17.1: Appendix A to Applied Probability- 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, and 7.04. 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:

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.
Relational operations, such as less than, equal, etc. to obtain zero-one matrices with ones at element positions where the conditions are met.
Logical operations on zero-one matrices utilizing logical operators *and*, *or*, and *not*, as well as certain related functions such as *any*, *all*, *not*, *find*, etc. *Note*. Relational operations and logical operations produce zero-one arrays, called *logical arrays*, which MATLAB treats differently from zero-one *numeric arrays*. A rectangular array in which some rows are logical arrays but others are not is treated as a numeric array. Any zero-one rectangular array can be converted to a numeric array (matrix) by the command `A = ones(size(A)).*A` ,
Certain MATLAB functions, such as *meshgrid*, *sum*, *cumsum*, *prod*, *cumprod* are used repeatedly. The function *dot* for dot product does not work if either array is a logical array. If one of the pair is numeric, the command C = A*B' will work.

Auxiliary user-defined building blocks

csort.m

Description of Code:

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.

Answer

```
function [t,p] = csort(T,P)
% CSORT  [t,p] = csort(T,P) Sorts T, consolidates P
% Version of 4/6/97
% Modified to work with Versions 4.2 and 5.1, 5.2
% T and P matrices with the same number of elements
% The vector T(:)' is sorted:
```

```
%   * Identical values in T are consolidated;
%   * Corresponding values in P are added.
T = T(:)';
n = length(T);
[TS,I] = sort(T);
d  = find([1,TS(2:n) - TS(1:n-1) >1e-13]); % Determines distinct values
t  = TS(d);                                % Selects the distinct values
m  = length(t) + 1;
P  = P(I);                                 % Arranges elements of P
F  = [0 cumsum(P(:)')];
Fd = F([d length(F)]);                     % Cumulative sums for distinct values
p  = Fd(2:m) - Fd(1:m-1);                  % Separates the sums for these values
```

distinct.m

Description of Code:

distinct.m `function y = distinct(T)` determines and sorts the distinct members of matrix T.

Answer

```
function y = distinct(T)
% DISTINCT y = distinct(T) Disinct* members of T
% Version of 5/7/96  Rev 4/20/97 for version 4 & 5.1, 5.2
% Determines distinct members of matrix T.
% Members which differ by no more than 10^{-13}
% are considered identical.  y is a row
% vector of the distinct members.
TS = sort(T(:)');
n  = length(TS);
d  = [1  abs(TS(2:n) - TS(1:n-1)) >1e-13];
y  = TS(find(d));
```

freq.m

Description of Code:

freq.m sorts the distinct members of a matrix, counts the number of occurrences of each value, and calculates the cumulative relative frequencies.

Answer

```
% FREQ file freq.m Frequencies of members of matrix
% Version of 5/7/96
% Sorts the distinct members of a matrix, counts
% the number of occurrences of each value, and
% calculates the cumulative relative frequencies.
T  = input('Enter matrix to be counted  ');
[m,n] = size(T);
[t,f] = csort(T,ones(m,n));
```

```
p = cumsum(f)/(m*n);
disp(['The number of entries is  ',num2str(m*n),])
disp(['The number of distinct entries is  ',num2str(length(t)),] )
disp(' ')
dis = [t;f;p]';
disp('    Values     Count   Cum Frac')
disp(dis)
```

dsum.m

Description of Code:

dsum.m `function y = dsum(v,w)` determines and sorts the distinct elements among the sums of pairs of elements of row vectors **v** and **w**.

Answer

```
function y = dsum(v,w)
% DSUM y = dsum(v,w) Distinct pair sums of elements
% Version of 5/15/97
% y is a row vector of distinct
% values among pair sums of elements
% of matrices v, w.
% Uses m-function distinct
[a,b] = meshgrid(v,w);
t = a+b;
y = distinct(t(:)');
```

rep.m

Description of Code:

rep.m `function y = 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.

Answer

```
function y = rep(A,m,n)
% REP y = rep(A,m,n) Replicates matrix A
% Version of 4/21/96
% Replicates A,
% m times vertically,
% n times horizontally
% Essentially the same as repmat in version 5.1, 5.2
[r,c] = size(A);
R = [1:r]';
C = [1:c]';
v = R(:,ones(1,m));
w = C(:,ones(1,n));
y = A(v,w);
```

elrep.m

Description of Code:

elrep.m `function y = elrep(A,m,n)` replicates each element of **A**, m times vertically and n times horizontally.

Answer

```
function y = elrep(A,m,n)
% ELREP y = elrep(A,m,n) Replicates elements of A
% Version of 4/21/96
% Replicates each element,
% m times vertically,
% n times horizontally
[r,c] = size(A);
R = 1:r;
C = 1:c;
v = R(ones(1,m),:);
w = C(ones(1,n),:);
y = A(v,w);
```

kronf.m

Description of Code:

kronf.m `function y = kronf(A,B)` determines the Kronecker product of matrices **A**,**B** Achieves the same result for full matrices as the MATLAB function *kron*.

Answer

```
function y = kronf(A,B)
% KRONF y = kronf(A,B) Kronecker product
% Version of 4/21/96
% Calculates Kronecker product of full matrices.
% Uses m-functions elrep and rep
% Same result for full matrices as kron for version 5.1, 5.2
[r,c] = size(B);
[m,n] = size(A);
y = elrep(A,r,c).*rep(B,m,n);
```

colcopy.m

Description of Code:

colcopy.m `function y = colcopy(v,n)` treats row or column vector **v** as a column vector and makes a matrix with n columns of **v**.

Answer

```
function y = colcopy(v,n)
% COLCOPY y = colcopy(v,n)  n columns of v
% Version of 6/8/95 (Arguments reversed 5/7/96)
% v a row or column vector
% Treats v as column vector
% and makes n copies
% Procedure based on "Tony's trick"
[r,c] = size(v);
if r == 1
  v = v';
end
y = v(:,ones(1,n));
```

colcopyi.m

Description of Code:

colcopyi.m `function y = 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.

Answer

```
function y = colcopyi(v,n)
% COLCOPYI y = colcopyi(v,n) n columns in reverse order
% Version of 8/22/96
% v a row or column vector.
% Treats v as column vector,
% reverses the order of the
% elements, and makes n copies.
% Procedure based on "Tony's trick"
N = ones(1,n);
[r,c] = size(v);
if r == 1
  v = v(c:-1:1)';
else
  v = v(r:-1:1);
end
y = v(:,N);
```

rowcopy.m

Description of Code:

rowcopy.m `function y = rowcopy(v,n)` treats row or column vector **v** as a row vector and makes a matrix with n rows of **v**.

Answer

```
function y = rowcopy(v,n)
% ROWCOPY y = rowcopy(v,n)  n rows of v
% Version of 5/7/96
% v a row or column vector
% Treats v as row vector
% and makes n copies
% Procedure based on "Tony's trick"
[r,c] = size(v);
if c == 1
  v = v';
end
y = v(ones(1,n),:);
```

repseq.m

Description of Code:

repseq.m `function y = repseq(V,n)` replicates vector V n times—horizontally if V is a row vector and vertically if V is a column vector.

Answer

```
function y = repseq(V,n);
% REPSEQ y = repseq(V,n) Replicates vector V n times
% Version of 3/27/97
% n replications of vector V
% Horizontally if V a row vector
% Vertically if V a column vector
m = length(V);
s = rem(0:n*m-1,m)+1;
y = V(s);
```

total.m

Description of Code:

total.m Total of all elements in a matrix, calculated by: `total(x) = sum(sum(x))` .

Answer

```
function y = total(x)
% TOTAL y = total(x)
% Version of 8/1/93
% Total of all elements in matrix x.
y = sum(sum(x));
```

dispv.m

Description of Code:

dispv.m Matrices A, B are transposed and displayed side by side.

Answer

```
function y = dispv(A,B)
% DISPV y = dispv(A,B) Transpose of A, B side by side
% Version of 5/3/96
% A, B are matrices of the same size
% They are transposed and displayed
% side by side.
y  = [A;B]';
```

roundn.m

Description of Code:

roundn.m `function y = roundn(A,n)` rounds matrix **A** to n decimal places.

Answer

```
function y = roundn(A,n);
% ROUNDN y = roundn(A,n)
% Version of 7/28/97
% Rounds matrix A to n decimals
y = round(A*10^n)/10^n;
```

arrep.m

Description of Code:

arrep.m `function y = arrep(n,k)` forms all arrangements, with repetition, of k elements from the sequence $1:n$.

Answer

```
function y = arrep(n,k);
% ARREP  y = arrep(n,k);
% Version of 7/28/97
% Computes all arrangements of k elements of 1:n,
% with repetition allowed. k may be greater than n.
% If only one input argument n, then k = n.
% To get arrangements of column vector V, use
% V(arrep(length(V),k)).
N = 1:n;
if nargin == 1
  k = n;
end
y = zeros(k,n^k);
for i = 1:k
```

```
  y(i,:) = rep(elrep(N,1,n^(k-i)),1,n^(i-1));
end
```

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.

minterm.m

Description of Code:

minterm.m `function y = minterm(n,k)` generates the kth minterm vector in a class of n.

Answer

```
function y = minterm(n,k)
% MINTERM y = minterm(n,k) kth minterm of class of n
% Version of 5/5/96
% Generates the kth minterm vector in a class of n
% Uses m-function rep
y = rep([zeros(1,2^(n-k)) ones(1,2^(n-k))],1,2^(k-1));
```

mintable.m

Description of Code:

mintable.m `function y = mintable(n)` generates a table of minterm vectors by repeated use of the m-function *minterm*.

Answer

```
function y = mintable(n)
% MINTABLE y = mintable(n)  Table of minterms vectors
% Version of 3/2/93
% Generates a table of minterm vectors
% Uses the m-function minterm
y = zeros(n,2^n);
for i = 1:n
    y(i,:) = minterm(n,i);
end
```

minvec3.m

Description of Code:

minvec3.m sets basic minterm vectors **A**, **B**, **C**, $\mathbf{A}^c$, $\mathbf{B}^c$, $\mathbf{C}^c$, for the class $\{A, B, C\}$. (Similarly for *minvec4.m*, *minvec5.m*, etc.)

Answer

```
% MINVEC3  file minvec3.m Basic minterm vectors
% Version of 1/31/95
A = minterm(3,1);
B = minterm(3,2);
C = minterm(3,3);
Ac = ~A;
Bc = ~B;
Cc = ~C;
disp('Variables are A, B, C, Ac, Bc, Cc')
disp('They may be renamed, if desired.')
```

minmap

Description of Code:

minmap `function  y = minmap(pm)` reshapes a row or column vector **pm** of minterm probabilities into minterm map format.

Answer

```
function y = minmap(pm)
% MINMAP y = minmap(pm) Reshapes vector of minterm probabilities
% Version of 12/9/93
% Reshapes a row or column vector pm of minterm
% probabilities into minterm map format
m = length(pm);
n = round(log(m)/log(2));
a = fix(n/2);
if m ~= 2^n
  disp('The number of minterms is incorrect')
else
  y = reshape(pm,2^a,2^(n-a));
end
```

binary.m

Description of Code:

binary.m `function y = 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 = setstr(y + '0').

Answer

```
function y = binary(d,n)
% BINARY y = binary(d,n) Integers to binary equivalents
% Version of 7/14/95
% Converts a matrix d of floating point, nonnegative
% integers to a matrix of binary equivalents. Each row
% is the binary equivalent (n places) of one number.
```

```
% Adapted from the programs dec2bin.m, which shared
% first prize in an April 95 Mathworks contest.
% Winning authors: Hans Olsson from Lund, Sweden,
% and Simon Cooke from Glasgow, UK.
% Each matrix row may be converted to an unspaced string
% of zeros and ones by the device:  ys = setstr(y + '0').
if nargin < 2,  n = 1; end     % Allows omission of argument n
[f,e] = log2(d);
n = max(max(max(e)),n);
y = rem(floor(d(:)*pow2(1-n:0)),2);
```

mincalc.m

Description of Code:

mincalc.m The m-procedure *mincalc* determines minterm probabilities from suitable data. For a discussion of the data formatting and certain problems, see 2.6.

Answer

```
% MINCALC file mincalc.m Determines minterm probabilities
% Version of 1/22/94 Updated for version 5.1 on  6/6/97
% 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 = [];
[md,nd] = size(DV);
ND = 0:nd-1;
ID = eye(nd);                 % Row i is minterm vector i-1
[mt,nt] = size(TV);
MT = 1:mt;
rd = rank(DV);
if rd < md
   disp('Data vectors are NOT linearly independent')
  else
   disp('Data vectors are linearly independent')
end
% Identification of which minterm probabilities can be determined from the data
% (i.e., which minterm vectors are not linearly independent of data vectors)
AM = zeros(1,nd);
for i = 1:nd
  AM(i) = rd == rank([DV;ID(i,:)]);  % Checks for linear dependence of each
```

```
  end
am = find(AM);                              % minterm vector
CAM = ID(am,:)/DV;     % Determination of coefficients for the available minterm
pma = DP*CAM';                % Calculation of probabilities of available minter
PMA = [ND(am);pma]';
if sum(pma < -0.001) > 0      % Check for data consistency
   disp('Data probabilities are INCONSISTENT')
else
% Identification of which target probabilities are computable from the data
CT = zeros(1,mt);
for j = 1:mt
  CT(j) = rd == rank([DV;TV(j,:)]);
  end
ct  = find(CT);
CCT = TV(ct,:)/DV;            % Determination of coefficients for computable tar
ctp = DP*CCT';                % Determination of probabilities
disp(' Computable target probabilities')
disp([MT(ct); ctp]')
end                           % end for "if sum(pma < -0.001) > 0"
disp(['The number of minterms is ',num2str(nd),])
disp(['The number of available minterms is ',num2str(length(pma)),])
disp('Available minterm probabilities are in vector pma')
disp('To view available minterm probabilities, call for PMA')
```

mincalct.m

Description of Code:

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

Answer

```
% MINCALCT file mincalct.m  Aditional target probabilities
% Version of 9/1/93  Updated for version 5 on 6/6/97
% Assumes a data file which includes
%  1. Call for minvecq to set q basic minterm vectors.
%  2. Data vectors DV.  The first combination is always A|Ac.
%  3. Row matrix DP of data probabilities. The first entry is always 1.
TV = input('Enter matrix of target Boolean combinations  ');
[md,nd] = size(DV);
[mt,nt] = size(TV);
MT = 1:mt;
rd = rank(DV);
CT = zeros(1,mt);   % Identification of computable target probabilities
for j = 1:mt
  CT(j) = rd == rank([DV;TV(j,:)]);
end
```

```
ct  = find(CT);
CCT = TV(ct,:)/DV;  % Determination of coefficients for computable targets
ctp = DP*CCT';      % Determination of probabilities
disp(' Computable target probabilities')
disp([MT(ct); ctp]')
```

Independent events

minprob.m

Description of Code:

minprob.m `function y = minprob(p)` calculates minterm probabilities for the basic probabilities in row or column vector **p**. Uses the m-functions *mintable*, *colcopy*.

Answer

```
function y = minprob(p)
% MINPROB y = minprob(p) Minterm probs for independent events
% Version of 4/7/96
% p is a vector [P(A1) P(A2) ... P(An)], with
% {A1,A2, ... An} independent.
% y is the row vector of minterm probabilities
% Uses the m-functions mintable, colcopy
n = length(p);
M = mintable(n);
a = colcopy(p,2^n);          % 2^n columns, each the vector p
m = a.*M + (1 - a).*(1 - M); % Puts probabilities into the minterm
                             % pattern on its side (n by 2^n)
y = prod(m);                 % Product of each column of m
```

imintest.m

Description of Code:

imintest.m `function y = imintest(pm)` checks minterm probabilities for independence.

Answer

```
function y = imintest(pm)
% IMINTEST y = imintest(pm) Checks minterm probs for independence
% Version of 1/25//96
% Checks minterm probabilities for independence
% Uses the m-functions mintable and minprob
m = length(pm);
n = round(log(m)/log(2));
if m ~= 2^n
  y = 'The number of minterm probabilities is incorrect';
else
P = mintable(n)*pm';
```

```
pt = minprob(P');
a = fix(n/2);
s = abs(pm - pt) > 1e-7;
  if sum(s) > 0
    disp('The class is NOT independent')
    disp('Minterms for which the product rule fails')
    y = reshape(s,2^a,2^(n-a));
  else
    y = 'The class is independent';
  end
end
```

ikn.m

Description of Code:

ikn.m `function y = 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).

Answer

```
function y = ikn(P,k)
% IKN y = ikn(P,k) Individual probabilities of k of n successes
% Version of 5/15/95
% Uses the m-functions mintable, minprob, csort
n = length(P);
T = sum(mintable(n));  % The number of successes in each minterm
pm = minprob(P);       % The probability of each minterm
[t,p] = csort(T,pm);   % Sorts and consolidates success numbers
                       % and adds corresponding probabilities
y = p(k+1);
```

ckn.m

Description of Code:

ckn.m `function y = 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)

Answer

```
function y = ckn(P,k)
% CKN y = ckn(P,k) Probability of k or more successes
% Version of 5/15/95
% Probabilities of k or more of n independent events
% Uses the m-functions mintable, minprob, csort
n = length(P);
m = length(k);
T = sum(mintable(n));  % The number of successes in each minterm
```

```
pm = minprob(P);       % The probability of each minterm
[t,p] = csort(T,pm);   % Sorts and consolidates success numbers
                       % and adds corresponding probabilities
for i = 1:m            % Sums probabilities for each k value
  y(i) = sum(p(k(i)+1:n+1));
end
```

parallel.m

Description of Code:

parallel.m `function y = parallel(p)` determines the probability of a parallel combination of the independent events whose probabilities are in row or column vector **p**.

Answer

```
function y = parallel(p)
% PARALLEL y = parallel(p) Probaaability of parallel combination
% Version of 3/3/93
% Probability of parallel combination.
% Individual probabilities in row matrix p.
y = 1 - prod(1 - p);
```

Conditional probability and conditional idependence

bayes.m

Description of Code:

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 $\{A_i : 1 \le i \le n\}$ whose union contains E. The procedure calculates $P(A_i|E)$ and $P(A_i|E^c)$ for $1 \le i \le n$.

Answer

```
% BAYES file bayes.m Bayesian reversal of conditional probabilities
% Version of 7/6/93
% Input P(E|Ai) and P(Ai)
% Calculates P(Ai|E) and P(Ai|Ec)
disp('Requires input PEA = [P(E|A1) P(E|A2) ... P(E|An)]')
disp(' and PA = [P(A1) P(A2) ... P(An)]')
disp('Determines PAE  = [P(A1|E) P(A2|E) ... P(An|E)]')
disp('       and PAEc = [P(A1|Ec) P(A2|Ec) ... P(An|Ec)]')
PEA  = input('Enter matrix PEA of conditional probabilities  ');
PA   = input('Enter matrix  PA of probabilities  ');
PE   = PEA*PA';
PAE  = (PEA.*PA)/PE;
PAEc = ((1 - PEA).*PA)/(1 -  PE);
disp(' ')
disp(['P(E) = ',num2str(PE),])
disp(' ')
```

```
disp('    P(E|Ai)   P(Ai)     P(Ai|E)   P(Ai|Ec)')
disp([PEA; PA; PAE; PAEc]')
disp('Various quantities are in the matrices PEA, PA, PAE, PAEc, named above')
```

odds.m

Description of Code:

odds.m The procedure calculates posterior odds for for a specified profile E. Assumes data have been entered by the procedure *oddsf* or *oddsp*.

Answer

```
% ODDS file odds.m  Posterior odds for profile
% Version of 12/4/93
% Calculates posterior odds for profile E
% Assumes data has been entered by oddsdf or oddsdp
E = input('Enter profile matrix E  ');
C =  diag(a(:,E))';       % aa = a(:,E) is an n by n matrix whose ith column
D =  diag(b(:,E))';       % is the E(i)th column of a.  The elements on the
                          % diagonal are b(i, E(i)), 1 <= i <= n
                          % Similarly for b(:,E)

R = prod(C./D)*(p1/p2);   % Calculates posterior odds for profile
disp(' ')
disp(['Odds favoring Group 1:   ',num2str(R),])
if R > 1
  disp('Classify in Group 1')
else
  disp('Classify in Group 2')
end
```

oddsdf.m

Description of Code:

oddsdf.m Sets up calibrating frequencies for calculating posterior odds.

Answer

```
% ODDSDF file oddsdf.m Frequencies for calculating odds
% Version of 12/4/93
% Sets up calibrating frequencies
% for calculating posterior odds
A = input('Enter matrix A of frequencies for calibration group 1  ');
B = input('Enter matrix B of frequencies for calibration group 2  ');
n = length(A(:,1));       % Number of questions (rows of A)
m = length(A(1,:));       % Number of answers to each question
p1 = sum(A(1,:));         % Number in calibration group 1
p2 = sum(B(1,:));         % Number in calibration group 2
```

```
a = A/p1;
b = B/p2;
disp(' ')                 % Blank line in presentation
disp(['Number of questions = ',num2str(n),])  % Size of profile
disp(['Answers per question = ',num2str(m),]) % Usually 3: yes, no, uncertain
disp(' Enter code for answers and call for procedure "odds"  ')
disp(' ')
```

oddsdp.m

Description of Code:

oddsdp.m Sets up conditional probabilities for odds calculations.

Answer

```
  % ODDSDP file oddsdp.m Conditional probs for calculating posterior odds
% Version of 12/4/93
% Sets up conditional probabilities
% for odds calculations
a  = input('Enter matrix A of conditional probabilities for Group 1  ');
b  = input('Enter matrix B of conditional probabilities for Group 2  ');
p1 = input('Probability p1 an individual is from Group 1  ');
n = length(a(:,1));
m = length(a(1,:));
p2 = 1 - p1;
disp(' ')                 % Blank line in presentation
disp(['Number of questions = ',num2str(n),])  % Size of profile
disp(['Answers per question = ',num2str(m),]) % Usually 3: yes, no, uncertain
disp(' Enter code for answers and call for procedure "odds"  ')
disp(' ')
```

Bernoulli and multinomial trials

btdata.m

Description of Code:

btdata.m Sets parameter p and number n of trials for generating Bernoulli sequences. Prompts for *bt* to generate the trials.

Answer

```
  % BTDATA file btdata.m Parameters for Bernoulli trials
% Version of 11/28/92
% Sets parameters for generating Bernoulli trials
% Prompts for bt to generate the trials
n = input('Enter n, the number of trials  ');
p = input('Enter p, the probability of success on each trial  ');
disp(' ')
```

```
disp(' Call for bt')
disp(' ')
```

bt.m

Description of Code:

bt.m Generates Bernoulli sequence for parameters set by btdata. Calculates relative frequency of "successes."

Answer

```
% BT file bt.m Generates Bernoulli sequence
% version of 8/11/95  Revised 7/31/97 for version 4.2 and 5.1, 5.2
% Generates Bernoulli sequence for parameters set by btdata
% Calculates relative frequency of 'successes'
clear SEQ;
B  = rand(n,1) <= p;         %  ones for random numbers <= p
F  = sum(B)/n;               % relative frequency of ones
N  = [1:n]';                 % display details
disp(['n = ',num2str(n),'   p = ',num2str(p),])
disp(['Relative frequency = ',num2str(F),])
SEQ = [N B];
clear N;
clear B;
disp('To view the sequence, call for SEQ')
disp(' ')
```

binomial.m

Description of Code:

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*.

Answer

```
01 % BINOMIAL file binomial.m Generates binomial tables
02 % Version of 12/10/92  (Display modified 4/28/96)
03 % Calculates a TABLE of binomial probabilities
04 % for specified n, p, and row vector k,
05 % Uses the m-functions ibinom and cbinom.
06 n = input('Enter n, the number of trials ');
07 p = input('Enter p, the probability of success ');
08 k = input('Enter k, a row vector of success numbers ');
09 y = ibinom(n,p,k);
10 z = cbinom(n,p,k);
11 disp(['    n = ',int2str(n),'    p = ' num2str(p)])
12 H = ['    k         P(X = k)  P(X >= k)'];
13 disp(H)
14 disp([k;y;z]')
```

multinom.m

Description of Code:

multinom.m Multinomial distribution (small N, m).

Answer

```
% MULTINOM file multinom.m  Multinomial distribution
% Version of 8/24/96
% Multinomial distribution (small N, m)
N = input('Enter the number of trials  ');
m = input('Enter the number of types   ');
p = input('Enter the type probabilities  ');
M = 1:m;
T = zeros(m^N,N);
for i = 1:N
  a = rowcopy(M,m^(i-1));
  a = a(:);
  a = colcopy(a,m^(N-i));
  T(:,N-i+1) = a(:);        % All possible strings of the types
end
MT = zeros(m^N,m);
for i = 1:m
 MT(:,i) = sum(T'==i)';
end
clear T                     % To conserve memory
disp('String frequencies for type k are in column matrix MT(:,k)')
P = zeros(m^N,N);
for i = 1:N
  a = rowcopy(p,m^(i-1));
  a = a(:);
  a = colcopy(a,m^(N-i));
  P(:,N-i+1) = a(:);        % Strings of type probabilities
end
PS = prod(P');              % Probability of each string
clear P                     % To conserve memory
disp('String probabilities are in row matrix PS')
```

Some matching problems

Cardmatch.m

Description of Code:

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 nsns of samples is specified.

Answer

```
% CARDMATCH file cardmatch.m Prob of matches in cards from identical decks
% Version of 6/27/97
% Estimates the probability of one or more matches
% in drawing cards from nd decks of c cards each
% Produces a supersample of size n = nd*ns, where
% ns is the number of samples
% Each sample is sorted, and then tested for differences
% between adjacent elements.  Matches are indicated by
% zero differences between adjacent elements in sorted sample
c  = input('Enter the number c  of cards in a deck ');
nd = input('Enter the number nd of decks ');
ns = input('Enter the number ns of sample runs ');
X  = 1:c;                    % Population values
PX = (1/c)*ones(1,c);        % Population probabilities
N  = nd*ns;                  % Length of supersample
U  = rand(1,N);              % Matrix of n random numbers
T  = dquant(X,PX,U);         % Supersample obtained with quantile function;
                             % the function dquant determines quantile
                             % function values of random number sequence U
ex = sum(T)/N;               % Sample average
EX = dot(X,PX);              % Population mean
vx = sum(T.^2)/N - ex^2;     % Sample variance
VX = dot(X.^2,PX) - EX^2;    % Population variance
A  = reshape(T,nd,ns);       % Chops supersample into ns samples of size nd
DS = diff(sort(A));          % Sorts each sample
m  = sum(DS==0)>0;           % Differences between elements in each sample
                             % Zero difference iff there is a match
pm = sum(m)/ns;              % Fraction of samples with one or more matches
Pm = 1 - comb(c,nd)*gamma(nd + 1)/c^(nd);  % Theoretical probability of match
disp('The sample is in column vector T')   % Displays of results
disp(['Sample average ex = ', num2str(ex),])
disp(['Population mean E(X) = ',num2str(EX),])
disp(['Sample variance vx = ',num2str(vx),])
disp(['Population variance V(X) = ',num2str(VX),])
disp(['Fraction of samples with one or more matches   pm = ', num2str(pm),])
disp(['Probability of one or more matches in a sample Pm = ', num2str(Pm),])
```

trialmatch.m

Description of Code:

trialmatch.m Estimates the probability of matches in n independent trials from identical distributions. The sample size and number of trials must be kept relateively small to avoid exceeding available memory.

Answer

```
% TRIALMATCH file trialmatch.m  Estimates probability of matches
% in n independent trials from identical distributions
% Version of 8/20/97
% Estimates the probability of one or more matches
% in a random selection from n identical distributions
% with a small number of possible values
% Produces a supersample of size N = n*ns, where
% ns is the number of samples.  Samples are separated.
% Each sample is sorted, and then tested for differences
% between adjacent elements.  Matches are indicated by
% zero differences between adjacent elements in sorted sample.
X  = input('Enter the VALUES in the distribution ');
PX = input('Enter the PROBABILITIES  ');
c  = length(X);
n  = input('Enter the SAMPLE SIZE n ');
ns = input('Enter the number ns of sample runs ');
N  = n*ns;                  % Length of supersample
U  = rand(1,N);             % Vector of N random numbers
T  = dquant(X,PX,U);        % Supersample obtained with quantile function;
                            %   the function dquant determines quantile
                            %   function values for random number sequence U
ex = sum(T)/N;              % Sample average
EX = dot(X,PX);             % Population mean
vx = sum(T.^2)/N - ex^2;    % Sample variance
VX = dot(X.^2,PX) - EX^2;   % Population variance
A  = reshape(T,n,ns);       % Chops supersample into ns samples of size n
DS = diff(sort(A));         % Sorts each sample
m  = sum(DS==0)>0;          % Differences between elements in each sample
                            % -- Zero difference iff there is a match
pm = sum(m)/ns;             % Fraction of samples with one or more matches
d  = arrep(c,n);
p  = PX(d);
p  = reshape(p,size(d));    % This step not needed in version 5.1
ds = diff(sort(d))==0;
mm = sum(ds)>0;
m0 = find(1-mm);
pm0 = p(:,m0);              % Probabilities for arrangements with no matches
P0 = sum(prod(pm0));
disp('The sample is in column vector T')   % Displays of results
disp(['Sample average ex = ', num2str(ex),])
disp(['Population mean E(X) = ',num2str(EX),])
disp(['Sample variance vx = ',num2str(vx),])
disp(['Population variance V(X) = ',num2str(VX),])
disp(['Fraction of samples with one or more matches   pm = ', num2str(pm),])
disp(['Probability of one or more matches in a sample Pm = ', num2str(1-P0),])
```

comb.m

Description of Code:

comb.m `function y = 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.

Answer

```
function y = comb(n,k)
% COMB y = comb(n,k) Binomial coefficients
% Version of 12/10/92
% Computes binomial coefficients C(n,k)
% k may be a matrix of integers between 0 and n
% result y is a matrix of the same dimensions
y = round(gamma(n+1)./(gamma(k + 1).*gamma(n + 1 - k)));
```

ibinom.m

Description of Code:

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.

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

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 = 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.

Answer

```
function y = ibinom(n,p,k)
% IBINOM  y = ibinom(n,p,k) Individual binomial probabilities
% Version of 10/5/93
% n is a positive integer; p is a probability
% k a matrix of integers between 0 and n
% y = P(X>=k) (a matrix of probabilities)
if p > 0.5
a = [1 ((1-p)/p)*ones(1,n)];
b = [1 n:-1:1];
c = [1 1:n];
br = (p^n)*cumprod(a.*b./c);
bi = fliplr(br);
else
a = [1 (p/(1-p))*ones(1,n)];
b = [1 n:-1:1];
```

```
c = [1 1:n];
bi = ((1-p)^n)*cumprod(a.*b./c);
end
y = bi(k+1);
```

ipoisson.m

Description of Code:

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 = ipoisson(mu,k)` calculates the probabilities for mu a positive integer, k a row or column vector of nonnegative integers. The output is a row vector of the corresponding Poisson probabilities.

Answer

```
function y = ipoisson(mu,k)
% IPOISSON y = ipoisson(mu,k) Individual Poisson probabilities
% Version of 10/15/93
% mu = mean value
% k may be a row or column vector of integer values
% y = P(X = k) (a row vector of probabilities)
K = max(k);
p = exp(-mu)*cumprod([1 mu*ones(1,K)]./[1 1:K]);
y = p(k+1);
```

cpoisson.m

Description of Code:

cpoisson.m Poisson distribution—cumulative terms. `function y = 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.

Answer

```
function y = cpoisson(mu,k)
% CPOISSON y = cpoisson(mu,k) Cumulative Poisson probabilities
% Version of 10/15/93
% mu = mean value mu
% k may be a row or column vector of integer values
% y = P(X >= k) (a row vector of probabilities)
K = max(k);
p = exp(-mu)*cumprod([1 mu*ones(1,K)]./[1 1:K]);
pc = [1 1 - cumsum(p)];
y = pc(k+1);
```

nbinom.m

Description of Code:

nbinom.m Negative binomial — `function y = nbinom(m, p, k)` calculates the probability that the mth success in a Bernoulli sequence occurs on the kth trial.

Answer

```
function y = nbinom(m, p, k)
% NBINOM y = nbinom(m, p, k) Negative binomial probabilities
% Version of 12/10/92
% Probability the mth success occurs on the kth trial
% m a positive integer;  p a probability
% k a matrix of integers greater than or equal to m
% y = P(X=k) (a matrix of the same dimensions as k)
q = 1 - p;
y = ((p^m)/gamma(m)).*(q.^(k - m)).*gamma(k)./gamma(k - m + 1);
```

gaussian.m

Desciption of Code:

gaussian.m `function y = 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.

Answer

```
function y = gaussian(m,v,t)
% GAUSSIAN y = gaussian(m,v,t) Gaussian distribution function
% Version of 11/18/92
% Distribution function for X ~ N(m, v)
% m = mean,  v = variance
% t is a matrix of evaluation points
% y = P(X<=t) (a matrix of the same dimensions as t)
u = (t - m)./sqrt(2*v);
if u >= 0
        y = 0.5*(erf(u) + 1);
else
        y = 0.5*erfc(-u);
end
```

gaussdensity.m

Description of Code:

gaussdensity.m `function y = gaussdensity(m,v,t)` calculates the Gaussian density function $f_X(t)$ for mean value m, variance t, and matrix t of values.

Answer

```
function y = gaussdensity(m,v,t)
% GAUSSDENSITY y = gaussdensity(m,v,t) Gaussian density
% Version of 2/8/96
% m = mean,  v = variance
% t is a matrix of evaluation points
y = exp(-((t-m).^2)/(2*v))/sqrt(v*2*pi);
```

norminv.m

Description of Code:

norminv.m `function y = 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.

Answer

```
function y = norminv(m,v,p)
% NORMINV y = norminv(m,v,p) Inverse gaussian distribution
% (quantile function for gaussian)
% Version of 8/17/94
% m = mean,  v = variance
% t is a matrix of evaluation points
if p >= 0
  u = sqrt(2)*erfinv(2*p - 1);
else
  u = -sqrt(2)*erfinv(1 - 2*p);
end
y = sqrt(v)*u + m;
```

gammadbn.m

Description of Code:

gammadbn.m `function y = gammadbn(alpha, lambda, t)` calculates the distribution function for a gamma distribution with parameters alpha, lambda. t is a matrix of evaluation points. The result is a matrix of the same size.

Answer

```
function y = gammadbn(alpha, lambda, t)
% GAMMADBN y = gammadbn(alpha, lambda, t) Gamma distribution
% Version of 12/10/92
% Distribution function for X ~ gamma (alpha, lambda)
% alpha, lambda are positive parameters
% t may be a matrix of positive numbers
% y = P(X<= t) (a matrix of the same dimensions as t)
y = gammainc(lambda*t, alpha);
```

beta.m

Description of Code:

beta.m `function y = 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.

Answer

```
function y = beta(r,s,t)
% BETA y = beta(r,s,t) Beta density function
% Version of 8/5/93
% Density function for Beta (r,s) distribution
% t is a matrix of evaluation points between 0 and 1
% y is a matrix of the same dimensions as t
y = (gamma(r+s)/(gamma(r)*gamma(s)))*(t.^(r-1).*(1-t).^(s-1));
```

betadbn.m

Description of Code:

betadbn.m `function y = betadbn(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.

Answer

```
function y = betadbn(r,s,t)
% BETADBN y = betadbn(r,s,t) Beta distribution function
% Version of 7/27/93
% Distribution function for X  beta(r,s)
% y = P(X<=t) (a matrix of the same dimensions as t)
y = betainc(t,r,s);
```

weibull.m

Description of Code:

weibull.m `function y = weibull(alpha,lambda,t)` calculates the density function for the Weibull distribution with parameters alpha, lambda. t is a matrix of evaluation points. The result is a matrix of the same size.

Answer

```
function y = weibull(alpha,lambda,t)
% WEIBULL y = weibull(alpha,lambda,t) Weibull density
% Version of 1/24/91
% Density function for X ~ Weibull (alpha, lambda, 0)
% t is a matrix of positive evaluation points
% y is a matrix of the same dimensions as t
y = alpha*lambda*(t.^(alpha - 1)).*exp(-lambda*(t.^alpha));
```

weibulld.m

Description of Code:

weibulld.m `function y = weibulld(alpha, lambda, t)` calculates the distribution function for the Weibull distribution with parameters alpha, lambda. t is a matrix of evaluation points. The result is a matrix of the same size.

Answer

```
function y = weibulld(alpha, lambda, t)
% WEIBULLD y = weibulld(alpha, lambda, t) Weibull distribution function
% Version of 1/24/91
% Distribution function for X ~ Weibull (alpha, lambda, 0)
% t is a matrix of positive evaluation points
% y = P(X<=t) (a matrix of the same dimensions as t)
y = 1 - exp(-lambda*(t.^alpha));
```

Binomial, Poisson, and Gaussian dstributions

bincomp.m

Description of Code:

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.”

Answer

```
% BINCOMP file bincomp.m  Approx of binomial by Poisson and gaussian
% Version of 5/24/96
% Gaussian adjusted for "continuity correction"
% Plots distribution functions for specified parameters n, p
n = input('Enter the parameter n  ');
p = input('Enter the parameter p  ');
a = floor(n*p-2*sqrt(n*p));
a = max(a,1);                          % Prevents zero or negative indices
b = floor(n*p+2*sqrt(n*p));
k = a:b;
Fb = cumsum(ibinom(n,p,0:n));          % Binomial distribution function
Fp = cumsum(ipoisson(n*p,0:n));        % Poisson distribution function
Fg = gaussian(n*p,n*p*(1 - p),k+0.5); % Gaussian distribution function
stairs(k,Fb(k+1))                      % Plotting details
hold on
plot(k,Fp(k+1),'-.',k,Fg,'o')
hold off
xlabel('t values')                     % Graph labeling details
ylabel('Distribution function')
title('Approximation of Binomial by Poisson and Gaussian')
grid
```

```
legend('Binomial','Poisson','Adjusted Gaussian')
disp('See Figure for results')
```

poissapp.m

Desciption of Code:

poissapp.m Graphical comparison of the Poisson and Gaussian distributions. The procedure calls for a value of the Poisson parameter mu, then calculates and plots the Poisson distribution function, the Gaussian distribution function, and the adjusted Gaussian distribution function.

Answer

```
% POISSAPP file poissapp.m  Comparison of Poisson and gaussian
% Version of 5/24/96
% Plots distribution functions for specified parameter mu
mu = input('Enter the parameter mu  ');
n = floor(1.5*mu);
k = floor(mu-2*sqrt(mu)):floor(mu+2*sqrt(mu));
FP = cumsum(ipoisson(mu,0:n));
FG = gaussian(mu,mu,k);
FC = gaussian(mu,mu,k-0.5);
stairs(k,FP(k))
hold on
plot(k,FG,'-.',k,FC,'o')
hold off
grid
xlabel('t values')
ylabel('Distribution function')
title('Gaussian Approximation to Poisson Distribution')
legend('Poisson','Gaussian','Adjusted Gaussian')
disp('See Figure for results')
```

Setup for simple random variables

If a simple random variable X is in canonical form, the distribution consists of the coefficients of the indicator funtions (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.

canonic.m

Description of Code:

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.

Answer

```
% CANONIC file canonic.m Distribution for simple rv in affine form
% Version of 6/12/95
% Determines the distribution for a simple random variable
% in affine form, when the minterm probabilities are available.
% Uses the m-functions mintable and csort.
% The coefficient vector must contain the constant term.
% If the constant term is zero, enter 0 in the last place.
c  = input(' Enter row vector of coefficients  ');
pm = input(' Enter row vector of minterm probabilities  ');
n  = length(c) - 1;
if 2^n ~= length(pm)
   error('Incorrect minterm probability vector length');
end
M  = mintable(n);            % Provides a table of minterm patterns
s  = c(1:n)*M + c(n+1);      % Evaluates X on each minterm
[X,PX] = csort(s,pm);        % s = values; pm = minterm probabilities
XDBN = [X;PX]';
disp('Use row matrices X and PX for calculations')
disp('Call for XDBN to view the distribution')
```

canonicf.m

Description of Code:

canonicf.m `function [x,px] = canonicf(c,pm)` is a function version of canonic, which allows arbitrary naming of variables.

Answer

```
function [x,px] = canonicf(c,pm)
% CANONICF  [x,px] = canonicf(c,pm)  Function version of canonic
% Version of 6/12/95
% Allows arbitrary naming of variables
n = length(c) - 1;
if 2^n ~= length(pm)
   error('Incorrect minterm probability vector length');
end
M  = mintable(n);              % Provides a table of minterm patterns
s  = c(1:n)*M + c(n+1);        % Evaluates X on each minterm
[x,px]  = csort(s,pm);         % s = values; pm = minterm probabilities
```

jcalc.m

Description of Code:

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.

Answer

```
% JCALC  file jcalc.m  Calculation setup for joint simple rv
% Version of 4/7/95 (Update of prompt and display 5/1/95)
% Setup for calculations for joint simple random variables
% The joint probabilities arranged as on the plane
% (top row corresponds to largest value of Y)
P = input('Enter JOINT PROBABILITIES (as on the plane)  ');
X = input('Enter row matrix of VALUES of X  ');
Y = input('Enter row matrix of VALUES of Y  ');
PX = sum(P);            % probabilities for X
PY = fliplr(sum(P'));   % probabilities for Y
[t,u] = meshgrid(X,fliplr(Y));
disp(' Use array operations on matrices X, Y, PX, PY, t, u, and P')
```

jcalcf.m

Description of Code:

jcalcf.m `function [x,y,t,u,px,py,p] = jcalcf(X,Y,P)` is a function version of jcalc, which allows arbitrary naming of variables.

Answer

```
function [x,y,t,u,px,py,p] = jcalcf(X,Y,P)
% JCALCF [x,y,t,u,px,py,p] = jcalcf(X,Y,P) Function version of jcalc
% Version of 5/3/95
% Allows arbitrary naming of variables
if sum(size(P) ~= [length(Y) length(X)]) > 0
  error('     Incompatible vector sizes')
end
x = X;
y = Y;
p = P;
px = sum(P);
py = fliplr(sum(P'));
[t,u] = meshgrid(X,fliplr(Y));
```

jointzw.m

Description of Code:

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.

Answer

```
% JOINTZW file jointzw.m Joint dbn for two functions of (X,Y)
% Version of 4/29/97
% Obtains joint distribution for
% Z = g(X,Y) and W = h(X,Y)
% Inputs P, X, and Y as well as array
% expressions for g(t,u) and h(t,u)
P = input('Enter joint prob for (X,Y) ');
X = input('Enter values for X ');
Y = input('Enter values for Y ');
[t,u] = meshgrid(X,fliplr(Y));
G = input('Enter expression for g(t,u) ');
H = input('Enter expression for h(t,u) ');
[Z,PZ] = csort(G,P);
[W,PW] = csort(H,P);
r = length(W);
c = length(Z);
PZW = zeros(r,c);
for i = 1:r
  for j = 1:c
   a = find((G==Z(j))&(H==W(i)));
   if ~isempty(a)
    PZW(i,j) = total(P(a));
   end
  end
end
PZW = flipud(PZW);
[v,w] = meshgrid(Z,fliplr(W));
if (G==t)&(H==u)
  disp(' ')
  disp('  Note:  Z = X and W = Y')
  disp(' ')
elseif  G==t
  disp(' ')
  disp('  Note:  Z = X')
  disp(' ')
elseif H==u
  disp(' ')
  disp('  Note:  W = Y')
  disp(' ')
end
disp('Use array operations on Z, W, PZ, PW, v, w, PZW')
```

jdtest.m

Description of Code:

jdtest.m Tests a joint probability matrix P for negative entries and unit total probability..

Answer

```
function y = jdtest(P)
% JDTEST y = jdtest(P) Tests P for unit total and negative elements
% Version of 10/8/93
M = min(min(P));
S = sum(sum(P));
if M < 0
  y = 'Negative entries';
elseif abs(1 - S) > 1e-7
  y = 'Probabilities do not sum to one';
else
  y = 'P is a valid distribution';
end
```

Setup for general random variables

tappr.m

Description of Code:

tappr.m Uses the density function to set up a discrete approximation to the distribution for absolutely continuous random variable X.

Answer

```
% TAPPR file tappr.m  Discrete approximation to ac random variable
% Version of 4/16/94
% Sets up discrete approximation to distribution for
% absolutely continuous random variable  X
% Density is entered as a function of t
r = input('Enter matrix [a b] of x-range endpoints  ');
n = input('Enter number of x approximation points  ');
d = (r(2) - r(1))/n;
t = (r(1):d:r(2)-d) +d/2;
PX = input('Enter density as a function of t  ');
PX = PX*d;
PX = PX/sum(PX);
X  = t;
disp('Use row matrices X and PX as in the simple case')
```

tuappr.m

Description of Code:

tuappr.m Uses the joint density to set up discrete approximations to X, Y, t, u, and density.

Answer

```
% TUAPPR file tuappr.m Discrete approximation to joint ac pair
% Version of 2/20/96
% Joint density entered as a function of t, u
% Sets up discrete approximations to X, Y, t, u, and density
rx = input('Enter matrix [a b] of X-range endpoints  ');
ry = input('Enter matrix [c d] of Y-range endpoints  ');
nx = input('Enter number of X approximation points  ');
ny = input('Enter number of Y approximation points  ');
dx = (rx(2) - rx(1))/nx;
dy = (ry(2) - ry(1))/ny;
X  = (rx(1):dx:rx(2)-dx) + dx/2;
Y  = (ry(1):dy:ry(2)-dy) + dy/2;
[t,u] = meshgrid(X,fliplr(Y));
P  = input('Enter expression for joint density  ');
P  = dx*dy*P;
P  = P/sum(sum(P));
PX = sum(P);
PY = fliplr(sum(P'));
disp('Use array operations on X, Y, PX, PY, t, u, and P')
```

dfappr.m

dfappr.m Approximate discrete distribution from distribution function entered as a function of t.

Answer

```
% DFAPPR file dfappr.m Discrete approximation from distribution function
% Version of 10/21/95
% Approximate discrete distribution from distribution
% function entered as a function of t
r = input('Enter matrix [a b] of X-range endpoints  ');
s = input('Enter number of X approximation points  ');
d = (r(2) - r(1))/s;
t = (r(1):d:r(2)-d) +d/2;
m  = length(t);
f  = input('Enter distribution function F as function of t  ');
f  = [0 f];
PX = f(2:m+1) - f(1:m);
PX = PX/sum(PX);
X  = t - d/2;
disp('Distribution is in row matrices X and PX')
```

acsetup.m

Description of Code:

acsetup.m Approximate distribution for absolutely continuous random variable X. Density is entered as a *string variable*function of t.

Answer

```
% ACSETUP file acsetup.m Discrete approx from density as string variable
% Version of 10/22/94
% Approximate distribution for absolutely continuous rv X
% Density is entered as a string variable function of t
disp('DENSITY f is entered as a STRING VARIABLE.')
disp('either defined previously or upon call.')
r  = input('Enter matrix [a b] of x-range endpoints  ');
s  = input('Enter number of x approximation points  ');
d  = (r(2) - r(1))/s;
t  = (r(1):d:r(2)-d) +d/2;
m  = length(t);
f  = input('Enter density as a function of t  ');
PX = eval(f);
PX = PX*d;
PX = PX/sum(PX);
X  = t;
disp('Distribution is in row matrices X and PX')
```

dfsetup.m

Description of Code:

dfsetup.m Approximate discrete distribution from distribution function entered as a *string variable* function of t.

Answer

```
% DFSETUP file dfsetup.m  Discrete approx from string dbn function
% Version of 10/21/95
% Approximate discrete distribution from distribution
% function entered as string variable function of t
disp('DISTRIBUTION FUNCTION F is entered as a STRING')
disp('VARIABLE, either defined previously or upon call')
r = input('Enter matrix [a b] of X-range endpoints  ');
s = input('Enter number of X approximation points  ');
d = (r(2) - r(1))/s;
t = (r(1):d:r(2)-d) +d/2;
m  = length(t);
F  = input('Enter distribution function F as function of t  ');
f  = eval(F);
f  = [0 f];
PX = f(2:m+1) - f(1:m);
PX = PX/sum(PX);
```

```
X  = t - d/2;
disp('Distribution is in row matrices X and PX')
```

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.

icalc.m

Description of Code:

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.

Answer

```
% ICALC file icalc.m  Calculation setup for independent pair
% Version of 5/3/95
% Joint calculation setup for independent pair
X  = input('Enter row matrix of X-values  ');
Y  = input('Enter row matrix of Y-values  ');
PX = input('Enter X probabilities  ');
PY = input('Enter Y probabilities  ');
[a,b] = meshgrid(PX,fliplr(PY));
P  = a.*b;                      % Matrix of joint independent probabilities
[t,u] = meshgrid(X,fliplr(Y));  % t, u matrices for joint calculations
disp(' Use array operations on matrices X, Y, PX, PY, t, u, and P')
```

icalcf.m

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.

Answer

```
function [x,y,t,u,px,py,p] = icalcf(X,Y,PX,PY)
% ICALCF [x,y,t,u,px,py,p] = icalcf(X,Y,PX,PY) Function version of icalc
% Version of 5/3/95
% Allows arbitrary naming of variables
x = X;
y = Y;
px = PX;
py = PY;
if length(X) ~= length(PX)
  error('     X and PX of different lengths')
elseif length(Y) ~= length(PY)
  error('     Y and PY of different lengths')
end
[a,b] = meshgrid(PX,fliplr(PY));
```

```
p   = a.*b;                       % Matrix of joint independent probabilities
[t,u] = meshgrid(X,fliplr(Y));    % t, u matrices for joint calculations
```

icalc3.m

Description of Code:

icalc3.m Calculation setup for an independent class of three simple random variables.

Answer

```
% ICALC3 file icalc3.m Setup for three independent rv
% Version of 5/15/96
% Sets up for calculations for three
% independent simple random variables
% Uses m-functions rep, elrep, kronf
X  = input('Enter row matrix of X-values  ');
Y  = input('Enter row matrix of Y-values  ');
Z  = input('Enter row matrix of Z-values  ');
PX = input('Enter X probabilities  ');
PY = input('Enter Y probabilities  ');
PZ = input('Enter Z probabilities  ');
n  = length(X);
m  = length(Y);
s  = length(Z);
[t,u] = meshgrid(X,Y);
t  = rep(t,1,s);
u  = rep(u,1,s);
v  = elrep(Z,m,n);  % t,u,v matrices for joint calculations
P  = kronf(PZ,kronf(PX,PY'));
disp('Use array operations on matrices X, Y, Z,')
disp('PX, PY, PZ, t, u, v, and P')
```

icalc4.m

Description of Code:

icalc4.m Calculation setup for an independent class of four simple random variables.

Answer

```
% ICALC4 file icalc4.m Setup for four independent rv
% Version of 5/15/96
% Sets up for calculations for four
% independent simple random variables
% Uses m-functions rep, elrep, kronf
X  = input('Enter row matrix of X-values  ');
Y  = input('Enter row matrix of Y-values  ');
Z  = input('Enter row matrix of Z-values  ');
W  = input('Enter row matrix of W-values  ');
```

```
PX = input('Enter X probabilities  ');
PY = input('Enter Y probabilities  ');
PZ = input('Enter Z probabilities  ');
PW = input('Enter W probabilities  ');
n  = length(X);
m  = length(Y);
s  = length(Z);
r  = length(W);
[t,u] = meshgrid(X,Y);
t = rep(t,r,s);
u = rep(u,r,s);
[v,w] = meshgrid(Z,W);
v = elrep(v,m,n);  % t,u,v,w matrices for joint calculations
w = elrep(w,m,n);
P = kronf(kronf(PZ,PW'),kronf(PX,PY'));
disp('Use array operations on matrices X, Y, Z, W')
disp('PX, PY, PZ, PW, t, u, v, w, and P')
```

Calculations for random variables

ddbn.m

Description of Code:

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

Answer

```
% DDBN file ddbn.m Step graph of distribution function
% Version of 10/25/95
% Plots step graph of dbn function FX from
% distribution of simple rv (or simple approximation)
xc  = input('Enter row matrix of VALUES  ');
pc = input('Enter row matrix of PROBABILITIES  ');
m  = length(xc);
FX = cumsum(pc);
xt = [xc(1)-1-0.1*abs(xc(1)) xc xc(m)+1+0.1*abs(xc(m))];
FX = [0 FX 1];        % Artificial extension of range and domain
stairs(xt,FX)         % Plot of stairstep graph
hold on
plot(xt,FX,'o')       % Marks values at jump
hold off
grid
xlabel('t')
ylabel('u = F(t)')
title('Distribution Function')
```

cdbn.m

Description of Code:

cdbn.m Plots a continuous graph of a distribution function of a simple random variable (or simple approximation).

Answer

```
% CDBN file cdbn.m Continuous graph of distribution function
% Version of 1/29/97
% Plots continuous graph of dbn function FX from
% distribution of simple rv (or simple approximation)
xc  = input('Enter row matrix of VALUES  ');
pc = input('Enter row matrix of PROBABILITIES  ');
m  = length(xc);
FX = cumsum(pc);
xt = [xc(1)-0.01 xc xc(m)+0.01];
FX = [0 FX FX(m)];    % Artificial extension of range and domain
plot(xt,FX)           % Plot of continuous graph
grid
xlabel('t')
ylabel('u = F(t)')
title('Distribution Function')
```

simple.m

Description of Code:

simple.m Calculates basic quantites for simple random variables from the distribution, input as row matrices X and PX.

Answer

```
% SIMPLE file simple.m Calculates basic quantites for simple rv
% Version of 6/18/95
X  = input('Enter row matrix of X-values  ');
PX = input('Enter row matrix PX of X probabilities  ');
n  = length(X);          % dimension of X
EX = dot(X,PX)           % E[X]
EX2 = dot(X.^2,PX)       % E[X^2]
VX = EX2 - EX^2          % Var[X]
disp(' ')
disp('Use row matrices X and PX for further calculations')
```

jddbn.m

Description of Code:

jddbn.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.

Answer

```
% JDDBN file jddbn.m  Joint distribution function
% Version of 10/7/96
% Joint discrete distribution function for
% joint  matrix P (arranged as on the plane).
% Values at lower left hand corners of grid cells
P = input('Enter joint probability matrix (as on the plane)  ');
FXY = flipud(cumsum(flipud(P)));
FXY = cumsum(FXY')';
disp('To view corner values for joint dbn function, call for FXY')
```

jsimple.m

Description of Code:

jsimple.m Calculates basic quantities for a joint simple pair $\{X, Y\}$ from the joint distrsibution X, Y, P as in jcalc. Calculated quantities include means, variances, covariance, regression line, and regression curve (conditional expectation $E[Y|X = t]$

Answer

```
% JSIMPLE file jsimple.m  Calculates basic quantities for joint simple rv
% Version of 5/25/95
% The joint probabilities are arranged as on the plane
% (the top row corresponds to the largest value of Y)
P = input('Enter JOINT PROBABILITIES (as on the plane)  ');
X = input('Enter row matrix of VALUES of X  ');
Y = input('Enter row matrix of VALUES of Y  ');
disp(' ')
PX = sum(P);              % marginal distribution for X
PY = fliplr(sum(P'));     % marginal distribution for Y
XDBN = [X; PX]';
YDBN = [Y; PY]';
PT  = idbn(PX,PY);
D  = total(abs(P - PT));  % test for difference
if D > 1e-8               % to prevent roundoff error masking zero
  disp('{X,Y} is NOT independent')
 else
  disp('{X,Y} is independent')
end
disp(' ')
[t,u] = meshgrid(X,fliplr(Y));
EX  = total(t.*P)         % E[X]
EY  = total(u.*P)         % E[Y]
EX2 = total((t.^2).*P)    % E[X^2]
EY2 = total((u.^2).*P)    % E[Y^2]
EXY = total(t.*u.*P)      % E[XY]
VX  = EX2 - EX^2          % Var[X]
VY  = EY2 - EY^2          % Var[Y]
```

```
cv = EXY - EX*EY;          % Cov[X,Y] = E[XY] - E[X]E[Y]
if abs(cv) > 1e-9          % to prevent roundoff error masking zero
   CV = cv
 else
   CV = 0
end
a = CV/VX                  % regression line of Y on X is
b = EY - a*EX              %       u = at + b
R = CV/sqrt(VX*VY);        % correlation coefficient rho
disp(['The regression line of Y on X is:  u = ',num2str(a),'t + ',num2str(b),])
disp(['The correlation coefficient is:  rho = ',num2str(R),])
disp(' ')
eYx = sum(u.*P)./PX;
EYX = [X;eYx]';
disp('Marginal dbns are in X, PX, Y, PY; to view, call XDBN, YDBN')
disp('E[Y|X = x] is in eYx; to view, call for EYX')
disp('Use array operations on matrices X, Y, PX, PY, t, u, and P')
```

japprox.m

Description of Code:

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.

Answer

```
% JAPPROX file japprox.m Basic quantities for ac pair {X,Y}
% Version of 5/7/96
% Assumes tuappr has set X, Y, PX, PY, t, u, P
EX  = total(t.*P)          % E[X]
EY  = total(u.*P)          % E[Y]
EX2 = total(t.^2.*P)       % E[X^2]
EY2 = total(u.^2.*P)       % E[Y^2]
EXY = total(t.*u.*P)       % E[XY]
VX  = EX2 - EX^2           % Var[X]
VY  = EY2 - EY^2           % Var[Y]
cv = EXY - EX*EY;          % Cov[X,Y] = E[XY] - E[X]E[Y]
if abs(cv) > 1e-9  % to prevent roundoff error masking zero
   CV = cv
 else
   CV = 0
end
a = CV/VX                  % regression line of Y on X is
b = EY - a*EX              % u = at + b
R = CV/sqrt(VX*VY);
disp(['The regression line of Y on X is:  u = ',num2str(a),'t + ',num2str(b),])
disp(['The correlation coefficient is:  rho = ',num2str(R),])
```

```
disp(' ')
eY = sum(u.*P)./sum(P);    % eY(t) = E[Y|X = t]
RL = a*X + b;
plot(X,RL,X,eY,'-.')
grid
title('Regression line and Regression curve')
xlabel('X values')
ylabel('Y values')
legend('Regression line','Regression curve')
clear eY                   % To conserve memory
clear RL
disp('Calculate with X, Y, t, u, P, as in joint simple case')
```

Calculations and tests for independent random variables

mgsum.m

Description of Code:

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.

Answer

```
function [z,pz] = mgsum(x,y,px,py)
% MGSUM [z,pz] = mgsum(x,y,px,py)  Sum of two independent simple rv
% Version of 5/6/96
% Distribution for the sum of two independent simple random variables
% x is a vector (row or column) of X values
% y is a vector (row or column) of Y values
% px is a vector (row or column) of X probabilities
% py is a vector (row or column) of Y probabilities
% z and pz are row vectors
[a,b] = meshgrid(x,y);
t  = a+b;
[c,d] = meshgrid(px,py);
p  = c.*d;
[z,pz]  = csort(t,p);
```

mgsum3.m

Description of Code:

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.

Answer

```
function [w,pw] = mgsum3(x,y,z,px,py,pz)
% MGSUM3 [w,pw] = mgsum3(x,y,z,px,py,y) Sum of three independent simple rv
% Version of 5/2/96
% Distribution for the sum of three independent simple random variables
% x is a vector (row or column) of X values
% y is a vector (row or column) of Y values
% z is a vector (row or column) of Z values
% px is a vector (row or column) of X probabilities
% py is a vector (row or column) of Y probabilities
% pz is a vector (row or column) of Z probabilities
% W and pW are row vectors
[a,pa] = mgsum(x,y,px,py);
[w,pw] = mgsum(a,z,pa,pz);
```

mgnsum.m

Description of Code:

mgnsum.m `function [z,pz] = mgnsum(X,P)` determines the distribution for a sum of n independent random variables. X an n-row matrix of X-values and n-row matrix of P-values (padded with zeros, if necessary, to make all rows the same length.

Answer

```
function [z,pz] = mgnsum(X,P)
% MGNSUM [z,pz] = mgnsum(X,P)  Sum of n independent simple rv
% Version of 5/16/96
% Distribution for the sum of n independent simple random variables
% X an n-row matrix of X-values
% P an n-row matrix of P-values
% padded with zeros, if necessary
% to make all rows the same length
[n,r] = size(P);
z  = 0;
pz = 1;
for i = 1:n
  x = X(i,:);
  p = P(i,:);
  x = x(find(p>0));
  p = p(find(p>0));
  [z,pz] = mgsum(z,x,pz,p);
end
```

mgsumn.m

Description of Code:

mgsumn.m `function [z,pz] = mgsumn(varargin)` is an alternate to mgnsum, utilizing *varargin* in MATLAB version 5.1. The call is of the form `[z,pz] = mgsumn([x1;p1],[x2;p2], ..., [xn;pn])` .

Answer

```
function [z,pz] = mgsumn(varargin)
% MGSUMN [z,pz] = mgsumn([x1;p1],[x2;p2], ..., [xn;pn])
% Version of 6/2/97 Uses MATLAB version 5.1
% Sum of n independent simple random variables
% Utilizes distributions in the form [x;px] (two rows)
% Iterates mgsum
n  = length(varargin);   % The number of distributions
z  = 0;                  % Initialization
pz = 1;
for i = 1:n              % Repeated use of mgsum
  [z,pz] = mgsum(z,varargin{i}(1,:),pz,varargin{i}(2,:));
end
```

diidsum.m

Description of Code:

diidsum.m `function [x,px] = diidsum(X,PX,n)` determines the sum of n iid simple random variables, with the common distribution X, PX

Answer

```
function [x,px] = diidsum(X,PX,n)
% DIIDSUM [x,px] = diidsum(X,PX,n) Sum of n iid simple random variables
% Version of 10/14/95 Input rev 5/13/97
% Sum of n iid rv with common distribution X, PX
% Uses m-function mgsum
x  = X;                      % Initialization
px = PX;
for i = 1:n-1
  [x,px] = mgsum(x,X,px,PX);
end
```

itest.m

Description of Code:

itest.m Tests for independence the matrix P of joint probabilities for a simple pair $\{X, Y\}$ of random variables.

Answer

```
% ITEST file itest.m  Tests P for independence
% Version of 5/9/95
% Tests for independence the matrix of joint
% probabilities for a simple pair {X,Y}
pt = input('Enter matrix of joint probabilities  ');
disp(' ')
px = sum(pt);                % Marginal probabilities for X
```

```
py = sum(pt');                  % Marginal probabilities for Y (reversed)
[a,b] = meshgrid(px,py);
PT = a.*b;                      % Joint independent probabilities
D  = abs(pt - PT) > 1e-9;       % Threshold set above roundoff
if total(D) > 0
  disp('The pair {X,Y} is NOT independent')
  disp('To see where the product rule fails, call for D')
else
  disp('The pair {X,Y} is independent')
end
```

idbn.m

Description of Code:

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.

Answer

```
function p = idbn(px,py)
% IDBN p = idbn(px,py)  Matrix of joint independent probabilities
% Version of 5/9/95
% Determines joint probability matrix for two independent
% simple random variables (arranged as on the plane)
[a,b] = meshgrid(px,fliplr(py));
p = a.*b
```

isimple.m

Description of Code:

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.

Answer

```
% ISIMPLE file isimple.m  Calculations for independent simple rv
% Version of 5/3/95
X  = input('Enter row matrix of X-values  ');
Y  = input('Enter row matrix of Y-values  ');
PX = input('Enter X probabilities  ');
PY = input('Enter Y probabilities  ');
[a,b] = meshgrid(PX,fliplr(PY));
P  = a.*b;                      % Matrix of joint independent probabilities
[t,u] = meshgrid(X,fliplr(Y));  % t, u matrices for joint calculations
EX  = dot(X,PX)                 % E[X]
EY  = dot(Y,PY)                 % E[Y]
VX  = dot(X.^2,PX) - EX^2       % Var[X]
```

```
VY  = dot(Y.^2,PY) - EY^2       % Var[Y]
disp(' Use array operations on matrices X, Y, PX, PY, t, u, and P')
```

Quantile functions for bounded distributions

dquant.m

Description of Code:

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.

Answer

```
function t = dquant(X,PX,U)
% DQUANT t = dquant(X,PX,U)  Quantile function for a simple random variable
% Version of 10/14/95
% U is a vector of probabilities
m  = length(X);
n  = length(U);
F  = [0 cumsum(PX)+1e-12];
F(m+1) = 1;                     % Makes maximum value exactly one
if U(n) >= 1                    % Prevents improper values of probability U
  U(n) = 1;
end
if U(1) <= 0
  U(1) = 1e-9;
end
f  = rowcopy(F,n);              % n rows of F
u  = colcopy(U,m);              % m columns of U
t  = X*((f(:,1:m) < u)&(u <= f(:,2:m+1)))';
```

dquanplot.m

Description of Code:

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.

Answer

```
% DQUANPLOT file dquanplot.m  Plot of quantile function for a simple rv
% Version of 7/6/95
% Uses stairs to plot the inverse of FX
X  = input('Enter VALUES for X  ');
PX = input('Enter PROBABILITIES for X  ');
m  = length(X);
F  = [0 cumsum(PX)];
XP = [X X(m)];
```

```
stairs(F,XP)
grid
title('Plot of Quantile Function')
xlabel('u')
ylabel('t = Q(u)')
hold on
plot(F(2:m+1),X,'o')          % Marks values at jumps
hold off
```

dsample.m

Description of Code:

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.

Answer

```
% DSAMPLE file dsample.m  Simulates sample from discrete population
% Version of 12/31/95 (Display revised 3/24/97)
% Relative frequencies vs probabilities for
% sample from discrete population distribution
X  = input('Enter row matrix of VALUES  ');
PX = input('Enter row matrix of PROBABILITIES  ');
n  = input('Sample size n  ');
U  = rand(1,n);
T  = dquant(X,PX,U);
[x,fr] = csort(T,ones(1,length(T)));
disp('    Value      Prob    Rel freq')
disp([x; PX; fr/n]')
ex = sum(T)/n;
EX = dot(X,PX);
vx = sum(T.^2)/n - ex^2;
VX = dot(X.^2,PX) - EX^2;
disp(['Sample average ex = ',num2str(ex),])
disp(['Population mean E[X] = ',num2str(EX),])
disp(['Sample variance vx = ',num2str(vx),])
disp(['Population variance Var[X] = ',num2str(VX),])
```

quanplot.m

Description of Code:

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.

Answer

```
% QUANPLOT file quanplot.m  Plots quantile function for dbn function
% Version of 2/2/96
% Assumes dfsetup or acsetup has been run
% Uses m-function dquant
X  = input('Enter row matrix of values  ');
PX = input('Enter row matrix of probabilities  ');
h  = input('Probability increment h  ');
U  = h:h:1;
T  = dquant(X,PX,U);
U  = [0 U 1];
Te = X(m) + abs(X(m))/20;
T  = [X(1) T Te];
plot(U,T)             % Plot rather than stairs for general case
grid
title('Plot of Quantile Function')
xlabel('u')
ylabel('t = Q(u)')
```

qsample.m

Description of Code:

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 *dfsetup* or *acsetup* has been run.

Answer

```
% QSAMPLE file qsample.m  Simulates sample for given population density
% Version of 1/31/96
% Determines sample parameters
% and approximate population parameters.
% Assumes dfsetup or acsetup has been run
X  = input('Enter row matrix of VALUES  ');
PX = input('Enter row matrix of PROBABILITIES  ');
n  = input('Sample size n =  ');
m  = length(X);
U  = rand(1,n);
T  = dquant(X,PX,U);
ex = sum(T)/n;
EX = dot(X,PX);
vx = sum(T.^2)/n - ex^2;
VX = dot(X.^2,PX) - EX^2;
disp('The sample is in column vector T')
disp(['Sample average ex = ', num2str(ex),])
disp(['Approximate population mean E(X) = ',num2str(EX),])
```

```
disp(['Sample variance vx = ',num2str(vx),])
disp(['Approximate population variance V(X) = ',num2str(VX),])
```

targetset.m

Description of Code:

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.

Answer

```
% TARGETSET file targetset.m  Setup for sample arrival at target set
% Version of 6/24/95
X  = input('Enter population VALUES  ');
PX = input('Enter population PROBABILITIES  ');
ms = length(X);
x = 1:ms;                   % Value indices
disp('The set of population values is')
disp(X);
E  = input('Enter the set of target values  ');
ne = length(E);
e  = zeros(1,ne);
for i = 1:ne
  e(i) = dot(E(i) == X,x);  % Target value indices
end
F  = [0 cumsum(PX)];
A  = F(1:ms);
B  = F(2:ms+1);
disp('Call for targetrun')
```

targetrun.m

Description of Code:

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.

Answer

```
% TARGETRUN file targetrun.m Number of trials to visit k target values
% Version of 6/24/95  Rev for Version 5.1 1/30/98
% Assumes the procedure targetset has been run.
r  = input('Enter the number of repetitions  ');
disp('The target set is')
disp(E)
ks = input('Enter the number of target values to visit  ');
if isempty(ks)
  ks = ne;
```

```
end
if ks > ne
  ks = ne;
end
clear T             % Trajectory in value indices (reset)
R0 = zeros(1,ms);   % Indicator for target value indices
R0(e) = ones(1,ne);
S  = zeros(1,r);    % Number of trials for each run (reset)
for k = 1:r
  R = R0;
  i = 1;
  while sum(R) > ne - ks
    u = rand(1,1);
    s = ((A < u)&(u <= B))*x';
    if R(s) == 1     % Deletes indices as values reached
      R(s) = 0;
    end
    T(i) = s;
    i = i+1;
  end
  S(k) = i-1;
end
if r == 1
  disp(['The number of trials to completion is ',int2str(i-1),])
  disp(['The initial value is ',num2str(X(T(1))),])
  disp(['The terminal value is ',num2str(X(T(i-1))),])
  N  = 1:i-1;
  TR = [N;X(T)]';
  disp('To view the trajectory, call for TR')
else
  [t,f] = csort(S,ones(1,r));
  D  = [t;f]';
  p  = f/r;
  AV = dot(t,p);
  SD = sqrt(dot(t.^2,p) - AV^2);
  MN = min(t);
  MX = max(t);
  disp(['The average completion time is ',num2str(AV),])
  disp(['The standard deviation is ',num2str(SD),])
  disp(['The minimum completion time is ',int2str(MN),])
  disp(['The maximum completion time is ',int2str(MX),])
  disp(' ')
  disp('To view a detailed count, call for D.')
  disp('The first column shows the various completion times;')
  disp('the second column shows the numbers of trials yielding those times')
  plot(t,cumsum(p))
  grid
```

```
  title('Fraction of Runs t Steps or Less')
  ylabel('Fraction of runs')
  xlabel('t = number of steps to complete run')
end
```

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 kth customer. Then D is the total demand of the actual customers. Hence, we call D the *compound demand*.

gend.m

Description of Code:

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 $\{N, D\}$

Answer

```
% GEND file gend.m   Marginal and joint dbn for integer compound demand
% Version of 5/21/97
% Calculates marginal distribution for compound demand D
% and joint distribution for {N,D} in the integer case
% Do not forget zero coefficients for missing powers
% in the generating functions for N, Y
disp('Do not forget zero coefficients for missing powers')
gn = input('Enter gen fn COEFFICIENTS for gN  ');
gy = input('Enter gen fn COEFFICIENTS for gY  ');
n  = length(gn) - 1;           % Highest power in gN
m  = length(gy) - 1;           % Highest power in gY
P  = zeros(n + 1,n*m + 1);     % Base for generating P
y  = 1;                        % Initialization
P(1,1) = gn(1);                % First row of P (P(N=0) in the first position)
for i = 1:n                    % Row by row determination of P
   y  = conv(y,gy);            % Successive powers of gy
   P(i+1,1:i*m+1) = y*gn(i+1); % Successive rows of P
end
PD = sum(P);                   % Probability for each possible value of D
a  = find(gn);                 % Location of nonzero N probabilities
b  = find(PD);                 % Location of nonzero D probabilities
P  = P(a,b);                   % Removal of zero rows and columns
P  = rot90(P);                 % Orientation as on the plane
N  = 0:n;
N  = N(a);                     % N values with positive probabilites
PN = gn(a);                    % Positive N probabilities
Y  = 0:m;                      % All possible values of Y
```

```
Y  = Y(find(gy));              % Y values with positive probabilities
PY = gy(find(gy));             % Positive Y proabilities
D  = 0:n*m;                    % All possible values of D
PD = PD(b);                    % Positive D probabilities
D  = D(b);                     % D values with positive probabilities
gD = [D; PD]';                 % Display combination
disp('Results are in N, PN, Y, PY, D, PD, P')
disp('May use jcalc or jcalcf on N, D, P')
disp('To view distribution for D, call for gD')
```

gendf.m

Description of Code:

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 $\{N, D\}$

Answer

```
function [d,pd] = gendf(gn,gy)
% GENDF [d,pd] = gendf(gN,gY) Function version of gend.m
% Calculates marginal for D in the integer case
% Version of 5/21/97
% Do not forget zero coefficients for missing powers
% in the generating functions for N, Y
n  = length(gn) - 1;           % Highest power in gN
m  = length(gy) - 1;           % Highest power in gY
P  = zeros(n + 1,n*m + 1);     % Base for generating P
y  = 1;                        % Initialization
P(1,1) = gn(1);                % First row of P (P(N=0) in the first position)
for i = 1:n                    % Row by row determination of P
   y  = conv(y,gy);            % Successive powers of gy
   P(i+1,1:i*m+1) = y*gn(i+1); % Successive rows of P
end
PD = sum(P);                   % Probability for each possible value of D
D  = 0:n*m;                    % All possible values of D
b  = find(PD);                 % Location of nonzero D probabilities
d  = D(b);                     % D values with positive probabilities
pd = PD(b);                    % Positive D probabilities
```

mgd.m

Description of Code:

mgd.m Uses coefficients for the generating function for N and the distribution for simple Y to calculate the distribution for the compound demand.

Answer

```
% MGD file mgd.m  Moment generating function for compound demand
% Version of 5/19/97
% Uses m-functions csort, mgsum
disp('Do not forget zeros coefficients for missing')
disp('powers in the generating function for N')
disp(' ')
g  = input('Enter COEFFICIENTS for gN  ');
y  = input('Enter VALUES for Y  ');
p  = input('Enter PROBABILITIES for Y  ');
n  = length(g);              % Initialization
a  = 0;
b  = 1;
D  = a;
PD = g(1);
for i = 2:n
  [a,b] = mgsum(y,a,p,b);
  D  = [D a];
  PD = [PD b*g(i)];
  [D,PD] = csort(D,PD);
end
r = find(PD>1e-13);
D = D(r);                    % Values with positive probability
PD = PD(r);                  % Corresponding probabilities
mD = [D; PD]';               % Display details
disp('Values are in row matrix D; probabilities are in PD.')
disp('To view the distribution, call for mD.')
```

Exercise 17.1.1

Description of Code:

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.

Answer

```
function [d,pd] = mgdf(pn,y,py)
% MGDF [d,pd] = mgdf(pn,y,py)  Function version of mgD
% Version of 5/19/97
% Uses m-functions mgsum and csort
% Do not forget zeros coefficients for missing
% powers in the generating function for N
n  = length(pn);              % Initialization
a  = 0;
b  = 1;
d  = a;
pd = pn(1);
```

```
for i = 2:n
  [a,b] = mgsum(y,a,py,b);
  d  = [d a];
  pd = [pd b*pn(i)];
  [d,pd] = csort(d,pd);
end
a  = find(pd>1e-13);          % Location of positive probabilities
pd = pd(a);                   % Positive probabilities
d  = d(a);                    % D values with positive probability
```

Exercise 17.1.1

Description of Code:

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.

Answer

```
% RANDBERN file randbern.m   Random number of Bernoulli trials
% Version of 12/19/96; notation modified 5/20/97
% Joint and marginal distributions for a random number of Bernoulli trials
% N is the number of trials
% S is the number of successes
p  = input('Enter the probability of success  ');
N  = input('Enter VALUES of N  ');
PN = input('Enter PROBABILITIES for N  ');
n  = length(N);
m  = max(N);
S  = 0:m;
P  = zeros(n,m+1);
for i = 1:n
  P(i,1:N(i)+1) = PN(i)*ibinom(N(i),p,0:N(i));
end
PS = sum(P);
P  = rot90(P);
disp('Joint distribution N, S, P, and marginal PS')
```

Simulation of Markov systems

Exercise 17.1.1

Description of Code:

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).

Answer

```
% INVENTORY1 file inventory1.m  Generates P for (m,M) inventory policy
% Version of 1/27/97
% Data for transition probability calculations
% for (m,M) inventory policy
M  = input('Enter value M of maximum stock  ');
m  = input('Enter value m of reorder point  ');
Y  = input('Enter row vector of demand values  ');
PY = input('Enter demand probabilities  ');
states = 0:M;
ms = length(states);
my = length(Y);
% Calculations for determining P
[y,s] = meshgrid(Y,states);
T  =  max(0,M-y).*(s < m) + max(0,s-y).*(s >= m);
P  = zeros(ms,ms);
for i = 1:ms
   [a,b] = meshgrid(T(i,:),states);
   P(i,:) = PY*(a==b)';
end
disp('Result is in matrix P')
```

branchp.m

Description of Code:

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.

Answer

```
% BRANCHP file branchp.m  Transition P for simple branching process
% Version of 7/25/95
% Calculates transition matrix for a simple branching
% process with specified maximum population.
disp('Do not forget zero probabilities for missing values of Z')
PZ = input('Enter PROBABILITIES for individuals  ');
M  = input('Enter maximum allowable population  ');
mz = length(PZ) - 1;
EZ = dot(0:mz,PZ);
disp(['The average individual propagation is ',num2str(EZ),])
P  = zeros(M+1,M+1);
Z  = zeros(M,M*mz+1);
k  = 0:M*mz;
a  = min(M,k);
z  = 1;
P(1,1) = 1;
for i = 1:M                 % Operation similar to gend
```

```
  z = conv(PZ,z);
  Z(i,1:i*mz+1) = z;
  [t,p] = csort(a,Z(i,:));
  P(i+1,:) = p;
end
disp('The transition matrix is P')
disp('To study the evolution of the process, call for branchdbn')
```

chainset.m

Description of Code:

chainset.m Sets up for simulation of Markov chains. Inputs are the transition matrix **P** the set of states, and an optional set of target states. The chain generating procedures listed below assume this procedure has been run.

Answer

```
% CHAINSET file chainset.m Setup for simulating Markov chains
% Version of 1/2/96 Revise 7/31/97 for version 4.2 and 5.1
P  = input('Enter the transition matrix  ');
ms = length(P(1,:));
MS = 1:ms;
states = input('Enter the states if not 1:ms  ');
if isempty(states)
  states = MS;
end
disp('States are')
disp([MS;states]')
PI = input('Enter the long-run probabilities  ');
F  = [zeros(1,ms); cumsum(P')]';
A  = F(:,MS);
B  = F(:,MS+1);
e  = input('Enter the set of target states  ');
ne = length(e);
E  = zeros(1,ne);
for i = 1:ne
  E(i) = MS(e(i)==states);
end
disp(' ')
disp('Call for for appropriate chain generating procedure')
```

mchain.m

Description of Code:

mchain.m Assumes *chainset* has been run. Generates trajectory of specified length, with specified initial state.

Answer

```
% MCHAIN file mchain.m  Simulation of Markov chains
% Version of 1/2/96  Revised 7/31/97 for version 4.2 and 5.1
% Assumes the procedure chainset has been run
n  = input('Enter the number n of stages   ');
st = input('Enter the initial state  ');
if ~isempty(st)
  s  = MS(st==states);
else
  s = 1;
end
T  = zeros(1,n);           % Trajectory in state numbers
U  = rand(1,n);
for i = 1:n
  T(i) = s;
  s = ((A(s,:) < U(i))&(U(i) <= B(s,:)))*MS';
end
N  = 0:n-1;
tr = [N;states(T)]';
n10 = min(n,11);
TR = tr(1:n10,:);
f  = ones(1,n)/n;
[sn,p] = csort(T,f);
if isempty(PI)
  disp('     State     Frac')
  disp([states; p]')
else
  disp('     State     Frac       PI')
  disp([states; p; PI]')
end
disp('To view the first part of the trajectory of states, call for TR')
```

arrival.m

Description of Code:

arrival.m Assumes *chainset* has been run. Calculates repeatedly the arrival time to a prescribed set of states.

Answer

```
% ARRIVAL file arrival.m  Arrival time to a set of states
% Version of 1/2/96  Revised 7/31/97 for version 4.2 and 5.1
% Calculates repeatedly the arrival
% time to a prescribed set of states.
% Assumes the procedure chainset has been run.
r  = input('Enter the number of repetitions  ');
disp('The target state set is:')
disp(e)
st = input('Enter the initial state  ');
```

```
if ~isempty(st)
  s1 = MS(st==states); % Initial state number
else
  s1 = 1;
end
clear T               % Trajectory in state numbers (reset)
S  = zeros(1,r);      % Arrival time for each rep  (reset)
TS = zeros(1,r);      % Terminal state number for each rep (reset)
for k = 1:r
  R  = zeros(1,ms);   % Indicator for target state numbers
  R(E) = ones(1,ne);  % reset for target state numbers
  s  = s1;
  T(1) = s;
  i  = 1;
  while R(s) ~= 1     % While s is not a target state number
    u = rand(1,1);
    s = ((A(s,:) < u)&(u <= B(s,:)))*MS';
    i = i+1;
    T(i) = s;
  end
  S(k) = i-1;         % i is the number of stages; i-1 is time
  TS(k) = T(i);
end
[ts,ft] = csort(TS,ones(1,r));  % ts = terminal state numbers  ft = frequencies
fts = ft/r;                     % Relative frequency of each ts
[a,at]  = csort(TS,S);          % at = arrival time for each ts
w  = at./ft;                    % Average arrival time for each ts
RES = [states(ts); fts; w]';
disp(' ')
if r == 1
  disp(['The arrival time is ',int2str(i-1),])
  disp(['The state reached is ',num2str(states(ts)),])
  N = 0:i-1;
  TR = [N;states(T)]';
  disp('To view the trajectory of states, call for TR')
else
  disp(['The result of ',int2str(r),' repetitions is:'])
  disp('Term state  Rel Freq   Av time')
  disp(RES)
  disp(' ')
  [t,f]  = csort(S,ones(1,r));  % t = arrival times   f = frequencies
  p  = f/r;                     % Relative frequency of each t
  dbn = [t; p]';
  AV = dot(t,p);
  SD = sqrt(dot(t.^2,p) - AV^2);
  MN = min(t);
  MX = max(t);
```

```
  disp(['The average arrival time is ',num2str(AV),])
  disp(['The standard deviation is ',num2str(SD),])
  disp(['The minimum arrival time is ',int2str(MN),])
  disp(['The maximum arrival time is ',int2str(MX),])
  disp('To view the distribution of arrival times, call for dbn')
  disp('To plot the arrival time distribution, call for plotdbn')
end
```

recurrence.m

Description of Code:

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.

Answer

```
% RECURRENCE file recurrence.m Recurrence time to a set of states
% Version of 1/2/96  Revised 7/31/97 for version 4.2 and 5.1
% Calculates repeatedly the recurrence time
% to a prescribed set of states, if initial
% state is in the set; otherwise arrival time.
% Assumes the procedure chainset has been run.
r  = input('Enter the number of repititions  ');
disp('The target state set is:')
disp(e)
st = input('Enter the initial state  ');
if ~isempty(st)
  s1 = MS(st==states); % Initial state number
else
  s1 = 1;
end
clear T                % Trajectory in state numbers (reset)
S  = zeros(1,r);       % Recurrence time for each rep  (reset)
TS = zeros(1,r);       % Terminal state number for each rep (reset)
for k = 1:r
  R  = zeros(1,ms);    % Indicator for target state numbers
  R(E) = ones(1,ne);   % reset for target state numbers
  s  = s1;
  T(1) = s;
  i  = 1;
  if R(s) == 1
    u = rand(1,1);
    s = ((A(s,:) < u)&(u <= B(s,:)))*MS';
    i = i+1;
    T(i) = s;
  end
  while R(s) ~= 1      % While s is not a target state number
```

```
    u = rand(1,1);
    s = ((A(s,:) < u)&(u <= B(s,:)))*MS';
    i = i+1;
    T(i) = s;
  end
  S(k) = i-1;          % i is the number of stages; i-1 is time
  TS(k) = T(i);
end
[ts,ft]  = csort(TS,ones(1,r)); % ts = terminal state numbers  ft = frequencies
fts = ft/r;                % Relative frequency of each ts
[a,tt]  = csort(TS,S);    % tt = total time for each ts
w  = tt./ft;               % Average time for each ts
RES = [states(ts); fts; w]';
disp(' ')
if r == 1
  disp(['The recurrence time is ',int2str(i-1),])
  disp(['The state reached is ',num2str(states(ts)),])
  N = 0:i-1;
  TR = [N;states(T)]';
  disp('To view the trajectory of state numbers, call for TR')
else
  disp(['The result of ',int2str(r),' repetitions is:'])
  disp('Term state  Rel Freq   Av time')
  disp(RES)
  disp(' ')
  [t,f]  = csort(S,ones(1,r));  % t = recurrence times   f = frequencies
  p  = f/r;                     % Relative frequency of each t
  dbn = [t; p]';
  AV = dot(t,p);
  SD = sqrt(dot(t.^2,p) - AV^2);
  MN = min(t);
  MX = max(t);
  disp(['The average recurrence time is ',num2str(AV),])
  disp(['The standard deviation is ',num2str(SD),])
  disp(['The minimum recurrence time is ',int2str(MN),])
  disp(['The maximum recurrence time is ',int2str(MX),])
  disp('To view the distribution of recurrence times, call for dbn')
  disp('To plot the recurrence time distribution, call for plotdbn')
end
```

kvis.m

Description of Code:

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.

Answer

```
% KVIS file kvis.m  Time to complete k visits to a set of states
% Version of 1/2/96 Revised 7/31/97 for version 4.2 and 5.1
% Calculates repeatedly the time to complete
% visits to k of the states in a prescribed set.
% Default is visit to all the target states.
% Assumes the procedure chainset has been run.
r  = input('Enter the number of repetitions  ');
disp('The target state set is:')
disp(e)
ks = input('Enter the number of target states to visit  ');
if isempty(ks)
  ks = ne;
end
if ks > ne
  ks = ne;
end
st = input('Enter the initial state  ');
if ~isempty(st)
  s1 = MS(st==states); % Initial state number
else
  s1 = 1;
end
disp(' ')
clear T                    % Trajectory in state numbers (reset)
R0 = zeros(1,ms);          % Indicator for target state numbers
R0(E) = ones(1,ne);        % reset
S = zeros(1,r);            % Terminal transitions for each rep (reset)
for k = 1:r
  R = R0;
  s  = s1;
  if R(s) == 1
    R(s) = 0;
  end
  i  = 1;
  T(1) = s;
  while sum(R) > ne - ks
    u = rand(1,1);
    s = ((A(s,:) < u)&(u <= B(s,:)))*MS';
    if R(s) == 1
      R(s) = 0;
    end
    i = i+1;
    T(i) = s;
  end
  S(k) = i-1;
end
```

```
if r == 1
  disp(['The time for completion is ',int2str(i-1),])
  N = 0:i-1;
  TR = [N;states(T)]';
  disp('To view the trajectory of states, call for TR')
else
  [t,f]  = csort(S,ones(1,r));
  p  = f/r;
  D  = [t;f]';
  AV = dot(t,p);
  SD = sqrt(dot(t.^2,p) - AV^2);
  MN = min(t);
  MX = max(t);
  disp(['The average completion time is ',num2str(AV),])
  disp(['The standard deviation is ',num2str(SD),])
  disp(['The minimum completion time is ',int2str(MN),])
  disp(['The maximum completion time is ',int2str(MX),])
  disp(' ')
  disp('To view a detailed count, call for D.')
  disp('The first column shows the various completion times;')
  disp('the second column shows the numbers of trials yielding those times')
end
```

plotdbn

Description of Code:

plotdbn Used after m-procedures *arrival* or *recurrence* to plot arrival or recurrence time distribution.

Answer

```
% PLOTDBN file plotdbn.m
% Version of 1/23/98
% Plot arrival or recurrence time dbn
% Use after procedures arrival or recurrence
% to plot arrival or recurrence time distribution
plot(t,p,'-',t,p,'+')
grid
title('Time Distribution')
xlabel('Time in number of transitions')
ylabel('Relative frequency')
```