qnetworks

A Queueing Networks Analysis Package for GNU Octave
User's Guide, Edition 1 for release 0.8.4
2010-06-23

Moreno Marzolla

Copyright © 2008, 2009, 2010 Moreno Marzolla (marzolla@cs.unibo.it).

This is the first edition of the qnetworks documentation, and is consistent with version 0.8.4 of the package.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.

Portions of this document have been adapted from the octave manual, Copyright © John W. Eaton.

Table of Contents

1 Preface

This document describes the `qnetworks` package; `qnetworks` is a collection of functions written in GNU Octave which implement numerical algorithms for analyzing Queueing Network models. In particular, the `qnetworks` package contains functions for analyzing Jackson networks, open, closed or mixed product-form BCMP networks, and computation of performance bounds. More specifically, the following algorithms have been implemented

- Convolution for closed, single-class product-form networks with load-dependent service centers;
- Mean Value Analysis (MVA) algorithms for single as well as multiple-class closed product-form networks.
- MVA for mixed, multiple class product-form networks with load-independent service centers;
- Closed, single-class networks with blocking using the approximate MVABLO algorithm by F. Akyildiz;
- Closed, multiple class product-form networks using approximate MVA (Bard-Schweitzer approximation);
- Computation of Asymptotic Bounds, Balanced System Bounds as well as Geometric Bounds;

Furthermore, `qnetworks` provides functions for analyzing the following kind of single-station queueing systems:

- $M/M/1$
- $M/M/m$
- $M/M/\infty$
- $M/M/1/k$ single-server, finite capacity system
- $M/M/m/k$ multiple-server, finite capacity system
- Asymmetric $M/M/m$
- $M/G/1$ (general service time distribution)
- $M/H_m/1$ (Hyperexponential service time distribution)

Finally, functions for Markov Chain analysis are provided as well:

- Birth-death process;
- Computation of mean time to absorption;
- Computation of time-averages sojourn time.

`qnetworks` is distributed under the terms of the GNU General Public License (GPL), version 3 or later (see Appendix C [Copying], page 53) You are encouraged to share this software with others, and make this package more useful by contributing additional functions and reporting problems. See Appendix A [Contributing Guidelines], page 49.

You can cite `qnetworks` in a technical paper as:

> Moreno Marzolla, *The qnetworks Toolbox: A Software Package for Queueing Networks Analysis*. Khalid Al-Begain, Dieter Fiems and William J. Knottenbelt, Editors, Proceedings Analytical and Stochastic Modeling Techniques and

Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, June 14-16, 2010, volume 6148 of Lecture Notes in Computer Science, Springer pp. 102-116, ISBN 978-3-642-13567-5

An early draft of the above paper is also available as Technical Report UBLCS-2010-04, Department of Computer Science, University of Bologna, Italy, February 2010.

If you use BibTeX, this is the citation block:

```
@inproceedings{qnetworks,
  author    = {Moreno Marzolla},
  title     = {The qnetworks Toolbox: A Software Package for Queueing
               Networks Analysis},
  booktitle = {Analytical and Stochastic Modeling Techniques and
               Applications, 17th International Conference,
               ASMTA 2010, Cardiff, UK, June 14-16, 2010. Proceedings},
  editor    = {Khalid Al-Begain and
               Dieter Fiems and
               William J. Knottenbelt},
  year      = {2010},
  publisher = {Springer},
  series    = {Lecture Notes in Computer Science},
  volume    = {6148},
  pages     = {102--116},
  ee        = {http://dx.doi.org/10.1007/978-3-642-13568-2_8},
  isbn      = {978-3-642-13567-5}
}
```

2 Installing the qnetworks package

The most recent version of the `qnetworks` package can be downloaded from

http://www.moreno.marzolla.name/software/qnetworks/

Currently, the most recent version is 0.8.4, and is distributed in source form in the '`qnetworks-0.8.4.tar.gz`' archive. Download the source tarball from the URL above, unpack it somewhere:

```
tar xfz qnetworks-0.8.4.tar.gz
cd qnetworks-0.8.4/
```

The `qnetworks` source distribution includes the following subdirectories:

'`doc/`' Documentation source code

'`inst/`' This directory contains the actual `m`-files which implement the various Queueing Network algorithms. Note that, as a notational convention, most of the source files (and the functions defined therein) start with the '`qn`' prefix.

'`test/`' This directory contains the automated test function, which is the one used by GNU Octave.

'`scripts/`'
This directory contains some utility scripts taken from GNU Octave, which extract the documentation from the specially-formatted comments in the `m`-files.

'`examples/`'
This directory contains examples which are automatically extracted from the '`demo`' blocks in the source code. Some of these examples are included into the documentation.

'`broken/`' This directory contains scripts which are not working properly, or need additional testing before they can be included in the '`inst/`' directory.

The `qnetworks` package ships with a Makefile which can be used to automatically produce the documentation (in PDF and HTML format), and automatically run all function tests. The following targets are defined:

`all` Running '`make`' (or '`make all`') on the top-level `qnetworks` directory will build the necessary scripts used to extract the documentation from the comments embedded in the `m`-files. Then, it will produce the '`doc/qnetworks.pdf`' and '`doc/qnetworks.html`' files which contain the documentation in PDF and HTML format, respectively.

`check` Running '`make check`' will execute all the test functions contained in the `m`-files. If you modify the code of any function from the '`inst/`' directory, please run the test to ensure that no errors are introduced. Furthermore, you are encouraged to contribute new test functions to `qnetworks`, to help verifying that the algorithms are implemented correctly.

`clean`
`distclean`
`dist` The '`make clean`', '`make distclean`' and '`make dist`' commands are used to clean up the source directory and prepare the distribution archive in compressed tar format.

If you want to define a new function, simply create the corresponding `m`-file in the '`inst/`' directory. Each file should have the exact name of the function it defines. You should also add test blocks to ensure that your function computes the correct value; test blocks are extremely useful to help spot errors when a function is modified.

To install the `qnetworks` package, you can copy all `.m` files from the '`inst/`' directory to a location where Octave can find them. You can also start Octave with the '`-p`' option to tell it to add a path to its search path, so that it will find the functions automatically without the need to move them:

```
octave -p /path/to/qnetworks
```

For example, if all `qnetworks` m-files are installed in '`/usr/local/qnetworks`', then you can start Octave as follows:

```
octave -p '/usr/local/qnetworks'
```

Alternatively, you can use the '`pkg install`' Octave command to install the tarball directly. Start GNU Octave, and type the following command at the prompt:

```
octave:1> pkg install qnetworks-0.8.4.tar.gz
```

In this way, the `qnetworks` package can be uninstalled at any time with the '`pkg uninstall qnetworks`' Octave command.

You should now be able to use all `qnetworks` functions by simply invoking their name. You can also get on-line help about each function using the `help` Octave command. For example:

```
octave:2> help qnmvablo
```

will print the documentation for the `qnmvablo` function. Additional information can be found in the `qnetworks` manual, which is available in PDF format in '`doc/qnetworks.pdf`' and in HTML format in '`doc/qnetworks.html`'.

Within GNU Octave, you can also run the tests and demo blocks associated to the various functions, using the `test` and `demo` commands. For example, to run all the tests blocks of the `qnmvablo` function, use the following:

```
octave:3> test qnmvablo
 ⊣ PASSES 4 out of 4 tests
```

To execute the demo blocks of the `qnclosed` function, use the following:

```
octave:4> demo qnclosed
```

3 Getting Started

In this chapter we illustrate some usage examples of the `qnetworks` package. Additional examples are embedded in some of the function files; to display and execute the demos associated with function *fname* you can type `demo("fname")` at the Octave prompt. For example

```
demo("qnclosed");
```

executes all demos (if any) for the `qnclosed` function.

3.1 Analysis of Closed Networks

Let us consider a simple closed network with $K = 3$ service centers. Each center is of type $M/M/1$–FCFS. We denote with S_i the average service time at center i, $i = 1, 2, 3$. Let $S_1 = 1.0$, $S_2 = 2.0$ and $S_3 = 0.8$. The routing probability matrix is defined as follows:

$$P = \begin{pmatrix} 0 & 0.3 & 0.7 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

This network can be analyzed with the `qnclosed` function; in the case of single-class closed networks, as in the example above, the `qnclosed` function is a front-end to `qnclosedsinglemva` which implements the Mean Value Analysys (MVA) algorithm for single-class Queueing Networks.

`qnclosed` requires the following parameters:

N Number of requests in the network

S Array of average service times at the centers: *S(k)* is the average service time at center k.

V Array of visit ratios: *V(k)* is the average number of visits to center k.

The visit count V_k to center k can be computed with the `qnvisits` function:

```
P = [0 0.3 0.7; 1 0 0; 1 0 0];
V = qnvisits(P)
   ⇒ V = 1.00000 0.30000 0.70000
```

Note that the visit counts V_k satisfy the following relation:

$$V_j = \sum_{i=1}^{K} V_i P_{ij}$$

We can check that the computed values satisfy the above equation by evaluating the following expression:

```
V*P
     ⇒ ans = 1.00000 0.30000 0.70000
```

Now, we can analyze the network for a given population size N (for example, $N = 10$) as follows:

```
N = 10;
S = [1 2 0.8];
V = [1 0.3 0.7];
[U R Q X] = qnclosed( N, S, V )
   ⇒ U = 0.99139 0.59483 0.55518
   ⇒ R = 7.4360  4.7531  1.7500
   ⇒ Q = 7.3719  1.4136  1.2144
   ⇒ X = 0.99139 0.29742 0.69397
```

The output of the `qnclosed` function includes the center k utilization U_k, response time R_k, average number of customers Q_k and throughput X_k. In this particular case, for example, the throughput of center 1 is $X_1 = 0.99139$, and the average number of requests in center 3 is $Q_3 = 1.2144$. The utilization of center 1 is $U_1 = 0.99139$: note that center 1 has the higher utilization among the service centers, so it is the *bottleneck device*.

You can also use the `qnsolve` function to analyze this network. `qnsolve` can be applied to open, closed or mixed networks, and allows the network to be described in a very flexible way.

First, we let *Q1*, *Q2* and *Q3* to be the variables describing the service centers. Each variable is instantiated with the `qnmknode` function.

```
Q1 = qnmknode( "m/m/m-fcfs", 1 );
Q2 = qnmknode( "m/m/m-fcfs", 2 );
Q3 = qnmknode( "m/m/m-fcfs", 0.8 );
```

The first parameter of `qnmknode` is a string describing the type of the node. Here we use `"m/m/m-fcfs"` to denote a $M/M/m$–FCFS center. The second parameter gives the average service time. An optional third parameter can be used to specify the number m of service centers. If omitted, it is assumed $m = 1$ (single-server node).

Now, the network can be analyzed as follows:

```
N = 10;
V = [1 0.3 0.7];
[U R Q X] = qnsolve( "closed", N, { Q1, Q2, Q3 }, V )
   ⇒ U = 0.99139 0.59483 0.55518
   ⇒ R = 7.4360  4.7531  1.7500
   ⇒ Q = 7.3719  1.4136  1.2144
   ⇒ X = 0.99139 0.29742 0.69397
```

Of course, we get exactly the same results. Other functions can be used for closed networks, see Section 6.3 [Algorithms for Product-Form QN], page 28.

3.2 Analysis of Open Networks

Open networks can be analyzed in a similar way. Let us consider an open network with $K = 3$ service centers, and routing probability matrix as follows:

$$P = \begin{pmatrix} 0 & 0.3 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Note that in this network, requests can leave the system with probability $(1-0.3-0.5 = 0.2$ after completing service at center 1. Let us consider external arrivals to center 1 with rate $\lambda_1 = 0.15$; arrival rates to center 2 and 3 are zero.

Similarly with closed networks, we first need to compute the visit counts V_k to center k. This can be done with the `qnvisits` function as follows:

```
P = [0 0.3 0.5; 1 0 0; 1 0 0];
lambda = [0.15 0 0];
V = qnvisits(P, lambda)
   ⇒ V = 5.00000 1.50000 2.50000
```

The visit counts V_k open networks satisfy the following equation:

$$V_j = P_{0j} + \sum_{i=1}^{K} V_i P_{ij}$$

where P_{0j} is the probability of an external arrival to center j. This can be computed as:

$$P_{0j} = \frac{\lambda_j}{\sum_{i=1}^{K} \lambda_i}$$

Assuming the same service times as in the previous example, the network can be analyzed with the `qnopen` function, as follows:

```
S = [1 2 0.8];
[U R Q X] = qnopen( sum(lambda), S, V )
   ⇒ U = 0.75000 0.45000 0.30000
   ⇒ R = 4.0000  3.6364  1.1429
   ⇒ Q = 3.00000 0.81818 0.42857
   ⇒ X = 0.75000 0.22500 0.37500
```

Note that the first parameter of the `qnopen` function is the (scalar) aggregate arrival rate.

Again, it is possible to use the `qnsolve` high-level function:

```
Q1 = qnmknode( "m/m/m-fcfs", 1 );
Q2 = qnmknode( "m/m/m-fcfs", 2 );
Q3 = qnmknode( "m/m/m-fcfs", 0.8 );
lambda = [0.15 0 0];
[U R Q X] = qnsolve( "open", sum(lambda), { Q1, Q2, Q3 }, V )
   ⇒ U = 0.75000 0.45000 0.30000
   ⇒ R = 4.0000  3.6364  1.1429
   ⇒ Q = 3.00000 0.81818 0.42857
   ⇒ X = 0.75000 0.22500 0.37500
```

3.3 Markov Chains Analysis

(TBD)

4 Markov Chains

4.1 Discrete-Time Markov Chains

p = `dtmc_solve` (*P*) [Function File]

Compute the steady-state probability vector *p(1), p(2), ... p(N)* for a Discrete-Time Markov Chain with $N \times N$ transition probability matrix P.

INPUTS

P — *P(i,j)* is the transition probability from state i to state j. P must be an irreducible stochastic matrix, which means that the sum of each row must be 1, and the rank of P must be equal to its dimension.

OUTPUTS

p — *p(i)* is the steady-state probability that the system is in state i. p satisfies the equations $p = p\mathbf{P}$ and $\sum_i p_i = 1$

4.2 Continuous-Time Markov Chains

q = `ctmc_solve` (*Q*) [Function File]

Compute the steady-state probability vector *q = [q(1), q(2), ... q(N)]* for a Continuous-Time Markov Chain with $N \times N$ infinitesimal generator matrix Q. The steady state probability vector q satisfies the equation $q\mathbf{Q} = 0$ and $\sum_i q_i = 1$.

If the i-th column of matrix Q is zero, then we will set *q(i) = 0*, for any i.

INPUTS

Q — Infinitesimal generator matrix. *Q(i,j)* is the transition rate from state i to state j, for $i \neq j$. The matrix Q must also satisfy the condition `sum(Q,2) == 0`

OUTPUTS

q — *q(i)* is the steady-state probability that the system is in state i.

EXAMPLE

Consider a two-state CTMC such that transition rates between states are equal to 1. This can be solved as follows:

```
Q = [ -1  1; \
       1 -1  ];
q = ctmc_solve(Q)
   ⇒ q = 0.50000   0.50000
```

4.2.1 Birth-Death process

p = `ctmc_bd_solve` (*birth*, *death*) [Function File]

Compute the steady-state solution of a birth-death process with state space $(1, \ldots N)$.

INPUTS

birth — *birth* is a vector with $N-1$ elements, where `birth(i)` denotes the transition rate from state i to state $i+1$.

death — *death* is a vector with $N-1$ elements, where `death(i)` denotes the transition rate from state $i+1$ to state i

OUTPUTS

p — `p(i)` is the steady-state probability that the system is in state i. p is a vector with N elements.

4.2.2 Expected Sojourn Time

Given a N state continuous-time Markov Chain with infinitesimal generator matrix $\mathbf{Q}$, we define the vector $\mathbf{L}(t) = (L_1(t), L_2(t), \ldots L_N(t))$ such that $L_i(t)$ is the expected sojourn time in state i during the interval $[0, t)$, assuming that the initial occupancy probability at time 0 was $\pi(0)$. Then, $\mathbf{L}(t)$ is the solution of the following differential equation:

$$\frac{d\mathbf{L}(t)}{dt} = \mathbf{L}(t)\mathbf{Q} + \pi(0), \qquad \mathbf{L}(0) = \mathbf{0}$$

The function `ctmc_exps` can be used to compute $\mathbf{L}(t)$, by using the `lsode` Octave function to solve the above linear differential equation.

L = `ctmc_exps` (*Q*, *tt*, *p*) [Function File]

Compute the *expected total time* `L(t,j)` spent in state j during the time interval `[0,tt(t))`, assuming that at time 0 the state occupancy probability was *p*.

INPUTS

Q — Infinitesimal generator matrix. `Q(i,j)` is the transition rate from state i to state j, for $i \neq j$, $1 \leq i, j \leq N$. The matrix Q must also satisfy the condition `sum(Q,2) == 0`

tt — This parameter is a vector used for numerical integration. The first element `tt(1)` must be 0, and the last element `tt(end)` must be the upper limit of the interval $[0, t)$ of interest (`tt(end) ==` t).

p — `p(i)` is the probability that at time 0 the system was in state i, for all $i = 1, 2, \ldots N$

OUTPUTS

L — `L(t,j)` is the expected time spent in state j during the interval `[0,tt(t))`. $1 \leq t \leq$ `length(tt)`

EXAMPLE

Let us consider a pure-birth, 4-states CTMC such that the transition rate from state i to state $i+1$ is $\lambda_i = i\lambda$ ($i = 1, 2, 3$), with $\lambda = 0.5$. The following code computes the expected sojourn time in state i, $i = 1, 2, 3$ given initial occupancy probability $p_0 = (1, 0, 0, 0)$.

```
lambda = 0.5;
N = 4;
birth = lambda*linspace(N-1,1,N-1);
death = zeros(1,N-1);
Q = diag(birth,1)+diag(death,-1);
Q -= diag(sum(Q,2));
tt = linspace(0,10,100);
p0 = zeros(1,N); p0(1)=1;
L = ctmc_exps(Q,tt,p0);
plot( tt, L(:,1), ";State 1;", "linewidth", 2, \
      tt, L(:,2), ";State 2;", "linewidth", 2, \
      tt, L(:,3), ";State 3;", "linewidth", 2, \
      tt, L(:,4), ";State 4 (absorbing);", "linewidth", 2);
legend("location","northwest");
xlabel("Time");
ylabel("Expected sojourn time");
```

4.2.3 Time-Averaged Expected Sojourn Time

M = `ctmc_taexps` (*Q*, *tt*, *p*) [Function File]

Compute the *time-averaged sojourn time* `M(t,j)`, defined as the fraction of the time interval `[0,tt(t))` spent in state j, assuming that at time 0 the state occupancy probability was p.

INPUTS

Q Infinitesimal generator matrix. `Q(i,j)` is the transition rate from state i to state j, for $i \neq j$, $1 \leq i, j \leq N$. The matrix Q must also satisfy the condition `sum(Q,2) == 0`

tt This parameter is a vector used for numerical integration of the sujourn time. The first element `tt(1)` must be 0, and the last element `tt(end)` must be the upper limit of the interval $[0, t)$ of interest (`tt(end) ==` t). This vector is used by the ODE solver to compute the solution M.

p `p(i)` is the probability that, at time 0, the system was in state i, for all $i = 1, 2, \ldots N$

OUTPUTS

M `M(t,j)` is the expected fraction of time spent in state j during the interval $[0, tt(t))$ assuming that the state occupancy probability at time zero was p. `1` $\leq$ `t` $\leq$ `length(tt)`

EXAMPLE

```
lambda = 0.5;
N = 4;
birth = lambda*linspace(N-1,1,N-1);
death = zeros(1,N-1);
Q = diag(birth,1)+diag(death,-1);
Q -= diag(sum(Q,2));
t = linspace(0.001,10,100);
p = zeros(1,N); p(1)=1;
M = ctmc_taexps(Q,t,p);
plot(t, M(:,1), ";State 1;", "linewidth", 2, \
     t, M(:,2), ";State 2;", "linewidth", 2, \
     t, M(:,3), ";State 3;", "linewidth", 2);
legend("location","northeast");
xlabel("Time");
ylabel("Time-averaged Expected sojourn time");
```

4.2.4 Expected Time to Absorption

If we have a Markov Chain with absorbing states, it is possible to define the *expected time to absorption* as the expected time until the system goes into an absorbing state. More specifically, let us suppose that A is the set of transient (i.e., non-absorbing) states the the CTMC with N states and infinitesimal generator matrix $\mathbf{Q}$. Then, we define $\mathbf{L}_A(\infty)$ as the expected time to absorption as the solution of the following equation:

$$\mathbf{L}_A(\infty)\mathbf{Q}_A = -\pi_A(0)$$

where $\mathbf{Q}_A$ is the restriction of matrix $\mathbf{Q}$ to only states in A, and $\pi_A(0)$ is the initial state occupancy probability at time 0, also restricted only to states in A.

t = `ctmc_mtta` (*Q*, *p*) [Function File]

Compute the Mean-Time to Absorption (MTTA) starting from initial occupancy probability p at time 0. If there are no absorbing states, this function fails with an error.

INPUTS

Q $N \times N$ infinitesimal generator matrix. *Q(i,j)* is the transition rate from state i to state j, $i \neq j$. The matrix Q must satisfy the condition `sum(Q,2) == 0`

p *p(i)* is the probability that the system is in state i at time 0, for each $i = 1, 2, \ldots N$

OUTPUTS

t Mean time to absorption of the process represented by matrix Q. If there are no absorbing states, this function fails.

EXAMPLE

Let us consider a simple model of a redundant disk array. We assume that the array is made of 5 independent disks, such that the array can tolerate up to 2 disk failures. If three or more disks break, the array is dead and unrecoverable. We want to estimate the Mean-Time-To-Failure (MTTF) of the disk array.

We model this system as a 4 states Markov chain with state space $\{2, 3, 4, 5\}$ such that state i denotes that exactly i disks are active. State 2 is absorbing. Let λ denotes the failure rate of a single disk. The system starts in state 5 (all disks are operational). This is a pure death process, where the death rate from state i to state $i - 1$ (for all $i > 2$) is λi.

It turns out that the MTTF in this case is the MTTA. We can compute the MTTF (MTTA) as follows:

```
lambda = 0.01;
death = [ 3*lambda 4*lambda 5*lambda ];
Q = diag(death,-1);
Q -= diag(sum(Q,2));
t = ctmc_mtta(Q,[0 0 0 1])
```

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998.

5 Single Station Queueing Systems

Single Station Queueing Systems, as the name suggests, are special queueing systems which are made by a single station. Thus, such systems are quite easy to analyze. The `qnetworks` package contains functions for handling the following kind of systems:

- $M/M/1$ single-server queueing station;
- $M/M/m$ multiple-server queueing station;
- $M/M/\infty$ infinite-server station (delay center);
- $M/M/1/K$ single-server, finite-capacity queueing station;
- $M/M/m/K$ multiple-server, finite-capacity queueing station;
- $M/G/1$ system;
- $M/H_m/1$ system.

Note that these functions can be used as building blocks for analyzing Queueing Networks. For example, Jackson networks can be analyzed by computing the aggregate arrival rates to each node, and then solving each node in isolation as if it were a single station queueing system.

5.1 The $M/M/1$ System

The $M/M/1$ system is made of a single server connected to an unlimited FCFS queue. The mean arrival rate is Poisson with arrival rate λ, while the service time is exponentially distributed with average service rate μ. The system is stable if $\lambda < \mu$.

`[U, R, Q, X, p0] = qnmm1 (lambda, mu)` [Function File]

Compute utilization, response time, average number of requests and throughput for a $M/M/1$ queue.

The steady-state probability π_k that there are k jobs in the system, $k \geq 0$, can be computed as:

$$\pi_k = (1 - \rho)\rho^k$$

where $\rho = \lambda/\mu$ is the server utilization.

INPUTS

lambda Arrival rate (jobs/s, `lambda > 0`).

mu Service rate (jobs/s, `mu > lambda`).

OUTPUTS

U Server utilization

R Service center response time

Q Average number of requests in the system

X Service center throughput. If the system is ergodic, we will always have `X = lambda`

p0 Steady-state probability that there are no requests in the system.

lambda and *mu* can be vectors of the same size. In this case, the results will be vectors as well.

See also: qnmmm, qnmminf, qnmmmk.

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, Section 6.3.

5.2 The $M/M/m$ System

The $M/M/m$ system is similar to the $M/M/1$ system, except that there are $m \geq 1$ servers connected to a single queue. Thus, at most m requests can be in service at the same time. The $M/M/m$ system can be seen as a single server with load-dependent service time $\mu(n)$, which is a function of the number n of nodes in the center:

```
mu(n) = min(m,n)*mu
```

`[U, R, Q, X, p0, pm] = qnmmm (lambda, mu)` [Function File]
`[U, R, Q, X, p0, pm] = qnmmm (lambda, mu, m)` [Function File]

Compute utilization, response time, average number of requests in service and throughput for a $M/M/m$ queue, a queueing system with m service centers connected to a single queue.

The steady-state probability π_k that there are k jobs in the system, $k \geq 0$, can be computed as:

$$\pi_k = \begin{cases} \pi_0 \dfrac{(m\rho)^k}{k!} & 0 \leq k \leq m; \\ \pi_0 \dfrac{\rho^k m^m}{m!} & k > m. \end{cases}$$

where $\rho = \lambda/(m\mu)$ is the individual server utilization. The steady-state probability π_0 that there are no jobs in the system can be computed as:

$$\pi_0 = \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho} \right]^{-1}$$

INPUTS

lambda Arrival rate (jobs/s, `lambda>0`).

mu Service rate (jobs/s, `mu>lambda`).

m Number of servers (`m>0`). If omitted, it is assumed `m==1`.

OUTPUTS

U Service center utilization, $U = \lambda/(m\mu)$.

R Service center response time

Q Average number of requests in the system

X Service center throughput. If the system is ergodic, we will always have `X = lambda`

p0 Steady-state probability that there are 0 requests in the system

pm Steady-state probability that an arriving request has to wait in the queue

lambda, *mu* and *m* can be vectors of the same size. In this case, the results will be vectors as well.

See also: qnmm1,qnmminf,qnmmmk.

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, Section 6.5.

5.3 The $M/M/$inf System

The $M/M/\infty$ system is similar to the $M/M/m$ system, except that there are infinitely many servers (that is, $m = \infty$). Thus, there is no queueing at this node, as each arriving job always finds a free server. The $M/M/\infty$ system is always stable, regardless what the arrival and service rates λ and μ are.

`[U, R, Q, X, p0] = qnmminf (lambda, mu)` [Function File]

Compute utilization, response time, average number of requests and throughput for a $M/M/\infty$ queue. This is a system with an infinite number of servers. Note that a $M/M/\infty$ system is always stable, regardless the values of the arrival and service rates.

The steady-state probability π_k that there are k requests in the system, $k \geq 0$, can be computed as:

$$\pi_k = \frac{1}{k!} \left(\frac{\lambda}{\mu} \right)^k e^{-\lambda/\mu}$$

INPUTS

lambda Arrival rate (jobs/s, `lambda>0`).

mu Service rate (jobs/s, `mu>0`).

OUTPUTS

U Traffic intensity (defined as λ/μ). Note that this is different from the utilization, which in the case of $M/M/\infty$ centers is always zero.

R Service center response time.

Q Average number of requests in the system (which is equal to the traffic intensity λ/μ).

X Throughput (which is always equal to `X = lambda`).

p0 Steady-state probability that there are no requests in the system *lambda* and *mu* can be vectors of the same size. In this case, the results will be vectors as well.

See also: qnmm1,qnmmm,qnmmmk.

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, Section 6.4.

5.4 The $M/M/1/K$ System

In a $M/M/1/K$ finite capacity system there can be at most k jobs at any time. If a new request tries to join the system when there are already K other requests, the arriving request is lost. The queue has $K-1$ slots. The $M/M/1/K$ system is always stable, regardless of the arrival and service rates λ and μ.

`[U, R, Q, X, p0, pK] = qnmm1k (lambda, mu, K)` [Function File]

Compute utilization, response time, average number of requests and throughput for a $M/M/1/K$ finite capacity system. In a $M/M/1/K$ queue there is a single server, the maximum number of requests is K, and the maximum queue length is $K-1$.

The steady-state probability π_k that there are k jobs in the system, $0 \leq k \leq K$, can be computed as:

$$\pi_k = \frac{(1-a)a^k}{1-a^{K+1}}$$

where $a = \lambda/\mu$.

INPUTS

lambda Arrival rate (jobs/s, `lambda>0`).

mu Service rate (jobs/s, `mu>0`).

K Maximum number of requests allowed in the system (`K` $\geq$ `1`).

OUTPUTS

U Service center utilization, which is defined as `U = 1-p0`

R Service center response time

Q Average number of requests in the system

X Service center throughput

p0 Steady-state probability that there are no requests in the system

pK Steady-state probability that there are K requests in the system (i.e., that the system is full)

lambda, *mu* and K can be vectors of the same size. In this case, the results will be vectors as well.

See also: qnmm1,qnmminf,qnmmm.

5.5 The $M/M/m/K$ System

The $M/M/m/K$ finite capacity system is similar to the $M/M/1/k$ system except that the number of servers is m, where $1 \leq m \leq K$. The queue is made of $K - m$ slots. The $M/M/m/K$ system is always stable.

`[U, R, Q, X, p0, pK] = qnmmmk (lambda, mu, m, K)` [Function File]

Compute utilization, response time, average number of requests and throughput for a $M/M/m/K$ finite capacity system. In a $M/M/m/K$ queue there are $m \geq 1$ service centers. At any time, at most $K \geq m$ requests can be in the system. This function generates and solves the underlying CTMC.

The steady-state probability π_k that there are k jobs in the system, $0 \leq k \leq K$ can be expressed as:

$$\pi_k = \begin{cases} \dfrac{\rho^k}{k!}\pi_0 & \text{if } 0 \leq k \leq m; \\ \dfrac{\rho^m}{m!}\left(\dfrac{\rho}{m}\right)^{k-m}\pi_0 & \text{if } m < k \leq K \end{cases}$$

where $\rho = \lambda/\mu$ is the offered load. The probability π_0 that the system is empty can be computed by considering that all probabilities must sum to one: $\sum_{k=0}^{K} \pi_k = 1$, which gives:

$$\pi_0 = \left[\sum_{k=0}^{m} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \sum_{k=m+1}^{K} \left(\frac{\rho}{m}\right)^{k-m} \right]^{-1}$$

INPUTS

lambda Arrival rate (`lambda>0`).

mu Service rate (`mu>0`).

m Number of servers (`m` $\geq$ `1`).

K Maximum number of requests allowed in the system, including those inside the service centers (`K` $\geq$ `m`).

OUTPUTS

U Service center utilization

R Service center response time

Q Average number of requests in the system

X Service center throughput

p0 Steady-state probability that there are no requests in the system.

pK Steady-state probability that there are K requests in the system (i.e., probability that the system is full).

lambda, *mu*, *m* and *K* can be either scalars, or vectors of the same size. In this case, the results will be vectors as well.

See also: qnmm1,qnmminf,qnmmm.

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, Section 6.6.

5.6 The Asymmetric $M/M/m$ System

The Asymmetric $M/M/m$ system contains m servers connected to a single queue. Differently from the $M/M/m$ system, in the asymmetric $M/M/m$ each server may have a different service time.

`[U, R, Q, X] = qnammm (lambda, mu)` [Function File]

Compute *approximate* utilization, response time, average number of requests in service and throughput for an asymmetric $M/M/m$ queue. In this system there are m different service centers connected to a single queue. Each server has its own (possibly different) service time. If there is more than one server available, the requests are directed to a randomly-chosen one.

INPUTS

lambda Arrival rate (jobs/s, `lambda>0`).

mu `mu(i)` is the service rate (jobs/s) of server i, $1 \leq i \leq m$. The system must be ergodic (`lambda < sum(mu)`).

OUTPUTS

U Approximate service center utilization, $U = \lambda/(\sum_i \mu_i)$.

R Approximate service center response time

Q Approximate number of requests in the system

X Approximate service center throughput. If the system is ergodic, we will always have `X = lambda`

See also: qnmmm.

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998

5.7 The $M/G/1$ System

`[U, R, Q, X, p0] = qnmg1 (lambda, xavg, x2nd)` [Function File]

Compute utilization, response time, average number of requests and throughput for a $M/G/1$ system. The service time distribution is described by its mean *xavg*, and by its second moment *x2nd*. The computations are based on results from L. Kleinrock, *Queuing Systems*, Wiley, Vol 2, and Pollaczek-Khinchine formula.

INPUTS

lambda Arrival rate.

xavg Average service time

x2nd Second moment of service time distribution

OUTPUTS

U Service center utilization

R Service center response time

Q Average number of requests in the system

X Service center throughput

p0 probability that there is not any request at system

lambda, *xavg*, *t2nd* can be vectors of the same size. In this case, the results will be vectors as well.

See also: qnmh1.

5.8 The $M/H_m/1$ System

`[U, R, Q, X, p0] = qnmh1 (lambda, mu, alpha)` [Function File]

Compute utilization, response time, average number of requests and throughput for a $M/H_m/1$ system. In this system, the customer service times have hyper-exponential distribution:

$$B(x) = \sum_{j=1}^{m} \alpha_j (1 - e^{-\mu_j x}), \quad x > 0$$

where α_j is the probability that the request is served at phase j, in which case the average service rate is μ_j. After completing service at phase j, for some j, the request exits the system.

INPUTS

lambda Arrival rate.

mu `mu(j)` is the phase j service rate. The total number of phases m is `length(mu)`.

alpha `alpha(j)` is the probability that a request is served at phase j. *alpha* must have the same size as *mu*.

OUTPUTS

U Service center utilization

R Service center response time

Q Average number of requests in the system

X Service center throughput

6 Queueing Networks

6.1 Introduction

Queueing Networks (QN) are a very simple yet powerful modeling tool which is used to analyze many kind of systems. In its simplest form, a QN is made of K service centers. Each service center i has a queue, which is connected to m_i (generally identical) *servers*. Customers (or requests) arrive at the service center, and join the queue if there is a slot available. Then, requests are served according to a (de)queueing policy. After service completes, the requests leave the service center.

The service centers for which $m_i = \infty$ are called *delay centers* or *infinite servers*. If a service center has infinite servers, of course each new request will find one server available, so there will never be queueing.

Requests join the queue according to a *queueing policy*, such as:

FCFS First-Come-First-Served

LCFS-PR Last-Come-First-Served, Preemptive Resume

PS Processor Sharing

IS Infinite Server, there is an infinite number of identical servers so that each request always finds a server available, and there is no queueing

A population of *requests* or *customers* arrives to the system system, requesting service to the service centers. The request population may be *open* or *closed*. In open systems there is an infinite population of requests. New customers arrive from outside the system, and eventually leave the system. In closed systems there is a fixed population of request which continuously interacts with the system.

There might be a single class of requests, meaning that all requests behave in the same way (e.g., they spend the same average time on each particular server), or there might be multiple classes of requests.

6.1.1 Single class models

In single class models, all requests are indistinguishable and belong to the same class. This means that every request has the same average service time, and all requests move through the system with the same routing probabilities.

Model Inputs

$\backslash lambda_i$ External arrival rate to service center i.

$\backslash lambda$ Overall external arrival rate to the whole system: $\lambda = \sum_i \lambda_i$.

S_i Average service time. S_i is the average service time on service center i. In other words, S_i is the average time from the instant in which a request is extracted from the queue and starts being service, and the instant at which service finishes and the request moves to another queue (or exits the system).

P_{ij} Routing probability matrix. $\mathbf{P} = P_{ij}$ is a $K \times K$ matrix such that P_{ij} is the probability that a request completing service at server i will move directly to server j, The probability that a request leaves the system after service at service center i is $1 - \sum_{j=1}^{K} P_{ij}$.

V_i Average number of visits. V_i is the average number of visits to the service center i. This quantity will be described shortly.

Model Outputs

U_i Service center utilization. U_i is the utilization of service center i. The utilization is defined as the fraction of time in which the resource is busy (i.e., the server is processing requests).

R_i Average response time. R_i is the average response time of service center i. The average response time is defined as the average time between the arrival of a customer in the queue, and the completion of service.

Q_i Average number of customers. Q_i is the average number of requests in service center i. This includes both the requests in the queue, and the request being served.

X_i Throughput. X_i is the throughput of service center i. The throughput is defined as the ratio of job completions (i.e., average number of jobs completed over a fixed interval of time).

Given these output parameters, additional performance measures can be computed as follows:

X System throughput, $X = X_1/V_1$

R System response time, $R = \sum_{k=1}^{K} R_k V_k$

Q Average number of requests in the system, $Q = N - XZ$

For open, single-class models, the scalar λ denotes the external arrival rate of requests to the system. The average number of visits satisfy the following equation:

$V_j = P_{0j} + \sum_{i=1}^{K} V_i P_{ij}$

where P_{0j} is the probability that an external arrival goes to service center j. If λ_j is the external arrival rate to service center j, and $\lambda = \sum_j \lambda_j$ is the overall external arrival rate, then $P_{0j} = \lambda_j/\lambda$.

For closed models, the scalar N denotes the population size, i.e., the number of requests in the system. For closed networks, the visit ratios satisfy the following equation:

$V_j = \sum_{i=1}^{K} V_i P_{ij},\ V_1 = 1$

6.1.2 Multiple class models

In multiple class QN models, we assume that there exist C different classes of requests. Each request from class c spends on average time S_{ck} in service at service center k. For open models, we denote with $\lambda = \lambda_{ck}$ the arrival rates, where λ_{ck} is the external arrival rate of class c customers at service center k. For closed models, we denote with $\mathbf{N} = (N_1, N_2, \ldots N_C)$ the population vector, where N_c is the number of class c requests in the system.

The transition probability matrix for these kind of networks will be a $C \times K \times C \times K$ matrix $\mathbf{P} = P_{risj}$ such that P_{risj} is the probability that a class r request which completes service at center i will join server j as a class s request.

Model input and outputs can be adjusted by adding additional indexes for the customer classes.

Model Inputs

$\backslash lambda_{ci}$ External arrival rate of class-c requests to service center i

$\backslash lambda$ Overall external arrival rate to the whole system: $\lambda = \sum_c \sum_i \lambda_{ci}$

S_{ci} Average service time. S_{ci} is the average service time on service center i for class c requests.

P_{risj} Routing probability matrix. $\mathbf{P} = P_{risj}$ is a $C \times K \times C \times K$ matrix such that P_{risj} is the probability that a class r request which completes service at server i will move to server j as a class s request.

V_{ci} Average number of visits. V_{ci} is the average number of visits of class c requests to the service center i.

Model Outputs

U_{ci} Utilization of service center i by class c requests. The utilization is defined as the fraction of time in which the resource is busy (i.e., the server is processing requests).

R_{ci} Average response time experienced by class c requests on service center i. The average response time is defined as the average time between the arrival of a customer in the queue, and the completion of service.

Q_{ci} Average number of class c requests on service center i. This includes both the requests in the queue, and the request being served.

X_{ci} Throughput of service center i for class c requests. The throughput is defined as the rate of completion of class c requests.

It is possible to define aggregate performance measures as follows:

U_i Utilization of service center i: $U_i = \sum_{c=1}^{C} U_{ci}$

R_c System response time for class c requests: $R_c = \sum_{i=1}^{K} R_{ci} V_{ci}$

Q_c Average number of class c requests in the system: $Q_c = \sum_{i=1}^{K} Q_{ci}$

X_c Class c throughput: $X_c = X_{c1}/V_{c1}$

We can define the visit ratios V_{sj} for class s customers at service center j as follows:

$$V_{sj} = \sum_{r=1}^{C} \sum_{i=1}^{K} V_{ri} P_{risj},\ V_{s1} = 1$$

while for open networks:

$$V_{sj} = P_{0sj} + \sum_{r=1}^{C} \sum_{i=1}^{K} V_{ri} P_{risj}$$

where P_{0sj} is the probability that an external arrival goes to service center j as a class-s request. If λ_{sj} is the external arrival rate of class s requests to service center j, and $\lambda = \sum_s \sum_j \lambda_{sj}$ is the overall external arrival rate to the whole system, then $P_{0sj} = \lambda_{sj}/\lambda$.

6.2 Generic Queueing Network Analysis

The `qnetworks` package provides a couple of high-level functions for defining and solving QN models. These functions can be used to define a open or closed QN model (with single or multiple job classes), with arbitrary configuration and queueing disciplines. At the moment only product-form networks can be solved, See Section 6.3 [Algorithms for Product-Form QN], page 28.

The network is defined by two parameters. The first one is the list of nodes, encoded as an Octave *cell array*. The second parameter is the visit ration *V*, which can be either a vector (for single-class models) or a two-dimensional matrix (for multiple-class models).

Individual nodes in the network are structures build using the `qnmknode` function.

`Q = qnmknode ("m/m/m-fcfs", S)` [Function File]
`Q = qnmknode ("m/m/m-fcfs", S, m)` [Function File]
`Q = qnmknode ("m/m/1-lcfs-pr", S)` [Function File]
`Q = qnmknode ("-/g/1-ps", S)` [Function File]
`Q = qnmknode ("-/g/1-ps", S, s2)` [Function File]
`Q = qnmknode ("-/g/inf", S)` [Function File]
`Q = qnmknode ("-/g/inf", S, s2)` [Function File]

Creates a node; this function can be used together with `qnsolve`. It is possible to create either single-class nodes (where there is only one customer class), or multiple-class nodes (where the service time is given per-class). Furthermore, it is possible to specify load-dependent service times.

INPUTS

S Average service time. *S* can be either a scalar, a row vector, a column vector or a two-dimensional matrix.

- If *S* is a scalar, it is assumed to be a load-independent, class-independent service time.
- If *S* is a column vector, then `S(c)` is assumed to the the load-independent service time for class *c* customers.
- If *S* is a row vector, then `S(n)` is assumed to be the class-independent service time at the node, when there are *n* requests.
- Finally, if *S* is a two-dimensional matrix, then `S(c,n)` is assumed to be the class *c* service time when there are *n* requests at the node.

m Number of identical servers at the node. Default is `m=1`.

s2 Squared coefficient of variation for the service time. Default is 1.0.

The returned struct *Q* should be considered opaque to the client.

See also: qnsolve.

After the network has been defined, it is possible to solve it using the `qnsolve` function. Note that this function is somewhat less efficient than those described in later sections, but generally easier to use.

`[U, R, Q, X] = qnsolve ("closed", N, QQ, V)` [Function File]
`[U, R, Q, X] = qnsolve ("closed", N, QQ, V, Z)` [Function File]
`[U, R, Q, X] = qnsolve ("open", lambda, QQ, V)` [Function File]
`[U, R, Q, X] = qnsolve ("mixed", lambda, N, QQ, V)` [Function File]

General evaluator of QN models. Networks can be open, closed or mixed; single as well as multiclass networks are supported.

- For **closed** networks, the following server types are supported: $M/M/m$–FCFS, $-/G/\infty$, $-/G/1$–LCFS-PR, $-/G/1$–PS and load-dependent variants.

- For **open** networks, the following server types are supported: $M/M/m$–FCFS, $-/G/\infty$ and $-/G/1$–PS. General load-dependent nodes are *not* supported. Multiclass open networks do not support multiple server $M/M/m$ nodes, but only single server $M/M/1$–FCFS.
- For **mixed** networks, the following server types are supported: $M/M/1$–FCFS, $-/G/\infty$ and $-/G/1$–PS. General load-dependent nodes are *not* supported.

INPUTS

N Number of requests in the system for closed networks. For single-class networks, N must be a scalar. For multiclass networks, *`N(c)`* is the population size of closed class c.

lambda External arrival rate (scalar) for open networks. For single-class networks, *lambda* must be a scalar. For multiclass networks, `lambda(c)` is the class c overall arrival rate.

QQ List of queues in the network. This must be a cell array with N elements, such that *`QQ{i}`* is a struct produced by the `qnmknode` function.

Z External delay ("think time") for closed networks. Default 0.

OUTPUTS

U If i is a FCFS node, then *`U(i)`* is the utilization of service center i. If i is an IS node, then *`U(i)`* is the *traffic intensity* defined as *`X(i)*S(i)`*.

R *`R(i)`* is the average response time of service center i.

Q *`Q(i)`* is the average number of customers in service center i.

X *`X(i)`* is the throughput of service center i.

Note that for multiclass networks, the computed results are per-class utilization, response time, number of customers and throughput: *`U(c,k)`*, *`R(c,k)`*, *`Q(c,k)`*, *`X(c,k)`*,

EXAMPLE

Let us consider a closed, multiclass network with $C = 2$ classes and $K = 3$ service center. Let the population be $M = (2, 1)$ (class 1 has 2 requests, and class 2 has 1 request). The nodes are as follows:

- Node 1 is a $M/M/1$–FCFS node, with load-dependent service times. Service times are class-independent, and are defined by the matrix `[0.2 0.1 0.1; 0.2 0.1 0.1]`. Thus, *`S(1,2) = 0.2`* means that service time for class 1 customers where there are 2 requests in 0.2. Note that service times are class-independent;
- Node 2 is a $-/G/1$–PS node, with service times $S_{12} = 0.4$ for class 1, and $S_{22} = 0.6$ for class 2 requests;
- Node 3 is a $-/G/\infty$ node (delay center), with service times $S_{13} = 1$ and $S_{23} = 2$ for class 1 and 2 respectively.

After defining the per-class visit count V such that *`V(c,k)`* is the visit count of class c requests to service center k. We can define and solve the model as follows:

```
QQ = { qnmknode( "m/m/m-fcfs", [0.2 0.1 0.1; 0.2 0.1 0.1] ), \
       qnmknode( "-/g/1-ps", [0.4; 0.6] ), \
       qnmknode( "-/g/inf", [1; 2] ) };
V = [ 1 0.6 0.4; \
      1 0.3 0.7 ];
N = [ 2 1 ];
[U R Q X] = qnsolve( "closed", N, QQ, V );
```

6.3 Algorithms for Product-Form QN

Product-form queueing networks fulfill the following assumptions:

- The network can consist of open and closed job classes.
- The following queueing disciplines are allowed: FCFS, PS, LCFS-PR and IS.
- Service times for FCFS nodes must be exponentially distributed and class-independent. Service centers at PS, LCFS-PR and IS nodes can have any kind of service time distribution with a rational Laplace transform. Furthermore, for PS, LCFS-PR and IS nodes, different classes of customers can have different service times.
- The service rate of an FCFS node is only allowed to depend on the number of jobs at this node; in a PS, LCFS-PR and IS node the service rate for a particular job class can also depend on the number of jobs of that class at the node.
- In open networks two kinds of arrival processes are allowed: i) the arrival process is Poisson, with arrival rate λ which can depend on the number of jobs in the network. ii) the arrival process consists of U independent Poisson arrival streams where the U job sources are assigned to the U chains; the arrival rate can be load dependent.

6.3.1 Jackson Networks

Jackson networks satisfy the following conditions:

- There is only one job class in the network; the overall number of jobs in the system is unlimited.
- There are N service centers in the network. Each service center may have Poisson arrivals from outside the system. A job can leave the system from any node.
- Arrival rates as well as routing probabilities are independent from the number of nodes in the network.
- External arrivals and service times at the service centers are exponentially distributed, and in general can be load-dependent.
- Service discipline at each node is FCFS

We define the *joint probability vector* $\pi(k_1, k_2, \ldots k_N)$ as the steady-state probability that there are k_i requests at service center i, for all $i = 1, 2, \ldots N$. Jackson networks have the property that the joint probability is the product of the marginal probabilities π_i:

$$\pi(k_1, k_2, \ldots k_N) = \prod_{i=1}^{N} \pi_i(k_i)$$

where $\pi_i(k_i)$ is the steady-state probability that there are k_i requests at service center i.

`[U, R, Q, X] = qnjackson (lambda, S, P )` [Function File]
`[U, R, Q, X] = qnjackson (lambda, S, P, m )` [Function File]
`pr = qnjackson (lambda, S, P, m, k)` [Function File]

With three or four input parameters, this function computes the steady-state occupancy probabilities for a Jackson network. With five input parameters, this function computes the steady-state probability `pi(j)` that there are `k(j)` requests at service center j.

This function solves a subset of Jackson networks, with the following constraints:

- External arrival rates are load-independent.
- Service center i consists either of `m(i)` $\geq$ `1` identical servers with individual average service time `S(i)`, or of an Infinite Server (IS) node.

INPUTS

lambda
: `lambda(i)` is the external arrival rate to service center i. *lambda* must be a vector of length N, `lambda(i)` $\geq$ `0`.

S
: `S(i)` is the average service time on service center i *S* must be a vector of length N, `S(i)>0`.

P
: `P(i,j)` is the probability that a job which completes service at service center i proceeds to service center j. *P* must be a matrix of size $N \times N$.

m
: `m(i)` is the number of servers at service center i. If `m(i) < 1`, service center i is an infinite-server node. Otherwise, it is a regular FCFS queueing center with `m(i)` servers. If this parameter is omitted, default is `m(i) = 1` for all i. If this parameter is a scalar, it will be promoted to a vector with the same size as *lambda*. Otherwise, *m* must be a vector of length N.

k
: Compute the steady-state probability that there are `k(i)` requests at service center i. *k* must have the same length as *lambda*, with `k(i)` $\geq$ `0`.

OUTPUT

U
: If i is a FCFS node, then `U(i)` is the utilization of service center i. If i is an IS node, then `U(i)` is the *traffic intensity* defined as `X(i)*S(i)`.

R
: `R(i)` is the average response time of service center i.

Q
: `Q(i)` is the average number of customers in service center i.

X
: `X(i)` is the throughput of service center i.

pr
: `pr(i)` is the steady state probability that there are `k(i)` requests at service center i.

See also: qnopen.

REFERENCES

This implementation is based on G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, pp. 284–287.

6.3.2 The Convolution Algorithm

According to the BCMP theorem, the state probability of a closed single class queueing network with K nodes and N requests can be expressed as:

$$\pi(k_1, k_2, \ldots k_K) = \frac{1}{G(N)} \prod_{i=1}^{N} F_i(k_i)$$

Here $\pi(k_1, k_2, \ldots k_K)$ is the joint probability of having k_i requests at node i, for all $i = 1, 2, \ldots K$.

The *convolution algorithms* computes the normalization constant $G = (G(0), G(1), \ldots G(N))$ for single-class, closed networks with N requests. The normalization constants are returned as the Octave vector `G=[G(1), G(2), ... G(N+1)]` where `G(i+1)` is the value of $G(i)$, as Octave uses 1-base vectors. The normalization constant can be used to compute all performance measures of itnerest (utilization, average response time and so on).

`qnetworks` implements the convolution algorithm, in the function `qnconvolution` and `qnconvolutionld`. The first one supports single-station nodes, multiple-station nodes and IS nodes. The second one supports networks with general load-dependent service centers.

`[U, R, Q, X, G] = qnconvolution (N, S, V)` [Function File]
`[U, R, Q, X, G] = qnconvolution (N, S, V, m)` [Function File]

This function implements the *convolution algorithm* for product-form, single-class closed queueing networks. Load-independent service centers, multiple servers ($M/M/m$ queues) and IS nodes are supported. For general load-dependent service centers, use the `qnconvolutionld` function instead.

INPUTS

N Number of requests in the system (`N>0`).

S `S(k)` is the average service time on center k (`S(k)` $\geq$ `0`).

V `V(k)` is the visit count of service center k (`V(k)` $\geq$ `0`).

m `m(k)` is the number of servers at center k. If `m(k) < 1`, center k is a delay center (IS) otherwise it is a regular $M/M/m$ queueing center with `m(k)` identical servers. Default is `m(k) = 1` for all k.

OUTPUT

U `U(k)` is the utilization of center k. For IS nodes, `U(k)` is the *traffic intensity*.

R `R(k)` is the average response time of center k.

Q `Q(k)` is the average number of customers at center k.

X `X(k)` is the throughput of center k.

G Normalization constants (vector). `G(n+1)` corresponds to the normalization constant with n requests $G(n)$, $n = 0, 1, \ldots N$ as array indexes in Octave start from 1.

See also: qnconvolutionld.

EXAMPLE

The normalization constant G can be used to compute the steady-state probabilities for a closed single class product-form Queueing Network with K nodes. Let *k=*$[k_1, k_2, \ldots k_K]$ be a valid population vector. Then, the steady-state probability *p(i)* to have *k(i)* requests at service center i can be computed as:

$$p_i(k_i) = \frac{(V_i S_i)^{k_i}}{G(K)} \left(G(K - k_i) - V_i S_i G(K - k_i - 1) \right), \quad i = 1, 2, \ldots K$$

```
k = [1 2 0];
K = sum(k); # Total population size
S = [ 1/0.8 1/0.6 1/0.4 ];
m = [ 2 3 1 ];
V = [ 1 .667 .2 ];
[U R Q X G] = qnconvolution( K, S, V, m );
p = [0 0 0]; # initialize p
# Compute the probability to have k(i) jobs at service center i
for i=1:3
  p(i) = (V(i)*S(i))^k(i) / G(K+1) * \
         (G(K-k(i)+1) - V(i)*S(i)*G(K-k(i)) );
  printf("i=%d k(i)=%d prob=%f\n", i, k(i), p(i) );
endfor
⊣ i=1 k(i)=1 prob=0.17975
⊣ i=2 k(i)=2 prob=0.48404
⊣ i=3 k(i)=0 prob=0.52779
```

NOTE

For a network with K service centers and population size N, this implementation of the convolution algorithm has time and space complexity $O(NK)$.

REFERENCES

Jeffrey P. Buzen, *Computational Algorithms for Closed Queueing Networks with Exponential Servers*, Communications of the ACM, volume 16, number 9, september 1973, pp. 527–531. http://doi.acm.org/10.1145/362342.362345

This implementation is based on G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, pp. 313–317.

[U, R, Q, X, G] = qnconvolutionld (*N*, *S*, *V*) [Function File]

This function implements the *convolution algorithm* for product-form, single-class closed queueing networks with general load-dependent service centers.

This function computes the normalization constant $G = (G(0), G(1), \ldots G(N))$ for single-class, closed networks with load-dependent service centers using the convolution algorithm. The normalization constants are returned as vector *G=[G(1), G(2), ... G(N+1)]* where *G(i+1)* is the value of $G(i)$. Furthermore, this function computes utilization, response time, average number of customers and throughput of individual servers using the normalization constants.

INPUTS

N Number of requests in the system (`N>0`).

S `S(k,n)` is the average service time at center k where there are n requests (`S(k,n)` $\geq$ `0`). S can be a matrix or a function handle. If S is a function handle, it **must** be possible to evaluate `S(k,n)` when n is a vector.

V `V(k)` is the visit count of service center k (`V(k)` $\geq$ `0`). The length of V is the number of servers K in the network.

OUTPUT

U `U(k)` is the utilization of center k.

R `R(k)` is the average response time at center k.

Q `Q(k)` is the average number of customers in center k.

X `X(k)` is the throughput of center k.

G Normalization constants (vector). `G(n+1)` corresponds to $G(n)$, as array indexes in Octave start from 1.

See also: qnconvolution.

REFERENCES

Herb Schwetman, *Some Computational Aspects of Queueing Network Models*, Technical Report CSD-TR-354, Department of Computer Sciences, Purdue University, feb, 1981 (revised). http://www.cs.purdue.edu/research/technical_reports/1980/TR%2080-354.pdf

M. Reiser, H. Kobayashi, *On The Convolution Algorithm for Separable Queueing Networks*, In Proceedings of the 1976 ACM SIGMETRICS Conference on Computer Performance Modeling Measurement and Evaluation (Cambridge, Massachusetts, United States, March 29–31, 1976). SIGMETRICS '76. ACM, New York, NY, pp. 109–117. http://doi.acm.org/10.1145/800200.806187

This implementation is based on G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, pp. 313–317. Function `qnconvolutionld` is slightly different from the version described in Bolch et al. because it supports general load-dependent centers (while the version in the book does not). The modification is in the definition of function `F()` in `qnconvolutionld` which has been made similar to function f_i defined in Schwetman, `Some Computational Aspects of Queueing Network Models`.

6.3.3 Open networks

`[U, R, Q, X] = qnopensingle (lambda, S, V)` [Function File]
`[U, R, Q, X] = qnopensingle (lambda, S, V, m)` [Function File]

Analyze open, single class BCMP queueing networks.

This function works for a subset of BCMP single-class open networks satisfying the following properties:

- The allowed service disciplines at network nodes are: FCFS, PS, LCFS-PR, IS (infinite server);

- Service times are exponentially distributed and load-independent;
- Service center i can consist of `m(i)` $\geq$ `1` identical servers.
- Routing is load-independent

INPUTS

lambda Overall external arrival rate (`lambda>0`).

S `S(k)` is the average service time at center i (`S(k)>0`).

V `V(k)` is the average number of visits to center k (`V(k)` $\geq$ `0`).

m `m(k)` is the number of servers at center i. If `m(k) < 1`, then service center k is a delay center (IS); otherwise it is a regular queueing center with `m(k)` servers. Default is `m(k) = 1` for each k.

OUTPUTS

U If k is a queueing center, `U(k)` is the utilization of center k. If k is an IS node, then `U(k)` is the *traffic intensity* defined as `X(k)*S(k)`.

R `R(k)` is the average response time of center k.

Q `Q(k)` is the average number of requests at center k.

X `X(k)` is the throughput of center k.

See also: qnopen,qnclosed,qnvisits.

From the results computed by this function, it is possible to derive other quantities of interest as follows:

- **System Response Time**: The overall system response time can be computed as $R_s = \sum_{i=1}^{K} V_i R_i$
- **Average number of requests**: The average number of requests in the system can be computed as: $Q_s = \sum_{i=1}^{K} Q(i)$

EXAMPLE

```
lambda = 3;
V = [16 7 8];
S = [0.01 0.02 0.03];
[U R Q X] = qnopensingle( lambda, S, V );
R_s = dot(R,V) # System response time
N = sum(Q) # Average number in system
⊣ R_s =  1.4062
⊣ N =  4.2186
```

REFERENCES

G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998.

`[U, R, Q, X] = qnopenmulti (lambda, S, V)` [Function File]
`[U, R, Q, X] = qnopenmulti (lambda, S, V, m)` [Function File]

Exact analysis of open, multiple-class BCMP networks. The network can be made of *single-server* queueing centers (FCFS, LCFS-PR or PS) or delay centers (IS). This function assumes a network with K service centers and C customer classes.

INPUTS

lambda
: `lambda(c)` is the external arrival rate of class c customers (`lambda(c)>0`).

S
: `S(c,k)` is the mean service time of class c customers on the service center k (`S(c,k)>0`). For FCFS nodes, average service times must be class-independent.

V
: `V(c,k)` is the average number of visits of class c customers to service center k (`V(c,k)` $\geq$ `0`).

m
: `m(k)` is the number of servers at service center k. Valid values are `m(k) < 1` to denote a delay center ($-/G/\infty$), and `m(k)==1` to denote a single server queueing center ($M/M/1$–FCFS, $-/G/1$–LCFS-PR or $-/G/1$–PS).

OUTPUTS

U
: If k is a queueing center, then `U(c,k)` is the class c utilization of center k. If k is an IS node, then `U(c,k)` is the class c *traffic intensity* defined as `X(c,k)*S(c,k)`.

R
: `R(c,k)` is the class c response time at center k. The system response time for class c requests can be computed as `dot(R, V, 2)`.

Q
: `Q(c,k)` is the average number of class c requests at center k. The average number of class c requests in the system Qc can be computed as `Qc = sum(Q, 2)`

X
: `X(c,k)` is the class c throughput at center k.

See also: qnopen,qnopensingle,qnvisits.

REFERENCES

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 7.4.1 ("Open Model Solution Techniques").

6.3.4 Closed Networks

`[U, R, Q, X, G] = qnclosedsinglemva (N, S, V)` [Function File]
`[U, R, Q, X, G] = qnclosedsinglemva (N, S, V, m)` [Function File]
`[U, R, Q, X, G] = qnclosedsinglemva (N, S, V, m, Z)` [Function File]

Analyze closed, single class queueing networks using the exact Mean Value Analysis (MVA) algorithm. The following queueing disciplines are supported: FCFS, LCFS-PR, PS and IS (Infinite Server). This function supports fixed-rate service centers or multiple server nodes. For general load-dependent service centers, use the function `qnclosedsinglemvald` instead.

Additionally, the normalization constant $G(n)$, $n = 0, 1, \ldots N$ is computed; $G(n)$ can be used in conjunction with the BCMP theorem to compute steady-state probabilities.

INPUTS

N
: Population size (number of requests in the system, `N > 0`).

S `S(k)` is the mean service time on server k (`S(k)>0`).

V `V(k)` is the average number of visits to service center k (`V(k)` $\geq$ `0`).

m `m(k)` is the number of servers at center k. if `m(k) < 1`, center k is a delay center (IS); otherwise it is a regular queueing center (FCFS, LCFS-PR or PS) with `m(k)` servers. Default is `m(k) = 1` (each service center has a single server).

Z External delay for customers (`Z` $\geq$ `0`). Default is 0.

OUTPUTS

U If k is a FCFS, LCFS-PR or PS node, then `U(k)` is the utilization of center k. If k is an IS node, then `U(k)` is the *traffic intensity* defined as `X(k)*S(k)`.

R `R(k)` is the response time at center k. The system response time *Rsys* can be computed as `Rsys = N/Xsys - Z`

Q `Q(k)` is the average number of requests at center k. The number of requests in the system can be computed either as `sum(Q)`, or using the formula `N-Xsys*Z`.

X `X(k)` is the throughput of center k. The system throughput *Xsys* can be computed as `Xsys = X(1) / V(1)`

G Normalization constants. `G(n+1)` corresponds to the value of the normalization constant $G(n)$, $n = 0, 1, \ldots N$ as array indexes in Octave start from 1. $G(n)$ can be used in conjunction with the BCMP theorem to compute steady-state probabilities.

See also: qnclosedsinglemvald.

From the results provided by this function, it is possible to derive other quantities of interest as follows:

EXAMPLE

```
S = [ 0.125 0.3 0.2 ];
V = [ 16 10 5 ];
N = 20;
m = ones(1,3);
Z = 4;
[U R Q X] = qnclosedsinglemva(N,S,V,m,Z);
X_s = X(1)/V(1); # System throughput
R_s = dot(R,V); # System response time
printf("System throughput\t%f\n", X_s );
printf("System Response Time\t%f\n", R_s );
printf("Avg.  number in system\t%f\n\n", N-X_s*Z );
printf("\t\tUtil\t\tQlen\t\tRes.  time\tTput\n");
for k=1:length(S)
  printf("Device %d\t%f\t%f\t%f\t%f\n", k, U(k), Q(k), R(k), X(k) );
endfor
```

REFERENCES

M. Reiser and S. S. Lavenberg, *Mean-Value Analysis of Closed Multichain Queuing Networks*, Journal of the ACM, vol. 27, n. 2, April 1980, pp. 313–322. http://doi.acm.org/10.1145/322186.322195

This implementation is described in R. Jain , *The Art of Computer Systems Performance Analysis*, Wiley, 1991, p. 577. Multi-server nodes are treated according to G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, Section 8.2.1, "Single Class Queueing Networks".

`[U, R, Q, X] = qnclosedsinglemvald (N, S, V)` [Function File]
`[U, R, Q, X] = qnclosedsinglemvald (N, S, V, Z)` [Function File]

Exact MVA algorithm for closed, single class queueing networks with load-dependent service centers. This function supports FCFS, LCFS-PR, PS and IS nodes. For networks with only fixed-rate service centers and multiple-server nodes, the function `qnclosedsinglemva` is more efficient.

INPUTS

N Population size (number of requests in the system, `N >0`).

S `S(k,n)` denotes the mean service time at server k where there are n requests at that center, $1 \leq n \leq N$ (`S(k,n)` $\geq$ `0`). *S* can be either a matrix or a function handle. If *S* is a function handle, it must support vector arguments (i.e., it must be possible to evaluate `S(k,n)` with k and/or n being vectors).

V `V(k)` is the average number of visits to service center k (`V(k)` $\geq$ `0`).

Z external delay ("think time", `Z` $\geq$ `0`); default 0.

OUTPUTS

U `U(k)` is the utilization of service center k. The utilization is defined as the probability that service center k is not empty, that is, $U_k = 1 - \pi_k(0)$ where $\pi_k(0)$ is the steady-state probability that there are 0 jobs at service center k.

R `R(k)` is the response time on service center k.

Q `Q(k)` is the average number of requests in service center k.

X `X(k)` is the throughput of service center k.

REFERENCES

M. Reiser and S. S. Lavenberg, *Mean-Value Analysis of Closed Multichain Queuing Networks*, Journal of the ACM, vol. 27, n. 2, April 1980, pp. 313–322. http://doi.acm.org/10.1145/322186.322195

This implementation is described in G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998, Section 8.2.4.1, “Networks with Load-Dependent Service: Closed Networks”.

`[U, R, Q, X] = qnclosedmultimva (N, S, V)` [Function File]
`[U, R, Q, X] = qnclosedmultimva (N, S, V, m)` [Function File]
`[U, R, Q, X] = qnclosedmultimva (N, S, V, m, Z)` [Function File]

Analyze closed, multiclass queueing networks with K service centers and C independent customer classes (chains) using the Mean Value Analysys (MVA) algorithm. Only fixed-rate service centers or multiple server nodes are supported. The number of requests in each class (chain) is fixed. Actually this implementation requires that each customer class is an independent chain.

Queueing policies at service centers can be any of the following:

FCFS (First-Come-First-Served) customers are served in order of arrival; multiple servers are allowed. For this kind of queueing discipline, average service times must be class-independent.

PS (Processor Sharing) customers are served in parallel by a single server, each customer receiving an equal share of the service rate.

LCFS-PR (Last-Come-First-Served, Preemptive Resume) customers are served in reverse order of arrival by a single server and the last arrival preempts the customer in service who will later resume service at the point of interruption.

IS (Infinite Server) customers are delayed independently of other customers at the service center (there is effectively an infinite number of servers).

INPUTS

N `N(c)` is the number of class c requests in the system (`N(c) > 0`).

S `S(c,k)` is the mean service time for class c customers at center k (`S(c,k)` $\geq$ `0`). If center k is a FCFS node (`m(k)>1`), then the service times must be class-independent: `all(S(:,k) == S(1,k))`.

V `V(c,k)` is the average number of visits of class c customers to service center k (`V(c,k)` $\geq$ `0`.

m If `m(k)<1`, then center k is assumed to be a delay center (IS node $-/G/\infty$). If `m(k)==1`, then service center k is a regular queueing center ($M/M/1$–FCFS, $-/G/1$–LCFS-PR or $-/G/1$–PS). Finally, if `m(k)>1`, center k is a $M/M/m$–FCFS center with `m(k)` identical servers. Default is `m(k)=1` for each k.

Z `Z(c)` is the class c external delay (think time). Default is `Z(c)=0` for all c.

OUTPUTS

U If k is a FCFS, LCFS-PR or PS node, then `U(c,k)` is the class c utilization at center k. If k is IS, then `U(c,k)` is the class c *traffic intensity* at center k, defined as `U(c,k) = X(c,k)*S(c,k)`.

R `R(c,k)` is the class c response time at service center k. The class c system response time can be computed as `dot(R, V, 2)`.

Q `Q(c,k)` is the average number of class c requests at center k. The total number of requests at center k is `sum(Q(:,k))`. The total number of class c requests in the system is `sum(Q(c,:))`.

X `X(c,k)` is the class c throughput at center k. The class c system throughput can be computed as `X(c,1) / V(c,1)`.

See also: qnclosed, qnclosedmultimvaapprox.

NOTE

Given a network with K service centers, C job classes and population vector $\mathbf{N} = (N_1, N_2, \ldots N_C)$, the MVA algorithm requires space $O(C \prod_i (N_i + 1))$. The time complexity is $O(CK \prod_i (N_i + 1))$. This implementation is slightly more space-efficient (see details in the code). While the space requirement can be mitigated by using some optimizations, the time complexity can not. If you need to analyze large closed networks you should consider the `qnclosedmultimvaapprox` function, which implements the approximate MVA algorithm. Note however that `qnclosedmultimvaapprox` will only provide approximate results.

REFERENCES

M. Reiser and S. S. Lavenberg, *Mean-Value Analysis of Closed Multichain Queuing Networks*, Journal of the ACM, vol. 27, n. 2, April 1980, pp. 313–322. http://doi.acm.org/10.1145/322186.322195

This implementation is based on G. Bolch, S. Greiner, H. de Meer and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 1998 and Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 7.4.2.1 ("Exact Solution Techniques").

`[U, R, Q, X] = qnclosedmultimvaapprox (N, S, V)` [Function File]
`[U, R, Q, X] = qnclosedmultimvaapprox (N, S, V, m)` [Function File]
`[U, R, Q, X] = qnclosedmultimvaapprox (N, S, V, m, Z)` [Function File]
`[U, R, Q, X] = qnclosedmultimvaapprox (N, S, V, m, Z, epsilon)` [Function File]
`[U, R, Q, X] = qnclosedmultimvaapprox (N, S, V, m, Z, epsilon, iter_max)` [Function File]

Analyze closed, multiclass queueing networks with K service centers and C customer classes using the approximate Mean Value Analysys (MVA) algorithm.

This implementation uses Bard and Schweitzer approximation. It is based on the assumption that

$$Q_i(\mathbf{N} - \mathbf{1}_c) \approx \frac{n-1}{n} Q_i(\mathbf{N})$$

where $\mathbf{N}$ is a valid population mix, $\mathbf{N} - \mathbf{1}_c$ is the population mix $\mathbf{N}$ with one class c customer removed, and $n = \sum_c N_c$ is the total number of requests.

This implementation works for networks made of infinite server (IS) nodes and single-server nodes only.

INPUTS

N `N(c)` is the number of class *c* requests in the system (`N(c)>0`).

S `S(c,k)` is the mean service time for class *c* customers at center *k* (`S(c,k)` $\geq$ `0`).

V `V(c,k)` is the average number of visits of class *c* requests to center *k* (`V(c,k)` $\geq$ `0`).

m `m(k)` is the number of servers at service center *k*. If `m(k) < 1`, then the service center *k* is assumed to be a delay center (IS). If `m(k) == 1`, service center *k* is a regular queueing center (FCFS, LCFS-PR or PS) with a single server node. If omitted, each service center has a single server. Note that multiple server nodes are not supported.

Z `Z(c)` is the class *c* external delay. Default is `Z(c)=0` for all *c*.

epsilon Stopping tolerance (`epsilon>0`). The algorithm stops if the queue length computed on two subsequent iterations are less than *epsilon*. Default is $1e-5$.

iter_max Maximum number of iterations (`iter_max>0`, default is 100). The function aborts if convergenge is not reached within the maximum number of iterations.

OUTPUTS

U If *k* is a FCFS, LCFS-PR or PS node, then `U(c,k)` is the utilization of class *c* requests on service center *k*. If *k* is an IS node, then `U(c,k)` is the class *c* *traffic intensity* at device *k*, defined as `U(c,k) = X(c)*S(c,k)`

R `R(c,k)` is the response time of class *c* requests at service center *k*.

Q `Q(c,k)` is the average number of class *c* requests at service center *k*.

X `X(c,k)` is the class *c* throughput at service center *k*.

See also: qnclosed.

REFERENCES

Y. Bard, *Some Extensions to Multiclass Queueing Network Analysis*, proc. 4th Int. Symp. on Modelling and Performance Evaluation of Computer Systems, feb. 1979, pp. 51–62.

P. Schweitzer, *Approximate Analysis of Multiclass Closed Networks of Queues*, Proc. Int. Conf. on Stochastic Control and Optimization, jun 1979, pp. 25–29.

This implementation is based on Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 7.4.2.2 ("Approximate Solution Techniques"). This implementation is slightly different from the one described above, as it computes the average response times *R* instead of the residence times.

6.3.5 Mixed Networks

`[U, R, Q, X] = qnmix (lambda, N, S, V, m)` [Function File]

Solution of mixed queueing networks through MVA. The network consists of K service centers (single-server or delay centers) and C independent customer chains. Both open and closed chains are possible. *lambda* is the vector of per-chain arrival rates (open classes); N is the vector of populations for closed chains.

> **Note:** In this implementation class switching is **not** allowed. Each customer class *must* correspond to an independent chain.

If the network is made of open or closed classes only, then this function calls `qnopenmulti` or `qnclosedmultimva` respectively, and prints a warning message.

INPUTS

lambda
N
: For each customer chain c:
  - if c is a closed chain, then `N(c)>0` is the number of class c requests and `lambda(c)` must be zero;
  - If c is an open chain, `lambda(c)>0` is the arrival rate of class c requests and `N(c)` must be zero;

  For each c, the following must hold:

  `(lambda(c)>0 && N(c)==0) || (lambda(c)==0 && N(c)>0)`

  which means that either `lambda(c)` is nonzero and `N(n)` is zero, or the other way around. If for some c, $lambda(c) \neq 0$ and $N(c) \neq 0$, an error is reported and this function aborts.

S
: `S(c,k)` is the mean service time for class c customers on service center k, `S(c,k)` $\geq$ `0`. For FCFS nodes, service times must be class-independent.

V
: `V(c,k)` is the average number of visits of class c customers to service center k (`V(c,k)` $\geq$ `0`).

m
: `m(k)` is the number of servers at service center k. Only single-server (`m(k)==1`) or IS (Infinite Server) nodes (`m(k)<1`) are supported. If omitted, each service center is assumed to have a single server. Queueing discipline for single-server nodes can be FCFS, PS or LCFS-PR.

OUTPUTS

U
: `U(c,k)` is the utilization of class c requests on service center k.

R
: `R(c,k)` is the response time of class c requests on service center k.

Q
: `Q(c,k)` is the average number of class c requests on service center k.

X
: `X(c,k)` is the class c throughput on service center k.

See also: qnclosedmultimva, qnopenmulti.

REFERENCES

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 7.4.3 ("Mixed Model Solution Techniques"). Note that in this function we compute the mean response time R instead of the mean residence time as in the reference.

Herb Schwetman, *Implementing the Mean Value Algorithm for the Solution of Queueing Network Models*, Technical Report CSD-TR-355, Department of Computer Sciences, Purdue University, feb 15, 1982, available at http://www.cs.purdue.edu/research/technical_reports/1980/TR%2080-355.pdf

6.4 Algorithms for non Product-Form QN

`[U, R, Q, X] = qnmvablo (K, S, M, P)` [Function File]

MVA algorithm for closed queueing networks with blocking. `qnmvablo` computes approximate utilization, response time and mean queue length for closed, single class queueing networks with blocking.

INPUTS

K population size, i.e., number of requests in the system. K must be strictly greater than zero, and less than the overall network capacity: `K < sum(M)`.

S Average service time. `S(i)` is the average service time requested on server i (`S(i) > 0`).

M Server capacity. `M(i)` is the capacity of service center i. The capacity is the maximum number of requests in a service center, including the request currently in service (`M(i) > 1`).

P `P(i,j)` is the probability that a request which completes service at server i will be transferred to server j.

OUTPUTS

U `U(i)` is the utilization of service center i.

R `R(i)` is the average response time of service center i.

Q `Q(i)` is the average number of requests in service center i (including the request in service).

X `X(i)` is the throughput of service center i.

See also: qnopen, qnclosed.

REFERENCES

Ian F. Akyildiz, *Mean Value Analysis for Blocking Queueing Networks*, IEEE Transactions on Software Engineering, vol. 14, n. 2, april 1988, pp. 418–428. http://dx.doi.org/10.1109/32.4663

`[U, R, Q, X] = qnmarkov (lambda, S, C, P)` [Function File]
`[U, R, Q, X] = qnmarkov (lambda, S, C, P, m)` [Function File]
`[U, R, Q, X] = qnmarkov (N, S, C, P)` [Function File]
`[U, R, Q, X] = qnmarkov (N, S, C, P, m)` [Function File]

Compute utilization, response time, average queue length and throughput for open or closed queueing networks with finite capacity. Blocking type is Repetitive-Service

(RS). This function explicitly generates and solve the underlying Markov chain, and thus might require a large amount of memory.

More specifically, networks which can me analyzed by this function have the following properties:

- There exists only a single class of customers.
- The network has K service centers. Center i has $m_i > 0$ servers, and has a total (finite) capacity of $C_i \geq m_i$ which includes both buffer space and servers. The buffer space at service center i is therefore $C_i - m_i$.
- The network can be open, with external arrival rate to center i equal to λ_i, or closed with fixed population size N. For closed networks, the population size N must be strictly less than the network capacity: $N < \sum_i C_i$.
- Average service times are load-independent.
- P_{ij} is the probability that requests completing execution at center i are transferred to center j, $i \neq j$. For open networks, a request may leave the system from any node i with probability $1 - \sum_j P_{ij}$.
- Blocking type is Repetitive-Service (RS). Service center j is *saturated* if the number of requests is equal to its capacity `C_j`. Under the RS blocking discipline, a request completing service at center i which is being transferred to a saturated server j is put back at the end of the queue of i and will receive service again. Center i then processes the next request in queue. External arrivals to a saturated servers are dropped.

INPUTS

lambda
N
: If the first argument is a vector *lambda*, it is considered to be the external arrival rate `lambda(i)` $\geq$ `0` to service center i of an open network. If the first argument is a scalar, it is considered as the population size N of a closed network; in this case N must be strictly less than the network capacity: `N < sum(C)`.

S
: `S(i)` is the average service time at service center i

C
: `C(i)` is the Capacity of service center i. The capacity includes both the buffer and server space `m(i)`. Thus the buffer space is `C(i)-m(i)`.

P
: `P(i,j)` is the transition probability from service center i to service center j.

m
: `m(i)` is the number of servers at service center i. Note that `m(i)` $\geq$ `C(i)` for each *i*. If *m* is omitted, all service centers are assumed to have a single server (`m(i) = 1` for all i).

OUTPUTS

U
: `U(i)` is the utilization of service center i.

R
: `R(i)` is the response time on service center i.

Q
: `Q(i)` is the average number of customers in the service center i, *including* the request in service.

X `X(i)` is the throughput of service center i.

Note:

The space complexity of this implementation is $O(\prod_{i=1}^{K}(C_i + 1)^2)$. The time complexity is dominated by the time needed to solve a linear system with $\prod_{i=1}^{K}(C_i + 1)$ unknowns.

6.5 Bounds on performance

`[Xu, Rl] = qnopenab (lambda, D)` [Function File]

Compute Asymptotic Bounds for single-class, open Queueing Networks with K service centers.

INPUTS

lambda overall arrival rate to the system (scalar). Abort if `lambda` $\leq$ `0`

D `D(k)` is the service demand of service center k. Abort if `D(k) < 0` for any k

OUTPUTS

Xu Upper bound on the system throughput.

Rl Lower bound on the system response time.

See also: qnopenbsb.

REFERENCES

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 5.2 ("Asymptotic Bounds").

`[Xl, Xu, Rl, Ru] = qnclosedab (N, D)` [Function File]

`[Xl, Xu, Rl, Ru] = qnclosedab (N, D, Z)` [Function File]

Compute Asymptotic Bounds for single-class, closed Queueing Networks with K service centers.

INPUTS

N number of requests in the system (scalar, `N>0`).

D `D(k)` is the service demand of service center k, `D(k)` $\geq$ `0`.

Z external delay (think time, scalar, `Z` $\geq$ `0`). If omitted, it is assumed to be zero.

OUTPUTS

Xl
Xu Lower and upper bound on the system throughput.

Rl
Ru Lower and upper bound on the system response time.

See also: qnclosedbsb, qnclosedgb, qnclosedpb.

REFERENCES

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 5.2 ("Asymptotic Bounds").

`[Xu, Rl, Ru] = qnopenbsb (lambda, D)` [Function File]

Compute Balanced System Bounds for single-class, open Queueing Networks with K service centers.

INPUTS

lambda overall arrival rate to the system (scalar). Abort if `lambda < 0`

D `D(k)` is the service demand of service center k. Abort if `D(k) < 0` for any k.

OUTPUTS

Xl Lower bound on the system throughput.

Rl
Ru Lower and upper bound on the system response time.

See also: qnopenab.

REFERENCES

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, 1984. http://www.cs.washington.edu/homes/lazowska/qsp/. In particular, see section 5.4 ("Balanced Systems Bounds").

`[Xl, Xu, Rl, Ru] = qnclosedbsb (N, D)` [Function File]
`[Xl, Xu, Rl, Ru] = qnclosedbsb (N, D, Z)` [Function File]

Compute Balanced System Bounds for single-class, closed Queueing Networks with K service centers.

INPUTS

N number of requests in the system (scalar).

D `D(k)` is the service demand of service center k. Abort if `K(k) < 0` for any k.

Z external delay (think time, scalar) experienced by requests. If omitted, it is assumed to be zero.

OUTPUTS

Xl
Xu Lower and upper bound on the system throughput.

Rl
Ru Lower and upper bound on the system response time.

See also: qnclosedab, qnclosedgb, qnclosedpb.

`[Xl, Xu] = qnclosedpb (N, D )` [Function File]

Compute PB Bounds for single-class, closed Queueing Networks with K service centers.

INPUTS

N number of requests in the system (scalar). Must be `N > 0`.

D `D(k)` is the service demand of service center k. Must be `D(k)` $\geq$ `0` for all k.

Z external delay (think time, scalar). If omitted, it is assumed to be zero. Must be `Z` $\geq$ `0`.

OUTPUTS

Xl

Xu Lower and upper bound on the system throughput.

See also: qnclosedab, qbclosedbsb, qnclosedgb.

REFERENCES

The original paper describing PB Bounds is C. H. Hsieh and S. Lam, *Two classes of performance bounds for closed queueing networks*, PEVA, vol. 7, no. 1, pp. 3–30, 1987

This function implements the non-iterative variant described in G. Casale, R. R. Muntz, G. Serazzi, *Geometric Bounds: a Non-Iterative Analysis Technique for Closed Queueing Networks*, IEEE Transactions on Computers, 57(6):780-794, June 2008.

`[Xl, Xu, Ql, Qu] = qnclosedgb (N, D, Z)` [Function File]

Compute Geometric Bounds (GB) for single-class, closed Queueing Networks.

INPUTS

N number of requests in the system (scalar, `N > 0`).

D `D(k)` is the service demand of service center k (`D(k)` $\geq$ `0`).

Z external delay (think time, scalar). If omitted, it is assumed to be zero.

OUTPUTS

Xl

Xu Lower and upper bound on the system throughput. If `Z>0`, these bounds are computed using *Geometric Square-root Bounds* (GSB). If `Z==0`, these bounds are computed using *Geometric Bounds* (GB)

Ql

Qu `Ql(i)` and `Qu(i)` are the lower and upper bounds respectively of the queue length for service center i.

See also: qnclosedab.

REFERENCES

G. Casale, R. R. Muntz, G. Serazzi, *Geometric Bounds: a Non-Iterative Analysis Technique for Closed Queueing Networks*, IEEE Transactions on Computers, 57(6):780-794, June 2008. http://doi.ieeecomputersociety.org/10.1109/TC.2008.37

In this implementation we set X^+ and X^- as the upper and lower Asymptotic Bounds as computed by the `qnclosedab` function, respectively.

6.6 Utility functions

6.6.1 Open or closed networks

[U, R, Q, X] = qnclosed (*N*, *S*, *V*, ...) [Function File]

MVA algorithm for closed queueing networks. The network can be made of fixed-capacity centers or delay centers; load-dependent service centers are also supported. *N* is the population (number of requests) in the system.

- If *N* is a scalar, the exact MVA algorithm for single-class closed networks is used. If *S* is a vector, then *S(k)* is the average service time of service center k, and this function uses the `qnclosedsinglemva` algorithm for load-independent service centers. If *S* is a matrix, *S(k,i)* is the average service time at service center k when $i > 0$ jobs are present; in this case, this function uses the `qnclosedsinglemvald` function for load-dependent service centers.
- If *N* is a vector, the exact MVA algorithm for multiclass networks is used, as implemented by the `qnclosedmultimva` function.

See also: qnclosedsinglemva, qnclosedsinglemvald, qnclosedmultimva.

EXAMPLE

```
P = [0 0.3 0.7; 1 0 0; 1 0 0]; # Transition probability matrix
S = [1 0.6 0.2]; # Average service times
m = ones(1,3); # All centers are single-server
Z = 2; # External delay
Nmax = 15; # Maximum population to consider

V = qnvisits(P); # Compute number of visits from P
D = V .* S; # Compute service demand from S and V
X_bsb_lower = X_bsb_upper = zeros(1,Nmax);
X_ab_lower = X_ab_upper = zeros(1,Nmax);
X_mva = zeros(1,Nmax);
for n=1:Nmax
  [X_bsb_lower(n) X_bsb_upper(n)] = qnclosedbsb(n, D, Z);
  [X_ab_lower(n) X_ab_upper(n)] = qnclosedab(n, D, Z);
  [U R Q X] = qnclosed( n, S, V, m, Z );
  X_mva(n) = X(1)/V(1);
endfor
close all;
plot(1:Nmax, X_ab_lower,"g;Asymptotic Bounds;", \
     1:Nmax, X_bsb_lower,"k;Balanced System Bounds;", \
     1:Nmax, X_mva,"b;MVA;", "linewidth", 2, \
     1:Nmax, X_bsb_upper,"k", \
     1:Nmax, X_ab_upper,"g" );
title("MVA vs Balanced System Bounds");
axis([1,Nmax,0,1]);
xlabel("Request Population Size N");
ylabel("System Throughput X(N)");
legend("location","southeast");
```

`[U, R, Q, X] = qnopen (lambda, S, V, ...)` [Function File]

Compute utilization, response time, average number of requests in the system, and throughput for open queueing networks. If *lambda* is a scalar, the network is considered a single-class QN and is solved using `qnopensingle`. If *lambda* is a vector, the network is considered as a multiclass QN and solved using `qnopenmulti`.

See also: qnopensingle, qnopenmulti.

6.6.2 Computation of the visit counts

For closed, single-class networks the number of visits satisfies the following conditions:

$V_j = \sum_{i=1}^{N} V_i P_{ij}, \; V_1 = 1$

while for open networks with arrival rates `lambda(i)`:

$V_j = \lambda_j + \sum_{i=1}^{N} V_i P_{ij}$

where N is the dimension (number of rows or columns) of P.

`V = qnvisits (P)` [Function File]
`V = qnvisits (P, lambda)` [Function File]

Compute the average number of visits to the service centers of a single class, open or closed Queueing Network with N service centers.

INPUTS

P `P(i,j)` is the probability that a request which completes service at service center i will be transferred to server j. It must hold that `all(all(P ≥ 0))`. For closed networks additionally it must hold that `sum(P,2)==1`.

lambda (for open networks only) external arrival vector. `lambda(i)` is the external arrival rate to service center i. If this parameter is omitted, the network is assumed to be closed.

OUTPUTS

V `V(i)` is the average number of visits to server i.

EXAMPLE

```
P = [ 0 0.4 0.6 0; \
      0.2 0 0.2 0.6; \
      0 0 0 1; \
      0 0 0 0 ];
lambda = [0.1 0 0 0.3];
V = qnvisits(P,lambda);
S = [2 1 2 1.8];
m = [3 1 1 2];
[U R Q X] = qnopensingle( sum(lambda), S, V, m );
```

6.6.3 Other utility functions

`pop_mix = population_mix (k, N)` [Function File]

Return the set of valid population mixes with exactly k customers, for a closed multiclass Queueing Network with population vector N. More specifically, given a multiclass

Queueing Network with C customer classes, such that there are *N(i)* requests of class i, a k-mix *mix* is a C-dimensional vector with the following properties:

```
all( mix >= 0 );
all( mix <= N );
sum( mix ) == k;
```

This function enumerates all valid k-mixes, such that *pop_mix(i)* is a C dimensional row vector representing a valid population mix, for all i.

INPUTS

k Total population size (scalar) of the requested mix.

N *N(i)* is the number of class i requests. It must be $k \leq$ *sum(N)*.

OUTPUTS

pop_mix *pop_mix(i,j)* is the number of class j requests in the i-th population mix. The number of population mixes is `rows(` *pop_mix* `)` .

Note that if you are interested in the number of k-mixes and you don't care to enumerate them, you can use the funcion `qnmvapop`.

See also: qnmvapop.

REFERENCES

Herb Schwetman, *Implementing the Mean Value Algorithm for the Solution of Queueing Network Models*, Technical Report CSD-TR-355, Department of Computer Sciences, Purdue University, feb 15, 1982, available at http://www.cs.purdue.edu/research/technical_reports/1980/TR 80-355.pdf

Note that the slightly different problem of generating all tuples $k_1, k_2, \ldots k_N$ such that $\sum_i k_i = k$ for some fixed k has been described in S. Santini, *Computing the Indices for a Complex Summation*, unpublished report, available at http://arantxa.ii.uam.es/~ssantini/writing/notes/s668_summation.pdf

H = **qnmvapop** (*N*) [Function File]

Given a network with C customer classes, this function computes the number of valid population mixes *H(r,n)* that can be constructed by the multiclass MVA algorithm by allocating n customers to the first r classes.

INPUTS

N Population vector. *N(c)* is the number of class-c requests in the system. The total number of requests in the network is `sum(`*N*`)`.

OUTPUTS

H *H(r,n)* is the number of valid populations that can be constructed allocating n customers to the first r classes.

See also: qnclosedmultimva,population_mix.

REFERENCES

Zahorjan, J. and Wong, E. *The solution of separable queueing network models using mean value analysis*. SIGMETRICS Perform. Eval. Rev. 10, 3 (Sep. 1981), 80-85. DOI http://doi.acm.org/10.1145/1010629.805477

Appendix A Contributing Guidelines

Contributions and bug reports are *always* welcome. If you want to contribute to the `qnetworks` package, here are some guidelines you should consider:

- If you are contributing a new function, please embed proper documentation within the function itself. The documentation must be in `texinfo` format, so that it will be extracted and formatted into the printable manual. See the existing functions of the `qnetworks` package for the documentation style.
- The documentation should be as precise as possible. In particular, always state what the valid ranges of the parameters are.
- If you are contributing a new function, ensure that the function properly checks the validity of its input parameters. For example, each function accepting vectors should check whether the dimensions match.
- Always provide bibliographic references for each algorithm you contribute. If your implementation differs in some way from the reference you give, please describe how and why your implementation differs.
- Include Octave test and demo blocks with your code. Test blocks are particularly important, because Queueing Network algorithms tend to be quite complex to implement correctly, and we must ensure that the implementations provided with the `qnetworks` package are (mostly) correct.

Send your contribution to Moreno Marzolla (`marzolla@cs.unibo.it`).

Appendix B Acknowledgements

The following people (listed in alphabetical order) contributed to the `qnetworks` package, either by providing feedback, reporting bugs or contributing code: Philip Carinhas, Dmitry Kolesnikov.

Appendix C GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Concept Index

Function Index

C

D

P

Q

Author Index