == lambda wq
                  },
                  { alpha, p0, z, MTTF, rho, lambda, mu, w, ws, wq, len, lenq },
                  { alpha, p0, z, MTTF, rho, lambda, mu, w, ws, wq, len, lenq }]]]];
        Print[TableForm[mylist]];
        Print["The probability of ", n /. rulen," machines are out of services is ",mm1kkp[n /. rulen, mylist[[7,2]],
mylist[[12,2]], k /. rulek]];
        Print["Note: n=0,...,",k /. rulek];
        Print["The probability that the system is available is ", 1 - mm1kkp[k /. rulek, mylist[[7,2]], mylist[[12,2]],
k /. rulek]];
```

3.6.1. M/M/1/K/K - Visualization

The function mm1kklenplot generates a surface plot for L versus lambda and number of customers k ranging from 0 to lambda-max and 0 to k-max at a mean time to failure MTTF. The function mm1kkwaitplot generates a surface plot of W and Wq of 0 to lambda-max, 1 to k-max for given mu and mean time to failure, (MTTF).

mm1kklenplot [lambda-max, k-max, MTTF] - Note L is independent of mu

mm1kkwaitplot [lambda-max, mu, k-max, MTTF]

```
In[1]:= mm1kklenplot [ 10,5,0.7];
        mm1kkwaitplot [ 5,10,6,0.88 ];
```

4 Jackson Networks

```
In[1]:=  *)
         <<digraph.m
         (*
```

4.1. Input Network Information

The following two examples illustrate how to implement various types of Jackson networks. Input for the notebook consists of four parts. The first one is the network connectivity matrix. Element on the i-th row and j-th column in the matrix represents the probability of the output from node i to node j. This matrix informs mathematica as to how the network nodes are connected. The second one is a vector which represents the external rates into the network. The third one itemizes the server departure rate for each node in the network. Finally, the fourth one lists how many servers are at each node in the network.

Note if |matrix| = n x n, then |vector| = n.

4.1.1. Example 1 -

This is an example of a Jackson network with numerical parameters. When the notebook is first opened this cell is initialized.

```
In[1]:=  *)
         networkconnectivity = { { 0, 0.25, 0, .75},
                                   { 0, 0, 1, 0},
                                   { 0.15, 0.45, 0, 0.40},
                                 { 0, 0, 0, 0} };

         lwlambda = {20,30,0,0};
         mu = {40,100,150,55};
         numberofserver = {1,1,1,1};
         (*
```

4.1.2. Example 2 -

```
In[1]:=  networkconnectivity = { { 0, 1/4 },
                                   { 1/5, 0 } };
         lwlambda = {90,25};
         mu = {150,100};
         numberofserver = {1,1};
```

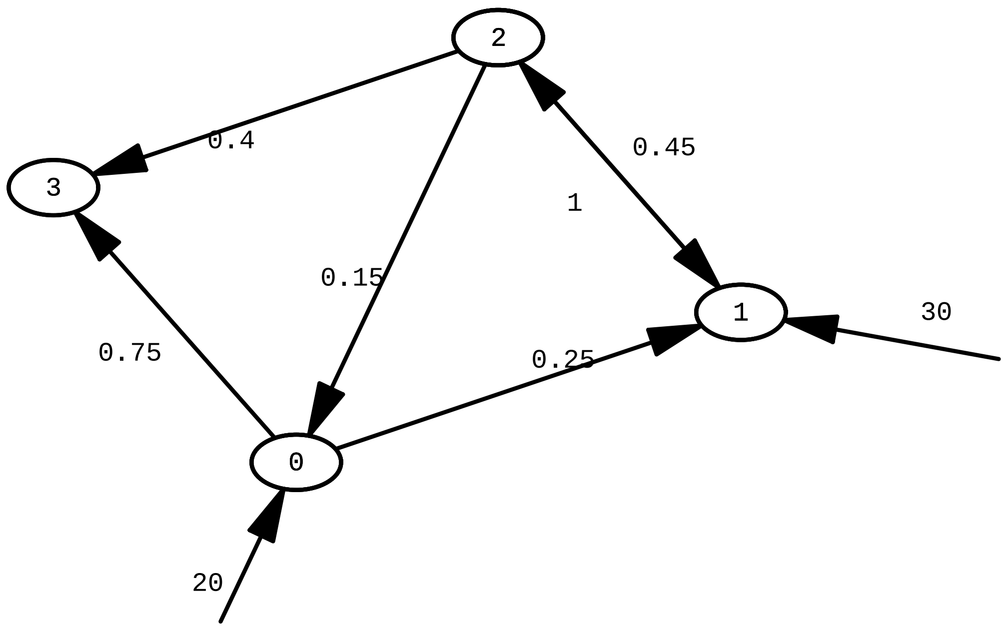
4.1.3. User Input Cell

```
In[1]:=
         networkconnectivity = { };
         lwlambda = { };
         mu = { };
         numberofserver = { };
```

4.1.4. Graphical Representation of Jackson Network

The network which has been input into the system can be viewed here. This is provided as an aid so that one can see if the system which was input matches what was expected. The Mathematica code to implement computenetwork is contained in the package digraph.m. The Mathematica notebook interface seems to contain bugs in its graphical display functions. The computegraph function might require multiple execution to generate the correct graphical output.

```
In[1]:=  *)
         computenetwork [ networkconnectivity, lwlambda ]
         (*
```




```
Out[1]=  -Graphics-
```

4.2. Jackson Network Solution

The function networksolve solves a Jackson network. The input parameters are the network connectivity matrix, the external input lambda vector, the server utilization vector, and the number of servers per node vector. Networksolve outputs the following information:

1) Probability of a customer leaving at node x,

2) The effectival arrival rates at node x,

3) The probability of one customer in the system,

4) A list of queue information for each node x.

The following cell contains the Mathematica code to implement networksolve. Initially this cell is hidden when the notebook is opened.

In[1]:=
```
*)

p[i_, c_, a_, pp0_] :=

     Module[{},

          If[Not[NumberQ[i]],
              Return[{pp0 a^i /i!, pp0 a^i/c!/c^(i-c)}],
               If[i <= c,
                    Return[{pp0 a^i /i! }],
                    Return[{pp0 a^i/c!/c^(i-c)}]
            ]
       ]
]
(*
```

In[2]:=
```
*)
calceffectivearrivalrate[ netcon_, lwlambda_]:=

     Module[ { pmatrix, i, b, uplambda },

          pmatrix = netcon;

          Do [ pmatrix [[ i, i ]] = pmatrix [[ i, i ]] - 1,
               { i, 1, Length[pmatrix] }
          ];

          pmatrix = Transpose [ pmatrix ];
          b = lwlambda;

          Do [ b[[i]] = - b[[i]],
               { i, 1, Length[b] }
          ];

          uplambda = LinearSolve[pmatrix, b];
          Return[uplambda]
     ]

(*
```

In[3]:=
```
*)

cca[ c_, aaa_ ] :=

     Module [ {},

          Return [ 1/((c!(1-(aaa/c))/aaa^c)
                     ( Sum [ aaa^i/i! , {i,0,c-1} ] ) + 1)
          ]
     ]
(*
```

```
In[4]:=  *)
         solvemmc[inode_, lambda_, muu_, numofservers_, n_ ]:=

              Module[{mylist, tmpstr},

                   mylist = Simplify [ Solve [ {
                        a == lambda / muu,
                        lenq == a cca[numofservers, a] / ( numofservers - a ),
                        ws == 1 / muu,
                        wq == lenq / lambda,
                        w == wq + ws,
                        len == lambda w,
                        lens == len - lenq },
                        { a, lenq, len, lens, w, ws, wq },
                        { a, lenq, len, lens, w, ws, wq }]
                   ];
                   mylist = Sort[ Flatten[ mylist ]];

                   p0 = Simplify[1 / (mylist[[1,2]]^numofservers/numofservers!/(1-mylist[[1,2]]/numofservers)+
                                     Sum[(mylist[[1,2]]^i)/(i!),{i,0,numofservers-1}])];
                   pn = Simplify[p[n, numofservers, mylist[[1,2]], p0]];

                   tmpstr = StringJoin["Node ", ToString[inode]];
                   mylist = Prepend[mylist, tmpstr];
                   mylist = Append [ mylist, "p0 - >"];
                   mylist = Append [ mylist, p0 ];
                   (*tmpstr = StringJoin["p0 -> ", ToString[p0]];
                   mylist = Append [ mylist, tmpstr];*)
                   mylist = Append [ mylist, "pn - >"];
                   mylist = Append [ mylist, pn ];

                   Return[mylist];
              ]
         (*
```

In[5]:=
```
*)
networksolve [ netcon_, lwlambda_, mulist_, numbservlist_, n_ ] :=

     Module [ {uplambda, size, nodelist={}, invalidflag, poneinsystem,i,j,tmp},

          size = Length [ netcon ];
        invalidflag = 0;

          uplambda = calceffectivearrivalrate[netcon, lwlambda];

          Do [
               If [ uplambda [[i]] > mulist[[i]] numbservlist[[i]],
                    invalidflag = 1,
                    {}
               ],
               { i, 1, Length[ uplambda ] }
         ];

         If [ invalidflag == 0,
         {
               q = Table [ 0, { i, 1, size } ];

               Do[  q[[i]] = 1 - Sum[netcon[[i,k]], {k,1,size}],
                    {i,1,size}
               ];

               Do[ nodelist = Append [nodelist,
                                        solvemmc[i, uplambda[[i]], mulist[[i]], numbservlist[[i]], n ] ],
                   {i, 1, size}
               ];

               poneinsystem = 0;
               Do[
               {
                    tmp = 1;
                    Do[{
                         If[i==j, tmp = tmp * nodelist[[j,12,1]] /. n -> 1, tmp = tmp * nodelist[[j,10]] ];
                    },
                    {j,1,size}];
                    poneinsystem = poneinsystem + tmp;
               },
               {i,1,size}
               ];

               Print["The probability of a customer leaving the network is: ", q];
               Print["The effective arrival rates are: ", Simplify[uplambda]];
               Print["The probability of one customer in the network is: ", poneinsystem];
               Print[TableForm[nodelist] ];
               Print[""];
               Print["Note: pn is a list. The first element is the probability of n customer in the node"];
               Print["when n is less than or equal to the number of server and second element is the"];
               Print["probability of n customer in the node when n is greater than or equal to the number"];
               Print["of server."];
          },
          {
               Print["This network does not have a steady state solution because"];
               Print["one or more effective arrival rates are greater than a node´s"];
               Print["number of servers times its departure rate."]
          }
          ]
          Return;
]
(*
```

In[6]:=
```
*)
networksolve [ networkconnectivity, lwlambda, mu, numberofserver, n]
(*
```