University of Würzburg
Institute of Computer Science
Research Report Series

A Toolkit of Octave Functions for Discrete–Time Analysis of Queuing Systems

Notker Gerlich

Report No. 128 December 95

Institute of Computer Science, University of Würzburg
Am Hubland, D-97074 Würzburg, FRG
Tel.: +49 931 888-5513
E-mail: gerlich@informatik.uni-wuerzburg.de

The modularity of the Discrete-Time Analysis (DTA) technique calls for a toolkit consisting of all the operators involved in order to easily implement the algorithms on a computer using object-oriented language. This paper presents a toolkit of DTA operators written in the Octave language. The use of the toolkit is demonstrated by the numerical analysis of the discrete-time $GI/GI/1$ queuing system and the recently often employed discrete-time $GI/GI/1$ system with bounded delay.

1 Introduction

The Discrete-Time Analysis (DTA) technique as presented by Ackroyd (1980) and Tran-Gia (1986, 1989) is a modular modelling paradigm and analysis technique for the numerical analysis of single stage queuing systems. Since DTA aims at numerical analysis there is a need to support the implementation of the algorithms on a computer that is as easy to understand and straightforward as the DTA modelling itself and that invites to experiment with.

The modularity, which is expressed by the fact that the development of discrete random variables is described by using a set of operators, calls for a toolkit of the operators and easy-to-learn, straightforward control structures (loops etc.) to combine the operators into algorithms. An efficient computation of some of the operators involves employing numerical techniques like the *Fast Fourier Transform* (FFT). Thus, the need for powerful numerical software requires interfaces to standard numerical procedures. Finally, input and output functions should be provided to present the final results as well as intermediate results. Thus, a certain capability for interactiveness is required.

The software package Octave provides many of these features for the analyst. Octave is an easy to learn, high-level interactive language for numerical computations. It was developed by John W. Eaton at the University of Texas at Austin (Eaton 1995) and has been used there for teaching linear algebra, differential equations and chemical reactor design. Octave allows elegant formulation of algorithms hiding low level details like memory allocation etc., provides a fairly comfortable interface to powerful numerical software of the celebrated *netlib*, interprets high-level control structures (conditions, loops etc.), and is able to produce graphical output via Gnuplot. The Octave software is free software in terms of the GNU General Public License and is running on various Unix platforms (SunOS, DEC OSF/1, NeXT, AIX etc.).[1] For details we refer the reader to the Octave manual (Eaton 1995).

This tutorial is organized as follows. Section 2 presents the DTA of the general discrete-time $GI/GI/1$ queue as an example to demonstrate the use of the toolkit. The section is split into four parts. The first part recalls the DTA modelling steps leading to the DTA basic equation. The second part shows the implementation of this basic equation using the toolkit. The third part is dedicated to an interactive use of Octave in order to get an visualisation of the algorithm. In part four the implementation of

[1] Octave can be obtained via anonymous ftp from the ftp-server at URL `ftp://ftp.che.utexas.edu/pub/octave` as well as from many other ftp-sites. Additional information about Octave can be found on the Octave homepage located at URL `http://bevo.che.wisc.edu/octave.html` in the WWW.

the DTA analysis of the discrete-time $GI/GI/1$ with bounded delay is demonstrated. A reference manual of the toolkit is included as an appendix.

2 Example: The Discrete-Time GI/GI/1 Queue

2.1 Basic Equation[2]

The subject of our study is the discrete-time $GI/GI/1$ queuing system with infinite waiting room. The time interval between the consecutive arrivals of two customers is described in terms of a discrete probability mass function (PMF) $a_n(k)$: $a_n(k)$ is the probability to have an interval of an integer number of k time units between the arrival of customer number n and customer number $n+1$. The service time of customer n is given in terms of a discrete probability mass function $b_n(k)$. The interarrival times and the service times are i.i.d. random variables. There is one single server and a infinite waiting room. The customers are served in the order of their arrival (first-come-first-served, FCFS).

The sum of the service times of the customers waiting for service and the remaining service time of the customer currently being served is termed the *unfinished work*. The service time distribution is discrete, and so is the distribution of the unfinished work. We denote the PMF of the unfinished work by $u(k)$. At the instant of arrival of customer n the unfinished work is increased by the service time of customer n. If the server is busy the unfinished work is decreased by one (discrete) work unit per (discrete) time unit (cf. Fig. 1).

In the following the random variables of the interarrival time, service time etc. are denoted by uppercase letters corresponding to the lowercase letters of the PMF's, e.g. B_n denotes the service time random variable of customer n. Indicating '*just prior to the arrival instant*' by adding a superscript '$-$' and '*just after the arrival instant*' by adding a superscript '$+$' to the random variables and PMF's respectively, we obtain the following relations for the development of the unfinished work U:

$$\begin{aligned} U_n^+ &= U_n^- + B_n, \\ U_{n+1}^- &= \max\{U_n^+ - A_n, 0\}. \end{aligned}$$

The previous equation results from the fact that the unfinished work is decreased by one work unit per time unit, i.e. during A_n time units U_n is diminished by A_n work units.

[2] We follow the derivation of Ackroyd (1980) and Tran-Gia (1986). A comprehensive treatment can be found in Tran-Gia (1989).

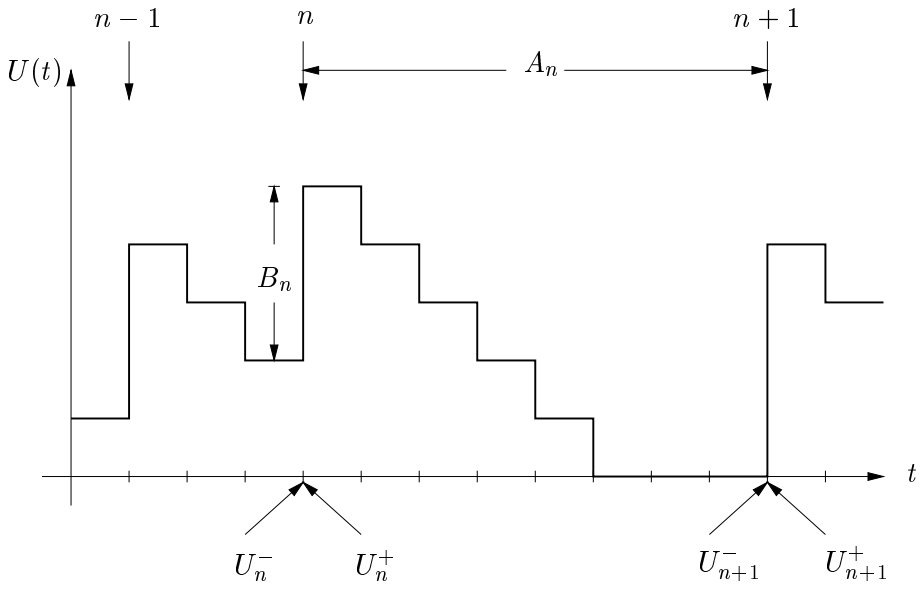

Figure 1: Unfinished work process of the $GI/GI/1$ system.

Combining the previous two equations the following recursive equation is obtained:

$$U_{n+1}^- \;=\; \max\{U_n^- + B_n - A_n, 0\}.$$

Going from random variables to the corresponding PMF's this equation becomes:

$$u_{n+1}^-(k) \;=\; \pi_0[u_n^-(k) \circledast b_n(k) \circledast a_n(-k)].$$

Here, the linear operator $\pi_0[\cdot]$ "sweep[s] the probability in the negative half-line up to the origin" (Kleinrock 1976, Ch. 2.6):

$$\pi_0[z(k)] \;=\; \begin{cases} 0 & \text{for } k < 0, \\ \displaystyle\sum_{i=-\infty}^{0} z(i) & \text{for } k = 0, \\ z(k) & \text{for } k > 0. \end{cases}$$

The operator '$\circledast$' denotes the discrete convolution. Defining the *system function* $c_n(k)$ by the cross-correlation of $a_n(k)$ and $b_n(k)$, i.e.

$$c_n(k) \;=\; b_n(k) \circledast a_n(-k),$$

we obtain

$$u_{n+1}^-(k) \;=\; \pi_0 \left[u_n^-(k) \circledast c_n(k)\right].$$

Observing the process of the unfinished work at arrival instants only, the unfinished work just prior to the arrival instant of a customer is equal to this customer's waiting time (Kleinrock 1975). Denoting the (discrete) PMF of the waiting time distribution of customer number n by $w_n(k)$ we obtain the final result:

$$w_{n+1}(k) \;=\; \pi_0 \left[w_n(k) \circledast c_n(k) \right].$$

The recursive formula just obtained can be viewed as the program to compute the waiting time distribution of the $GI/GI/1$ queuing system iteratively. Before we proceed to an algorithmic formulation of our solution we would like to add a few remarks.

The recursive scheme describes the *non-stationary behaviour* of the system even with interarrival time and service time PMF's changing on a customer's basis. The *equilibrium* distribution is obtained by iterating until convergence is achieved, to within an appropriate criterion.

The DTA can also be used to approximate the *continuous*-time $GI/GI/1$ queuing system. To do this the continuous-time distribution functions are approximated by suitably chosen discrete ones. Our Octave toolkit provides such approximations for the most common continuous distribution functions.

2.2 Time-Domain Algorithm and Octave Code

Fig. 2 shows a graphical representation of the algorithm just derived.

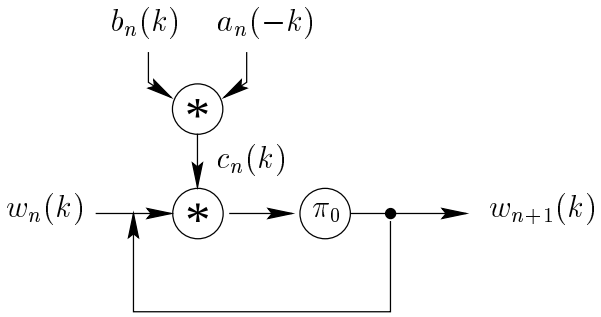

Figure 2: Computational diagram of the time-domain algorithm

Translated into an abstract high-level programming language the algorithm may look like the left side of the table below. From here, it is only a small step finally to arrive at the Octave coding of the algorithm as shown on the right side.[3]

```
1  funct WaitingTime(a, b)
2
3     c(k) ← b(k) ⊛ a(−k);
4     w(0) ← 1;
5
6     EW = w̄;
7     w(k) ← π0[w(k) ⊛ c(k)];
8     while |w̄ − EW| > ε do
9           EW = w̄;
10          w(k) ← π0[w(k) ⊛ c(k)];
11    od
12 end
```

```
function w = WaitingTime (a, b)
  global EPSILON;
  c = XCorr (b, a);
  w = Distribution (1);

  EW = Moment (w,1);
  w = PiUp (Conv (w,c), 0);
  while (abs (Moment (w, 1) - EW) > EPSILON)
    EW = Moment (w,1);
    w = PiUp (Conv (w,c), 0);
  endwhile
endfunction
```

In line *2* a global Variable `EPSILON` is made known within the scope of the function. This variable will be used later as the precision of the convergence criterion. Using a global variable here allows an overall precision of all operators without having to accept unhandy function interfaces. Having declared `EPSILON` the system function $c(k)$ is calculated by cross-correlating the service time PMF $b(k)$ with the interarrival time PMF $a(k)$ (*l.3*). In Octave this operation is carried out by the DTA toolkit function `XCorr()`. This operation is rather computationally expensive if carried out in the time domain. Fortunately, the discrete convolution may be efficiently performed in the frequency domain (Oppenheim and Schafer 1989): The *Discrete Fourier Transform* (DFT) of each probability vector is computed via the *Fast Fourier Transform* (FFT) and the DFT's are multiplied point-by-point; finally, the inverse DFT of the product is computed again using the FFT. The DTA Octave functions are split into two libraries (*script files* in Octave's terminolgy), `DDist.m` and `DDistOp.m`.[4] The latter defines operators on PMF's like `XCorr()`; the former contains functions to define PMF's. `Distribution(1)` (*l.4*) defines the initial waiting time PMF. Since the first customer finds an empty system upon arrival its waiting time is zero with probability one, i.e. $w(0) = 1$ and $w(i) = 0$ if $i \neq 0$.

[3] For simplicity, we restrict ourselves to the case of the interarrival time and service time distributions remaining constant. The algorithm of the general case is very similar to the one shown: the computation of the system function $c(k)$ has to be moved into the iteration loop.

[4] A reference manual of the toolkit functions is included in the appendix.

Missing a post-conditioned loop in Octave, we have to prepare for the pre-conditioned `while` statement. This is done by performing one iteration step outside the loop (*l.6–7* cf. *l.9–10*). We have chosen to use the difference of the mean waiting time $\bar{w}$ — the first moment of the waiting time PMF — before and after one iteration step as convergence criterion; there are several other criteria possible.

Now, once we have coded the algorithm in Octave, we can use it eg. to produce plots of the waiting time distribution of $GI/GI/1$ systems.

In the program shown in Fig. 4 the function `WaitingTime()` is used to create Fig. 3. The plot shows the complementary waiting time distribution functions of a *NEGBIN*$/D/1$ system with coefficients of variation of the interarrival time distribution $c = 0.5, 1.0$, and 1.5. It takes about 30 seconds on a SUN SPARC 20/612 workstation to compute the data and create the plot.

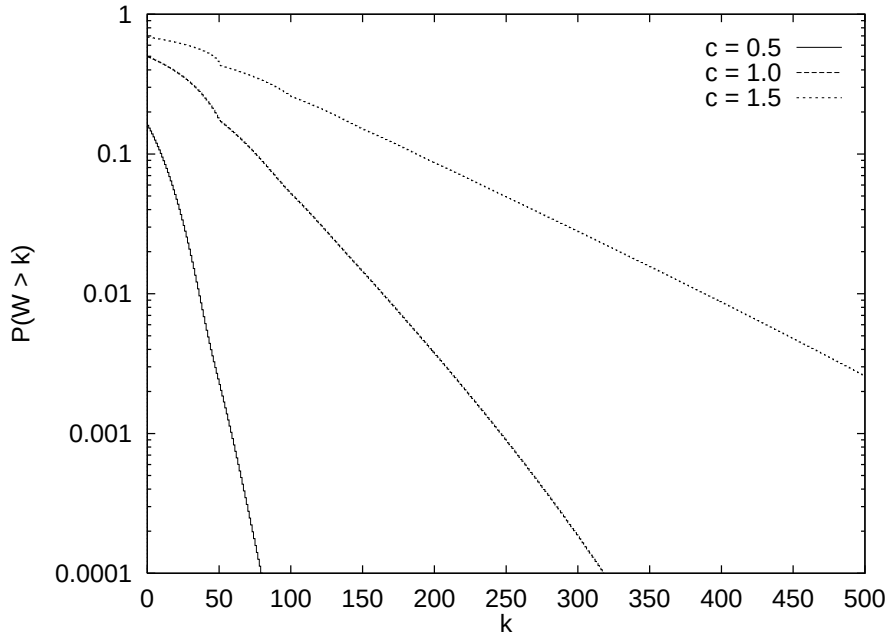

Figure 3: Complementary waiting time distribution functions of *NEGBIN/D/1* systems

Programs like this are well suited to produce equilibrium results. But since the successive PMF's arising out of each iteration have an interpretation of their own as the PMF's observed by the customers arriving successively, these intermediate results are of interest. In the next section we will make use of Octave's interactive features to watch the waiting time distribution of an example $GI/GI/1$ system develop in each iteration (the example is taken from Tran-Gia 1989).

```
#! /opt/bin/octave -qf
#
# create plot of the complementary waiting time distribution function of a
# NEGBIN/D/1 queuing system. The plots are computed with the NEGBIN distribution having
# coefficients of variation of 0.5, 0.1, and 1.5.
# Reference: Tran-Gia (1989)

DDist;                                                  # load the toolkit
DDistOp;

rho = 0.5;                                              # work load
EA = 100;                                               # interarrival time mean
EB = rho * EA;                                          # service time mean
global EPSILON = 1e-1;                                  # precisision

function w = WaitingTime (a, b)
    ⋮                                                   # function definition
endfunction                                             # see above

                                                        # compute the PMF's
e = WaitingTime (NegBin (EA, 0.5), Deterministic (EB));
m = WaitingTime (NegBin (EA, 1.0), Deterministic (EB));
h = WaitingTime (NegBin (EA, 1.5), Deterministic (EB));

                                                        # prepare the plot data
el = dplot (CPDF (Normalize (e)));
ml = dplot (CPDF (Normalize (m)));
hl = dplot (CPDF (Normalize (h)));

                                                        # create the plot
set term postscript;                                    # output will be PS
set output "gg1.ps";                                    # name the PS-file
set xlabel "k";
set ylabel "P(W > k)";
set xrange [0:500];                                     # set ranges
set yrange [1e-4:1];
set logscale y;                                         # logarithmic plot
gplot el title "c = 0.5" with lines, \
      ml title "c = 1.0" with lines, \
      hl title "c = 1.5" with lines;
```

Figure 4: Octave program using the function `WaitingTime()`

2.3 Interactive Visualization

First of all we have to invoke Octave and load the DTA library functions:

```
octave:1> DDist; DDistOp;
```

Now that the functions of the toolkit are known to the system we may use them to define the interarrival time and service time distributions $a(k)$ and $b(k)$, respectively. In our example we have:

$$a(2) = \tfrac{25}{72}, \quad a(5) = \tfrac{22}{72}, \quad a(8) = \tfrac{25}{72}, \quad a(k) = 0 \quad \text{else},$$
$$b(1) = \tfrac{1}{2}, \quad b(2) = \tfrac{1}{3}, \quad b(8) = \tfrac{1}{6}, \quad a(k) = 0 \quad \text{else}.$$

In Octave this reads:

```
octave:2> a = Distribution ([0; 0; 25/72; 0; 0; 22/72; 0; 0; 25/72]);
octave:3> b = Distribution ([0; 1/2; 1/3; 0; 0; 0; 0; 0; 1/6]);
```

As in the previous section we compute the system function and initialize the waiting time PMF:

```
octave:4> c = XCorr (b, a);
octave:5> w = Distribution (1);
```

To complete our preparations, we define the detail of the graphical output:

```
octave:6> set xrange [0:20]; set yrange [1e-3:1]; set logscale y;
```

Now, we perform the first iteration and plot the complementary distribution function of the waiting time distribution:

```
octave:7> w = PiUp (Conv (w,c), 0); dplot (CPDF (w));
```

By repeating the previous two commands[5] we can watch the successive development of the waiting time PMF. Fig. 5 shows the resulting waiting time PMF's of the first 8 iterations. It should be noted once again that the waiting time distribution function obtained in iteration number i corresponds to the waiting time experienced by the $(i+1)$th customer. Hence, the successive waiting time PMF's represent the development of the waiting time distribution under the non-equilibrium conditions of the initial transient phase conditioned on an empty system upon arrival of customer number 1.

[5] This can be easily achieved by pressing `<Ctrl-P>`.

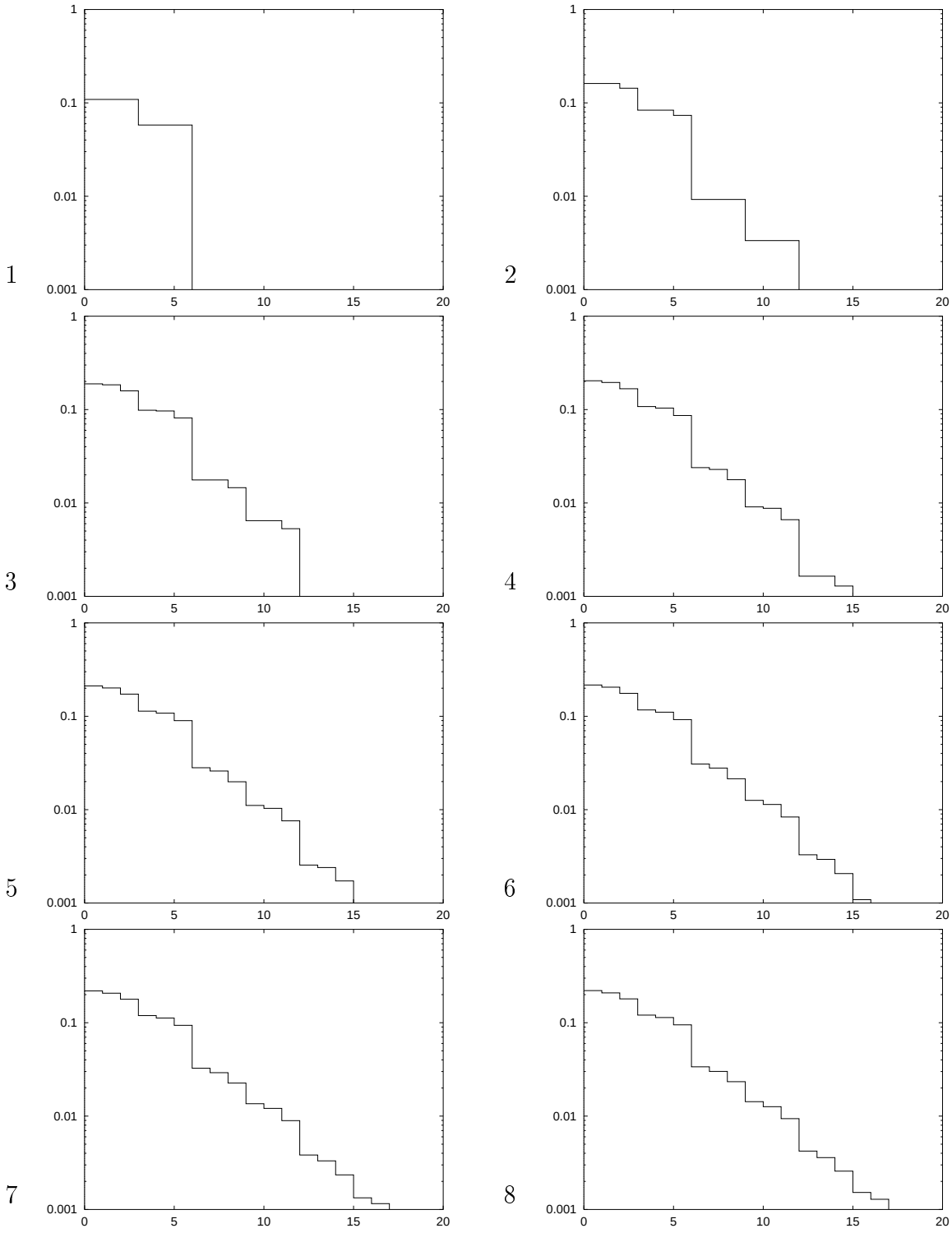

Figure 5: Development of the complementary waiting time distribution function

After 20 iterations the changes in the distribution become invisible to the naked eye. Thus, the number of iterations required for convergence depends on the convergence criterion used. It also depends on the queuing problem. "Generally, the greater the utilisation factor of the queue, the more iterations will be required for convergence, reflecting the fact that more customers must arrive for the queue to approach equilibrium" (Ackroyd 1980).

2.4 The Discrete-Time GI/GI/1-Queue with Bounded Delay

In this section we consider another example, the discrete-time $GI/GI/1$ system with bounded delay. In this system the waiting time of customers is limited to a maximum of L time units, i.e. customers who arrive and would wait longer than $L-1$ time units are rejected.

A derivation similar to that of Section 2.1 leads to the following computational diagram (for details the reader is referred to Tran-Gia 1993):

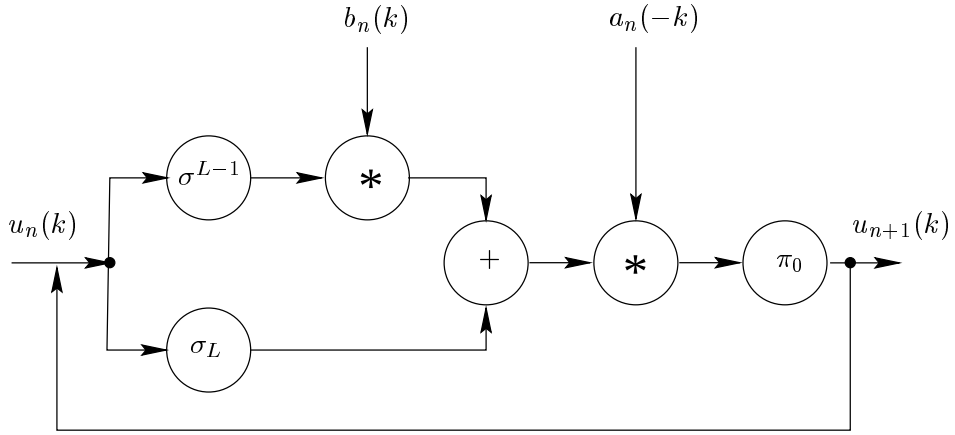

Figure 6: Computational diagram.

Here $\sigma^m[z(k)]$ and $\sigma_m[z(k)]$ are new operators which truncate parts of the PMF $z(k)$:

$$\sigma^m[z(k)] = \begin{cases} z(k) & \text{for } k \leq m, \\ 0 & \text{for } k > m, \end{cases}$$

$$\sigma_m[z(k)] = \begin{cases} 0 & \text{for } k < m, \\ z(k) & \text{for } k \geq m. \end{cases}$$

The customer rejection probability in steady-state is given by:

$$B = \sum_{i=L}^{\infty} u(i).$$

Translating the computational diagram into an algorithm we obtain:

```
1  funct WorkLoad(a, b)
2
3     u(0) ← 1;
4
5     EU = ū;
6     u(k) ← π0[(σ^{L−1}[u(k)] ⊛ b(k) + σL[u(k)]) ⊛ a(−k)];
7     while |ū − EU| > ε do
8           EU = ū;
9           u(k) ← π0[(σ^{L−1}[u(k)] ⊛ b(k) + σL[u(k)]) ⊛ a(−k)];
10    od
11 end
```

Again the coding in Octave is rather straightforward:

```
1  function u = WorkLoad (a, b)
2    global EPSILON;
3    u = Distribution (1);
4
5    EU = Moment (u,1);
6    u = PiUp (XCorr (Plus (Conv (SigmaDown(u, L-1), b), SigmaUp(u, L) ), a), 0);
7    while (abs (Moment (u, 1) - EU) > EPSILON)
8     EU = Moment (u,1);
9     u = PiUp (XCorr (Plus (Conv (SigmaDown(u, L-1), b), SigmaUp(u, L) ), a), 0);
10   endwhile
11 endfunction
```

Now, the rejection probability B is computed by:

```
u = WorkLoad (a, b);
B = sum (u.v(L+1:length (u.v));
```

What starts to become visible here is that the readability of the code suffers from the notation of the operators as functions when the algorithm becomes more complex. The infix-notation of the operators '$\circledast$' and '$+$' make the expression of line *6* of the algorithm in abstract high-level language more readable than its equivalent in Octave. The problem mainly appears, when expressions are nested like in line *6*. Since the prefix-notation is given by Octave only a decomposition of the expression into several lines of code may be helpful.

3 Conclusion and Outlook

We have shown that the modularity of the DTA approach can be exploited to support the implementation of the algorithms by a toolkit of the operators. The high-level language Octave is particularly appropriate for both implementing the toolkit and the algorithms using the toolkit. The coding in Octave is as easy to understand and straightforward as the DTA modelling itself. Furthermore, the interactiveness of Octave invites to experimentation.

Since Octave spares the analyst low-level programming detail like memory allocation etc. the implementation of new operators can easily be done even by an unexperienced programmer. Thus, more complicated discrete-time systems can be easily analysed e.g. $GEOM/D/n$ or $GEOM/D/1-S$.

The readability of the coding of more complex algorithms suffers from the prefix-notation of the operators given by Octave. Here, an infix-notation would help. But this would require a programming language equally powerful as Octave with the capability to define one's own infix-operators, which to the best of the authors knowledge is not available for the time being.

Acknowledgement

The author would like to thank Prof. P. Tran-Gia for encouraging him to write this paper and for reviewing the manuscript. The author would also like to thank Christoph Kern, who introduced him to Octave.

References

Ackroyd, M. H. (1980, January). Computing the waiting time distribution for the G/G/1 queue by signal processing methods. *IEEE Transactions on Communications COM-28*(1), 52–58.

Bednar, J. B. and T. L. Watt (1985). Calculating the complex cepstrum without phase unwrapping or integration. *IEEE ASSP-33*, 1014–1017.

Eaton, J. W. (1995). *Octave. A high-level interactive language for numerical computations*. Austin, TX, U.S.A.: University of Texas at Austin. Version 1.1.1.

Kleinrock, L. (1975). *Queueing Systems, Vol. 1: Theory*. New York: John Wiley & Sons.

Kleinrock, L. (1976). *Queueing Systems, Vol. 2: Computer Applications*. New York: John Wiley & Sons.

Oppenheim, A. V. and R. W. Schafer (1989). *Discrete-Time Signal Processing*. Englewood Cliffs: Prentice Hall.

Tran-Gia, P. (1986). Discrete-time analysis for the interdeparture distribution of GI/G/1 queues. In O. J. Boxma, J. W. Cohen, and H. C. Tijms (Eds.), *Teletraffic Analysis and Computer Performance Evaluation*, pp. 341–357. Elsevier Science Publishers (North-Holland).

Tran-Gia, P. (1989). Zeitdiskrete Analyse in verkehrstheoretischer Modelle in Rechner- und Kommunikationssystemen. Bericht 46, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart.

Tran-Gia, P. (1993, December). Discrete-time analysis technique and application to usage parameter control modelling in ATM systems. In *Proc. 8th Australian Teletraffic Seminar*, Melbourne, Australia.

Appendix

Probability mass functions are stored in Octave data structures with two members:

- `v` the distribution vector,
- `li` the 'real world' index of `v`'s first element, ie. `v[1]` $= PMF($`li`$)$.

A.1 `DDist.m`

`EPS`

Global variable determining the precision criterion i.e. before normalisation $\sum x(i) > 1 -$ `EPS`, if x is a distribution constructed by the following functions.

`DMAX`

Global variable determining the maximum number of distribution elements.

`Distribution (`v `[, `i`])`

If v is a column vector and i is an integer number, `Distribution` (v, i) is the PMF with probability vector v and 'real world' index of the first element of v equal to i. If invoked with one argument i is assumed to be 0.

Probability Mass Functions

`Binomial` (n, p)

If n is an integer number and p a probability, `Binomial` (n, p) is the Binomial distribution with parameters n and p, i.e. the i-th entry of the `Binomial` (n, p) probability vector is equal to $\binom{n}{i} p^i (1-p)^{n-i}$.

`Deterministic` (E)

If E is an integer number, `Deterministic` (E) is the deterministic distribution with mean E.

`Geometric (`E `[, `m`])`

If m is an integer number, `Geometric` (E, m) is the by m shifted geometric distribution with mean E. If invoked with one argument m is assumed to be 0.

`NegBin` (E, c)

If $Ec^2 > 1$, `NegBin` (E, c) is the negative-binomial distribution with mean E and coefficient of variation c.

`Poisson` (E)

`Poisson` (E) is the Poisson distribution with mean E.

`Uniform` (n)

`Uniform` (n) is the uniform distribution with n elements, i.e. each entry of `Uniform` (n) is equal to $1/n$.

Probability Density Functions

`ErlangK` (k, E)

if k is an integer number `ErlangK` (k, E) is a discrete approximation of the Erlang-k distribution density function with mean E.

`HyperExp2` (E, c)

if $c > 1$, `HyperExp2` (E, c) is a discrete approximation of the second order hyper-exponential distribution density function with mean E and coefficient of variation c.

`NegExp` (E)

`NegExp` (E) is discrete approximation of the exponential distribution density function with mean E.

A.2 `DDistOp.m`

`Normalize` (d)

If d is a discrete distribution, `Normalize` (d) is the normalized discrete distribution.

`Truncate` (d)

If d is a discrete distribution, `Truncate` (d) is the truncated discrete distribution such that $\sum_i d(i) < 1 -$ `EPS` or the number of elements is smaller than `DMAX`. The global variables `EPS` and `DMAX` must be defined e.g. by `DDist.m`.

`PiDown` (d, m)

If d is a discrete distribution, `PiDown` (d, m) is the distribution resulting from a 'sweep down' of the elements with indices $> m$ to the m-th element.

`PiUp` (d, m)

If d is a discrete distribution, `PiUp` (d, m) is the distribution resulting from a 'sweep up' of the elements with indices $< m$ to the m-th element.

`SigmaDown` (d, m)

If d is a discrete distribution, `SigmaDown` (d, m) is the distribution resulting from setting the elements with indices $> m$ to zero.

`SigmaUp` (d, m)

If d is a discrete distribution, `SigmaUp` (d, m) is the distribution resulting from setting the elements with indices $< m$ to zero.

`ShiftDown` (d, m)

`ShiftDown` (d, m) is the distribution d shifted down by m, i.e. m is subtracted from the real world indices of the distribution vector.

`ShiftUp` (d, m)

`ShiftUp` (d, m) is the distribution d shifted up by m elements.

`RotateDown` (d, m)

`RotateDown` (d, m) is the distribution d cyclicly shifted downwards by m elements.

`RotateUp` (d, m)

`RotateUp` (d, m) is the distribution d cyclicly shifted upwards by m elements.

`Expand` (d, m)

If d is a discrete distribution, `Expand` (d, m) is the expanded distribution, i.e. with m zeroes inserted between neighbouring elements of d.

`PDF` (d)

If d is a discrete distribution, `PDF` (d) is the probability distribution function of d.

`CPDF` (d)

If d is a discrete distribution, `PDF` (d) is the complementary probability distribution function of d.

`Recurrence` $(d\ [, a])$

If d is a discrete distribution, `Recurrence` (d, a) is the forward recurrence distribution of d with observation instants assumed to be immediately *after* a discrete time instant. If optional parameter a is ommited observation instants are assumed to be immediately *before* a discrete time instant.

(NOTE: The backward recurrence distribution of d with observation instants assumed to be immediately *after* (*before*) a discrete time instant is identical to the forward recurrence distribution of d with observation instants assumed to be immediately *before after* a discrete time instant.)

`Moment` (d, k)

If d is a discrete distribution, `Moment` (d, k) is the k-th moment of d.

`[`m_1, m_2, m_3, v, c`] = Moments (`d`)`

If d is a discrete distribution, m_1, m_2, and m_3 are the 1st, 2nd, and 3rd moments of d; v is the variance of d and c the coefficient of variation.

`Plus (`d_1, d_2`)`

`Plus (`d_1, d_2`)` is the sum of the discrete distributions d_1 and d_2.

`Conv (`d_1, d_2`)`

`Conv (`d_1, d_2`)` is the discrete convolution of d_1 and d_2. If d_1 and d_2 are discrete distributions of random variables D_1 and D_2, resp., `Conv (`d_1, d_2`)` is the discrete distribution of $D_1 + D_2$. For computational economy the algorithm computes the convolution in the FFT domain.

`MConv (`d, m`)`

`MConv (`d, m`)` is the m-fold convolution of d with itself using Discrete Fourier Transform technique.

`XCorr (`d_1, d_2`)`

If d_1 and d_2 are discrete distributions of random variables D_1 and D_2, resp., d is the discrete distribution of $D_1 - D_2$.

`Min (`d_1, d_2`)`

If d_1 and d_2 are discrete distributions of random variables D_1 and D_2, resp., `Min (`d_1, d_2`)` is the discrete distribution of $\min(D_1, D_2)$.

`Max (`d_1, d_2`)`

If d_1 and d_2 are discrete distributions of random variables D_1 and D_2, resp., `Min (`d_1, d_2`)` is the discrete distribution of $\max(D_1, D_2)$.

`[...] = dplot (`d`)`

If d is a discrete distribution, `dplot (`d`)` plots the distribution d, M = `dplot (`d`)` is the matrix for plotting d via the `gplot` command, and $[x,\ y]$ = `dplot (`d`)` returns the x/y vectors for plotting for plotting d via the `gplot` command.

`Ceps (`d`)`

If d is a discrete distribution, `Ceps (`d`)` is the complex cepstrum tranform of d. Since the probability vector is a finite sequence the algorithm uses the FFT instead of z-transforms. The cepstrum is computed employing the algorithm without phase-unwrapping as presented by Bednar and Watt (1985).

`Spec (`x`)`

If x is the complex cepstrum of a discrete distribution, `Ceps (`x`)` is the inverse cepstrum transform of x.

Preprint-Reihe
Institut für Informatik
Universität Würzburg

Verantwortlich: Die Vorstände des Institutes für Informatik.

[75] F. Hübner. Output Process Analysis of the Peak Cell Rate Monitor Algorithm. Januar 1994.

[76] K. Cronauer. A Criterion to Separate Complexity Classes by Oracles. Januar 1994.

[77] M. Ritter. Analysis of the Generic Cell Rate Algorithm Monitoring ON/OFF-Traffic. Januar 1994.

[78] K. Poeck, D. Fensel, D. Landes und J. Angele. Combining KARL and Configurable Role Limiting Methods for Configuring Elevator Systems. Januar 1994.

[79] O. Rose. Approximate Analysis of an ATM Multiplexer with MPEG Video Input. Januar 1994.

[80] A. Schömig. Using Kanban in a Semiconductor Fabrication Environment — a Simulation Study. März 1994.

[81] M. Ritter, S. Kornprobst und F. Hübner. Performance Analysis of Source Policing Architectures in ATM Systems. April 1994.

[82] U. Hertrampf, H. Vollmer und K. W. Wagner. On Balanced vs. Unbalanced Computation Trees. Mai 1994.

[83] M. Mittler und A. Schömig. Entwicklung von "'Due–Date"'–Warteschlangendisziplinen zur Optimierung von Produktionssystemen. Mai 1994.

[84] U. Hertrampf. Complexity Classes Defined via k-valued Functions. Juli 1994.

[85] U. Hertrampf. Locally Definable Acceptance: Closure Properties, Associativity, Finiteness. Juli 1994.

[86] O. Rose und M. R. Frater. Delivery of MPEG Video Services over ATM. August 1994.

[87] B. Reinhardt. Kritik von Symptomerkennung in einem Hypertext-Dokument. August 1994.

[88] U. Rothaug, E. Yanenko und K. Leibnitz. Artificial Neural Networks Used for Way Optimization in Multi-Head Systems in Application to Electrical Flying Probe Testers. September 1994.

[89] U. Hertrampf. Finite Acceptance Type Classes. Oktober 1994.

[90] U. Hertrampf. On Simple Closure Properties of #P. Oktober 1994.

[91] H. Vollmer und K. W. Wagner. Recursion Theoretic Characterizations of Complexity Classes of Counting Functions. November 1994.

[92] U. Hinsberger und R. Kolla. Optimal Technology Mapping for Single Output Cells. November 1994.

[93] W. Nöth und R. Kolla. Optimal Synthesis of Fanoutfree Functions. November 1994.

[94] M. Mittler und R. Müller. Sojourn Time Distribution of the Asymmetric $M/M/1//N$ – System with LCFS-PR Service. November 1994.

[95] M. Ritter. Performance Analysis of the Dual Cell Spacer in ATM Systems. November 1994.

[96] M. Beaudry. Recognition of Nonregular Languages by Finite Groupoids. Dezember 1994.

[97] O. Rose und M. Ritter. A New Approach for the Dimensioning of Policing Functions for MPEG-Video Sources in ATM-Systems. Januar 1995.

[98] T. Dabs und J. Schoof. A Graphical User Interface For Genetic Algorithms. Februar 1995.

[99] M. R. Frater und O. Rose. Cell Loss Analysis of Broadband Switching Systems Carrying VBR Video. Februar 1995.

[100] U. Hertrampf, H. Vollmer und K. W. Wagner. On the Power of Number-Theoretic Operations with Respect to Counting. Januar 1995.

[101] O. Rose. Statistical Properties of MPEG Video Traffic and their Impact on Traffic Modeling in ATM Systems. Februar 1995.

[102] M. Mittler und R. Müller. Moment Approximation in Product Form Queueing Networks. Februar 1995.

[103] D. Rooß und K. W. Wagner. On the Power of Bio-Computers. Februar 1995.

[104] N. Gerlich und M. Tangemann. Towards a Channel Allocation Scheme for SDMA-based Mobile Communication Systems. Februar 1995.

[105] A. Schömig und M. Kahnt. Vergleich zweier Analysemethoden zur Leistungsbewertung von Kanban Systemen. Februar 1995.

[106] M. Mittler, M. Purm und O. Gihr. Set Management: Synchronization of Prefabricated Parts before Assembly. März 1995.

[107] A. Schömig und M. Mittler. Autocorrelation of Cycle Times in Semiconductor Manufacturing Systems. März 1995.

[108] A. Schömig und M. Kahnt. Performance Modelling of Pull Manufacturing Systems with Batch Servers and Assembly-like Structure. März 1995.

[109] M. Mittler, N. Gerlich und A. Schömig. Reducing the Variance of Cycle Times in Semiconductor Manufacturing Systems. April 1995.

[110] A. Schömig und M. Kahnt. A note on the Application of Marie's Method for Queueing Networks with Batch Servers. April 1995.

[111] F. Puppe, M. Daniel und G. Seidel. Qualifizierende Arbeitsgestaltung mit tutoriellen Expertensystemen für technische Diagnoseaufgaben. April 1995.

[112] G. Buntrock, und G. Niemann. Weak Growing Context-Sensitive Grammars. Mai 1995.

[113] J. García and M. Ritter. Determination of Traffic Parameters for VPs Carrying Delay-Sensitive Traffic. Mai 1995.

[114] M. Ritter. Steady-State Analysis of the Rate-Based Congestion Control Mechanism for ABR Services in ATM Networks. Mai 1995.

[115] H. Graefe. Konzepte für ein zuverlässiges Message-Passing-System auf der Basis von UDP. Mai 1995.

[116] A. Schömig und H. Rau. A Petri Net Approach for the Performance Analysis of Business Processes. Mai 1995.

[117] K. Verbarg. Approximate Center Points in Dense Point Sets. Mai 1995.

[118] K. Tutschku. Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications. Juni 1995.

[119] U. Rhein-Desel. Eine "'Übersicht"' über medizinische Informationssysteme: Krankenhausinformationssysteme, Patientenaktensysteme und Kritiksysteme. Juli 1995.

[120] O. Rose. Simple and Efficient Models for Variable Bit Rate MPEG Video Traffic. Juli 1995.

[121] A. Schömig. On Transfer Blocking and Minimal Blocking in Serial Manufacturing Systems — The Impact of Buffer Allocation. Juli 1995.

[122] Th. Fritsch, K. Tutschku und K. Leibnitz. Field Strength Prediction by Ray-Tracing for Adaptive Base Station Positioning in Mobile Communication Networks. August 1995.

[123] R. V. Book, H. Vollmer und K. W. Wagner. On Type-2 Probabilistic Quantifiers. August 1995.

[124] M. Mittler, N. Gerlich, A. Schömig. On Cycle Times and Interdeparture Times in Semiconductor Manufacturing. September 1995.

[125] J. Wolff von Gudenberg. Hardware Support for Interval Arithmetic - Extended Version. Oktober 1995.

[126] M. Mittler, T. Ono-Tesfaye, A. Schömig. On the Approximation of Higher Moments in Open and Closed Fork/Join Primitives with Limited Buffers. November 1995.

[127] M. Mittler, C. Kern. Discrete-Time Approximation of the Machine Repairman Model with Generally Distributed Failure, Repair, and Walking Times. November 1995.