

DIRECTORY OF M-FUNCTIONS AND M-PROCEDURES

We use the term *m-function* to designate a user-defined function as distinct from the basic MATLAB functions which are part of the MATLAB package. For example, the m-function *minterm* produces the specified minterm vector. An *m-procedure* (or sometimes a *procedure*) is an m-file containing a set of MATLAB commands which carry out a prescribed set of operations. Generally, these will prompt for (or assume) certain data upon which the procedure is carried out. We use the term *m-program* to refer to either an m-function or an m-procedure.

In addition to the m-programs there is a collection of m-files with properly formatted data which can be entered into the workspace by calling the file.

Although the m-programs were written for MATLAB version 4.2, they work for versions 5.1, 5.2. The latter versions offer some new features which may make more efficient implementation of some of the m-programs, and which make possible some new ones. With one exception (so noted), these are not explored in this collection.

MATLAB features

Utilization of MATLAB resources is made possible by a systematic analysis of some features of the basic probability model. In particular, the minterm analysis of logical (or Boolean) combinations of events and the analysis of the structure of simple random variables with the aid of indicator functions and minterm analysis are exploited.

A number of standard features of MATLAB are utilized extensively. In addition to standard matrix algebra, we use:

1. Array arithmetic. This involves element by element calculations. For example, if a , b are matrices of the same size, then $a .* b$ is the matrix obtained by multiplying corresponding elements in the two matrices to obtain a new matrix of the same size.
2. Relational operations, such as less than, equal, etc. to obtain zero-one matrices at element positions where the conditions are met.
3. Logical operations on zero-one matrices utilizing logical operators *and*, *or*, and *not*, as well as certain related functions such as *any*, *all*, *find*, etc.
4. Certain MATLAB functions, such as *dot*, *meshgrid*, *sum*, *cumsum*, *prod*, *cumprod* are used repeatedly.

Auxiliary user-defined building blocks

1. **csort.m** One of the most useful is a special sorting and consolidation operation implemented in the m-function *csort*. A standard problem arises when each of a non distinct set of values has an associated probability. To obtain the distribution, it is necessary to sort the values and add the probabilities associated with each distinct value. The following m-function achieves these operations:

```
function [t,p] = csort(T,P).
```

T and P are matrices with the same number of elements. Values of T are sorted and identical values are consolidated; values of P corresponding to identical values of T are added.

A number of derivative functions and procedures utilize *csort*. The following two are useful.

2. **distinct.m** function $y = \text{distinct}(T)$ determines and sorts the distinct members of matrix T .
3. **freq.m** sorts the distinct members of a matrix, counts the number of occurrences of each value, and calculates the cumulative relative frequencies.
4. **dsum.m** function $y = \text{dsum}(v,w)$ determines and sorts the distinct elements among the sums of pairs of elements of row vectors v and w .
5. **rep.m** function $y = \text{rep}(A,m,n)$ replicates matrix A , m times vertically and n times horizontally. Essentially the same as the function *repmat* in MATLAB version 5, released December, 1996.

6. **elrep.m** function $y = \text{elrep}(A, m, n)$ replicates each element of A , m times vertically and n times horizontally.
7. **kronf.m** function $y = \text{kronf}(A, B)$ determines the Kronecker product of matrices A , B . Achieves the same result for full matrices as the MATLAB function *kron*.
8. **colcopy.m** function $y = \text{colcopy}(v, n)$ treats row or column vector v as a column vector and makes a matrix with n columns of v .
9. **colcopyi.m** function $y = \text{colcopyi}(v, n)$ treats row or column vector v as a column vector, reverses the order of the elements, and makes a matrix with n columns of the reversed vector.
10. **rowcopy.m** function $y = \text{rowcopy}(v, n)$ treats row or column vector v as a row vector and makes a matrix with n rows of v .
11. **repseq.m** function $y = \text{repseq}(V, n)$ replicates vector V n times—horizontally if V is a row vector and vertically if V is a column vector.
12. **total.m** Total of all elements in a matrix, calculated by: $\text{total}(x) = \text{sum}(\text{sum}(x))$.
13. **dispv.m** Matrices A , B are transposed and displayed. side by side.
14. **roundn.m** function $y = \text{roundn}(A, n)$ rounds matrix A to n decimal places.
15. **arrep.m** function $y = \text{arrep}(n, k)$ forms all arrangements, with repetition, of k elements from the sequence $1 : n$.

Minterm vectors and probabilities

The analysis of logical combinations of events (as sets) is systematized by the use of the minterm expansion. This leads naturally to the notion of minterm vectors. These are zero-one vectors which can be combined by logical operations. Production of the basic minterm patterns is essential to a number of operations. The following m-programs are key elements of various other programs.

1. **minterm.m** function $y = \text{minterm}(n, k)$ generates the k th minterm vector in a class of n .
2. **mintable.m** function $y = \text{mintable}(n)$ generates a table of minterm vectors by repeated use of the m-function *minterm*.
3. **minvec3.m** sets basic minterm vectors A, B, C, A^c, B^c, C^c for the class $\{A, B, C\}$. (Similarly for *minvec4.m*, *minvec5.m*, etc.)
4. **minmap** function $y = \text{minmap}(pm)$ reshapes a row or column vector pm of minterm probabilities into minterm map format.
5. **binary.m** function $y = \text{binary}(d, n)$ converts a matrix d of floating point nonnegative integers to a matrix of binary equivalents, one on each row. Adapted from m-functions written by Hans Olsson and by Simon Cooke. Each matrix row may be converted to an unspaced string of zeros and ones by the device $ys = \text{setstr}(y + '0')$.
6. **mincalc.m** The m-procedure *mincalc* determines minterm probabilities from suitable data. For a discussion of the data formatting and certain problems, see Sec 1.3 of BASIC PROBABILITY TOPICS using MATLAB or Sec 1.6 of the CAAM 381 text.


```

      % Assumes a data file which includes
      % 1. Call for minvecq to set q basic minterm vectors, each (1 x 2^q)
      % 2. Data vectors DV = matrix of md data Boolean combinations of basic sets--
      %    Matlab produces md minterm vectors-- one on each row.
      %    The first combination is always A|Ac (the whole space)
      % 3. DP = row matrix of md data probabilities.
      %    The first probability is always 1.
      % 4. Target vectors TV = matrix of mt target Boolean combinations.
      %    Matlab produces a row minterm vector for each target combination.
      %    If there are no target combinations, set TV = [];
      
```

```

EXAMPLE
% file rpl_13.m      (Formatted data file)
minvec3
DV = [A|Ac; A; B; A|B; B&Cc; A&B&C; Ac&C; A&Cc];
DP = [1 0.52 0.78 0.85 0.30 0.32 0.31 0.17];
TV = [A&B; A&B&Cc; Ac&C];
disp('Call for mincalc')
>> rpl_13           % Call for data file rpl_13
Variables are A, B, C, Ac, Bc, Cc
They may be renamed, if desired.
Call for mincalc
>> mincalc         % Call for m-program mincalc
Data vectors are linearly independent
Computable target probabilities
    1.0000    0.4500
    2.0000    0.1300
    3.0000    0.3100
The number of minterms is 8
The number of available minterms is 8
Available minterm probabilities are in row vector pma
To view available minterm probabilities, call for PMA

```

7. **mincalet.m** Modification of *mincalc*. — Assumes *mincalc* has been run, calls for new target vectors and performs same calculations as *mincalc*.

Independent events

1. **minprob.m** function $y = \text{minprob}(p)$ calculates minterm probabilities for the basic probabilities in row or column vector p . Uses the m-functions *mintable*, *colcopy*.
2. **imintest.m** function $y = \text{imintest}(pm)$ checks minterm probabilities for independence.
3. **ikn.m** function $y = \text{ikn}(P,k)$ determines the probability of the occurrence of exactly k of the n independent events whose probabilities are in row or column vector P (k may be a row or column vector of nonnegative integers less than or equal to n).
4. **ckn.m** function $y = \text{ckn}(P,k)$ determines the probability of the occurrence of k or more of the n independent events whose probabilities are in row or column vector P (k may be a row or column vector).
5. **parallel.m** function $y = \text{parallel}(p)$ determines the probability of a parallel combination of the independent events whose probabilities are in row or column vector p .

Conditional probability and conditional independence

1. **bayes.m** produces a Bayesian reversal of conditional probabilities. The input consists of $P(E|A_i)$ and $P(A_i)$ for a disjoint class $\{A_i : 1 \leq i \leq n\}$ whose union contains E . The procedure calculates $P(A_i|E)$ and $P(A_i|E^c)$ for $1 \leq i \leq n$.
2. **odds.m** The procedure calculates posterior odds for for a specified profile E . Assumes data has been entered by the procedure *oddsf* or *oddsf*.
3. **oddsdf.m** Sets up calibrating frequencies for calculating posterior odds.
4. **oddsdp.m** Sets up conditional probabilities for odds calculations.

Bernoulli and multinomial trials

1. **btdata.m** Sets parameter p and number n of trials for generating Bernoulli sequences. Prompts for bt to generate the trials.
2. **bt.m** Generates Bernoulli sequence for parameters set by `btdata`. Calculates relative frequency of “successes.”
3. **binomial.m** Uses `ibinom` and `cbinom` to generate *tables* of the individual and cumulative binomial probabilities for specified parameters. *Note* that for calculation in MATLAB it is usually much more convenient and efficient to use *ibinom* and/or *cbinom*.
4. **multinom.m** Multinomial distribution (small N , m).

Some matching problems

1. **Cardmatch.m** Sampling to estimate the probability of one or more matches when one card is drawn from each of nd identical decks of c cards. The number ns of samples is specified.
2. **trialmatch.m** Estimates the probability of matches in n independent trials from identical distributions. The sample size and number of trials must be kept relatively small to avoid exceeding available memory.

Distributions

1. **comb.m** function $y = \text{comb}(n, k)$ Calculates binomial coefficients. k may be a matrix of integers between 0 and n . The result y is a matrix of the same dimensions.
2. **ibinom.m** Binomial distribution — individual terms. We have two m-functions *ibinom* and *cbinom* for calculating individual and cumulative terms, $P(S_n = k)$ and $P(S_n \geq k)$, respectively.

$$P(S_n = k) = C(n, k)p^k(1-p)^{n-k} \quad \text{and} \quad P(S_n \geq k) = \sum_{r=k}^n P(S_n = r) \quad 0 \leq k \leq n$$

For these m-functions, we use a modification of a computation strategy employed by S. Weintraub: TABLES OF THE CUMULATIVE BINOMIAL PROBABILITY DISTRIBUTION FOR SMALL VALUES OF p , 1963. The book contains a particularly helpful error analysis, written by Leo J. Cohen. Experimentation with sums and expectations indicates a precision for *ibinom* and *cbinom* calculations that is better than 10^{-10} for $n = 1000$ and p from 0.01 to 0.99. A similar precision holds for values of n up to 5000, provided np or nq are limited to approximately 500. Above this value for np or nq , the computations break down.

For individual terms, function $y = \text{ibinom}(n, p, k)$ calculates the probabilities for n a positive integer, k a matrix of integers between 0 and n . The output is a matrix of the corresponding binomial probabilities.

3. **cbinom.m** function $y = \text{cbinom}(n, p, k)$ determines the cumulative binomial probabilities $P(S_n \geq k)$, where k a matrix of integers between 0 and n . The output is a matrix of the corresponding cumulative binomial probabilities.
4. **ipoisson.m** Poisson distribution — individual terms. As in the case of the binomial distribution, we have an m-function for the individual terms and one for the cumulative case. The m-functions *ipoisson* and *cpoisson* use a computational strategy similar to that used for the binomial case. Not only does this work for large μ , but the precision is at least as good as that for the binomial m-functions. Experience indicates that the m-functions are good for $\mu \leq 700$. They breaks down at about 710, largely because of limitations of the MATLAB exponential function.

For individual terms, function $y = \text{ipoisson}(\mu, k)$ calculates the probabilities for μ a positive integer, k a row or column vector of nonnegative integers. The output is a row vector of the corresponding Poisson probabilities.

5. **cpoisson.m** Poisson distribution—cumulative terms. function $y = \text{cpoisson}(\mu, k)$, calculates $P(X \geq k)$, where k may be a row or a column vector of nonnegative integers. The output is a row vector of the corresponding probabilities.

6. **nbinom.m** Negative binomial — function $y = \text{nbinom}(m, p, k)$ calculates the probability that the m th success in a Bernoulli sequence occurs on the k th trial.
7. **gaussian.m** function $y = \text{gaussian}(m, v, t)$ calculates the Gaussian (Normal) distribution function for mean value m , variance v , and matrix t of values. The result $y = P(X \leq t)$ is a matrix of the same dimensions as t .
8. **gaussdensity.m** function $y = \text{gaussdensity}(m, v, t)$ calculates the Gaussian density function $f_X(t)$ for mean value m , variance t , and matrix t of values.
9. **norminv.m** function $y = \text{norminv}(m, v, p)$ calculates the inverse (the quantile function) of the Gaussian distribution function for mean value m , variance v , and p a matrix of probabilities.
10. **gammadb.m** function $y = \text{gammadb}(\alpha, \lambda, t)$ calculates the distribution function for a gamma distribution with parameters α, λ . t is a matrix of evaluation points. The result is a matrix of the same size.
11. **beta.m** function $y = \text{beta}(r, s, t)$ calculates the density function for the beta distribution with parameters r, s . t is a matrix of numbers between zero and one. The result is a matrix of the same size.
12. **betadb.m** function $y = \text{betadb}(r, s, t)$ calculates the distribution function for the beta distribution with parameters r, s . t is a matrix of evaluation points. The result is a matrix of the same size.
13. **weibull.m** function $y = \text{weibull}(\alpha, \lambda, t)$ calculates the density function for the Weibull distribution with parameters α, λ . t is a matrix of evaluation points. The result is a matrix of the same size.
14. **weibulld.m** function $y = \text{weibulld}(\alpha, \lambda, t)$ calculates the distribution function for the Weibull distribution with parameters α, λ . t is a matrix of evaluation points. The result is a matrix of the same size.

Binomial, Poisson, and Gaussian distributions

1. **bincomp.m** Graphical comparison of the binomial, Poisson, and Gaussian distributions. The procedure calls for binomial parameters n, p , determines a reasonable range of evaluation points and plots on the same graph the binomial distribution function, the Poisson distribution function, and the gaussian distribution function with the adjustment called the “continuity correction.”
2. **poissapp.m** Graphical comparison of the Poisson and Gaussian distributions. The procedure calls for a value of the Poisson parameter μ , then calculates and plots the Poisson distribution function, the Gaussian distribution function, and the adjusted Gaussian distribution function.

Setup for simple random variables

If a simple random variable X is in canonical form, the distribution consists of the coefficients of the indicator functions (the values of X) and the probabilities of the corresponding events. If X is in a primitive form other than canonical, the `csort` operation is applied to the coefficients of the indicator functions and the probabilities of the corresponding events to obtain the distribution. If $Z = g(X)$ and X is in a primitive form, then the value of Z on the event in the partition associated with t_i is $g(t_i)$. The distribution for Z is obtained by applying `csort` to the $g(t_i)$ and the p_i . Similarly, if $Z = g(X, Y)$ and the joint distribution is available, the value $g(t_i, u_j)$ is associated with $P(X = t_i, Y = u_j)$. The distribution for Z is obtained by applying `csort` to the matrix of values and the corresponding matrix of probabilities.

1. **canonic.m** The procedure determines the distribution for a simple random variable in affine form, when the minterm probabilities are available. Input data are a row vector of coefficients for the indicator functions in the affine form (with the constant value last) and a row vector of the probabilities of the minterm generated by the events. Results consist of a row vector of values and a row vector of the corresponding probabilities.
2. **canonicf.m** function $[x, px] = \text{canonicf}(c, pm)$ is a function version of `canonic`, which allows arbitrary naming of variables.

3. **jcalc.m** Sets up for calculations for joint simple random variables. The matrix P of $P(X = t_i, Y = u_j)$ is arranged as on the plane (i.e., values of Y increase upward). The MATLAB function `meshgrid` is applied to the row matrix X and the reversed row matrix for Y to put an appropriate X -value and Y -value at each position. These are in the “calculating matrices” t and u , respectively, which are used in determining probabilities and expectations of various functions of t, u .
4. **jcalc.m** function `[x,y,t,u,px,py,p] = jcalc(X,Y,P)` is a function version of `jcalc`, which allows arbitrary naming of variables.
5. **jointzw.m** Sets up joint distribution for $Z = g(X, Y)$ and $W = h(X, Y)$ and provides calculating matrices as in `jcalc`. Inputs are P, X , and Y as well as array expressions for $g(t, u)$ and $h(t, u)$. Outputs are matrices Z, W, PZW for the joint distribution, marginal probabilities PZ, PW , and the calculating matrices v, w .
6. **jdtest.m** Tests a joint probability matrix P for negative entries and unit total probability.

Setup for general random variables

1. **tappr.m** Uses the density function to set up a discrete approximation to the distribution for absolutely continuous random variable X .
2. **tuappr.m** Uses the joint density to set up discrete approximations to X, Y, t, u , and density.
3. **dfappr.m** Approximate discrete distribution from distribution function entered as a function of t .
4. **acsetup.m** Approximate distribution for absolutely continuous random variable X . Density is entered as a *string variable* function of t .
5. **dfsetup.m** Approximate discrete distribution from distribution function entered as a *string variable* function of t .

Setup for independent simple random variables

MATLAB version 5.1 has provisions for multidimensional arrays, which make possible more direct implementation of `icalc3` and `icalc4`.

1. **icalc.m** Calculation setup for an independent pair of simple random variables. Input consists of marginal distributions for X, Y , Output is joint distribution and calculating matrices t, u .
2. **icalcf.m** `[x,y,t,u,px,py,p] = icalcf(X,Y,px,py)` is a function version of `icalc`, which allows arbitrary naming of variables.
3. **icalc3.m** Calculation setup for an independent class of three simple random variables.
4. **icalc4.m** Calculation setup for an independent class of four simple random variables.

Calculations for random variables

1. **ddb.m** Uses the distribution of a simple random variable (or simple approximation) to plot a step graph for the distribution function F_X .

2. **cdbn.m** Plots a continuous graph of a distribution function
3. **simple.m** Calculates basic quantities for simple random variables from the distribution, input as row matrices X and PX .
4. **jddb.m** Representation of joint distribution function for simple pair by obtaining the value of F_{XY} at the lower left hand corners of each grid cell.
5. **jsimple.m** Calculates basic quantities for a joint simple pair $\{X, Y\}$ from the joint distribution X, Y, P as in *jcalc*. Calculated quantities include means, variances, covariance, regression line, and regression curve (conditional expectation $E[Y|X = t]$).
6. **japprox.m** Assumes discrete setup and calculates basic quantities for a pair of random variables as in *jsimple*. Plots the regression line and regression curve.

Calculations and tests for independent random variables

1. **mgsum.m** function `[z,pz] = mgsum(x,y,px,py)` determines the distribution for the sum of an independent pair of simple random variables from their distributions.
2. **mgsum3.m** function `[w,pw] = mgsum3(x,y,z,px,py,pz)` extends *mgsum* to three random variables by repeated application of *mgsum*. Similarly for *mgsum4.m*.
3. **mgns.m** function `[z,pz] = mgns(X,P)` determines the distribution for a sum of n independent random variables. X an n -row matrix of X -values and P an n -row matrix of P -values (padded with zeros, if necessary, to make all rows the same length).
4. **mgsumn.m** function `[z,pz] = mgsumn(varargin)` is an alternate to *mgns*, utilizing *varargin* in MATLAB version 5.1. The call is of the form `[z,pz] = mgsumn([x1;p1],[x2;p2], ..., [xn;pn])`.
5. **diidsum.m** function `[x,px] = diidsum(X,PX,n)` determines the sum of n iid simple random variables, with the common distribution X, PX .
6. **itest.m** Tests for independence the matrix P of joint probabilities for a simple pair $\{X, Y\}$ of random variables.
7. **idbn.m** function `p = idbn(px,py)` uses marginal probabilities to determine the joint probability matrix (arranged as on the plane) for an independent pair of simple random variables.
8. **isimple.m** Takes as inputs the marginal distributions for an independent pair $\{X, Y\}$ of simple random variables. Sets up the joint distribution probability matrix P as in *idbn*, and forms the calculating matrices t, u as in *jcalc*. Calculates basic quantities and makes available matrices X, Y, PX, PY, P, t, u for additional calculations.

Quantile functions for bounded distributions

1. **dquant.m** function `t = dquant(X,PX,U)` determines the values of the quantile function for a simple random variable with distribution X, PX at the probability values in row vector U . The probability vector U is often determined by a random number generator.
2. **dquanplot.m** Plots as a stairs graph the quantile function for a simple random variable X . The plot is the values of X versus the distribution function F_X .
3. **dsample.m** Calculates a sample from a discrete distribution, determines the relative frequencies of values, and compares with actual probabilities. Input consists of value and probability matrices for X and the sample size n . A matrix U is determined by a random number generator, and the m-function *dquant* is used to calculate the corresponding sample values. Various data on the sample are calculated and displayed.
4. **quanplot.m** Plots the quantile function for a distribution function F_X . Assumes the procedure *dfsetup* or *acsetup* has been run. A suitable set U of probability values is determined and the m-function *dquant* is used to determine corresponding values of the quantile function. The results are plotted.

5. **qsample.m** Simulates a sample for a given population density. Determines sample parameters and approximate population parameters. Assumes *dfsetup* or *acsetup* has been run. Takes as input the distribution matrices X, PX and the sample size n . Uses a random number generator to obtain the probability matrix U and uses the m-function *dquant* to determine the sample. Assumes *ddfsetup* or *acsetup* has been run.
6. **targetset.m** Setup for arrival at a target set of values. Used in conjunction with the m-procedure *targetrun* to determine the number of trials needed to visit k of a specified set of target values. Input consists of the distribution matrices X, PX and the specified set E of target values.
7. **targetrun.m** Assumes the m-file *targetset* has provided the basic data. Input consists of the number r of repetitions and the number k of the target states to visit. Calculates and displays various results.

Compound demand

The following pattern provides a useful model in many situations. Consider

$$D = \sum_{k=0}^N Y_k$$

where $Y_0 = 0$, and the class $\{Y_k : 1 \leq k\}$ is iid, independent of the counting random variable N . One natural interpretation is to consider N to be the number of customers in a store and Y_k the amount purchased by the k th customer. Then D is the total demand of the actual customers. Hence, we call D the *compound demand*.

1. **gend.m** Uses coefficients of the generating functions for N and Y to calculate, in the integer case, the marginal distribution for the compound demand D and the joint distribution for $\{N, D\}$.
2. **gendf.m** function `[d,pd] = gendf(gn,gy)` is a function version of *gend*, which allows arbitrary naming of the variables. Calculates the distribution for D , but not the joint distribution for $\{N, D\}$.
3. **mgd.m** Uses coefficients for the generating function for N and the distribution for simple Y to calculate the distribution for the compound demand.
4. **mgdf.m** function `[d,pd] = mgdf(pn,y,py)` is a function version of *mgd*, which allows arbitrary naming of the variables. The input matrix pn is the coefficient matrix for the counting random variable generating function. Zeros for the missing powers must be included. The matrices y, py are the actual values and probabilities of the demand random variable.
5. **randbern.m** Let S be the number of successes in a random number N of Bernoulli trials, with probability p of success on each trial. The procedure *randbern* takes as inputs the probability p of success and the distribution matrices N, PN for the counting random variable N and calculates the joint distribution for $\{N, S\}$ and the marginal distribution for S .

Simulation of Markov systems

1. **inventory1.m** Calculates the transition matrix for an (m, M) inventory policy. At the end of each period, if the stock is less than a reorder point m , stock is replenished to the level M . Demand in each period is an integer valued random variable Y . Input consists of the parameters m, M and the distribution for Y as a simple random variable (or a discrete approximation).
2. **branchp.m** Calculates the transition matrix for a simple branching process with a specified maximum population. Input consists of the maximum population value M and the coefficient matrix for the generating function for the individual propagation random variables Z_i . The latter matrix must include zero coefficients for missing powers.
3. **chainset.m** Sets up for simulation of Markov chains. Inputs are the transition matrix \mathbf{P} the set of states, and an optional set of target states. The chain generating procedures listed below assume this procedure has been run.
4. **mchain.m** Assumes *chainset* has been run. Generates trajectory of specified length, with specified initial state.

5. **arrival.m** Assumes *chainset* has been run. Calculates repeatedly the arrival time to a prescribed set of states.
6. **recurrence.m** Assumes *chainset* has been run. Calculates repeatedly the recurrence time to a prescribed set of states, if initial state is in the set; otherwise calculates the arrival time.
7. **kvis.m** Assumes *chainset* has been run. Calculates repeatedly the time to complete visits to a specified k of the states in a prescribed set.
8. **plotdbn** Used after m-procedures *arrival* or *recurrence* to plot arrival or recurrence time distribution.

References

1. Pfeiffer: BASIC PROBABILITY TOPICS USING MATLAB, PWS Publishing Co., 1995
The current set of m-programs updates and extends the set provided with that book. Discussions of the basic computational strategies are, for the most part, still operative. A number of the revisions provide minor changes needed for Version 5.1 and 5.2 of MATLAB. Several changes in input/output formats provide a more uniform style, facilitating the use of more than one m-program or m-function in the solution of a problem.
2. Pfeiffer: AN INTRODUCTION TO APPLIED PROBABILITY USING MATLAB, January, 1998, an unpublished text prepared for the course CAAM 381 and made available as a “course packet” using the services of Copy Club. This text integrates the use of the m-programs into the development of basic probability theory.

Paul E. Pfeiffer
August 12, 1998