Necessary and useful functions used in marginal mixture estimation

Supply descriptions XXX

– **acc.sci**

The function computes accuracy of classification, i.e. the number of good classifications divided by the number of data used.

```
function [ac,z]=acc(y,yp,n)
  // accuracy
  if argn(2)<3, n=0; end
  e=abs(y(:)-yp(:))<=n;
  ac=sum(e)/length(y);
  z.all=length(y);
  z.wrong=sum(y(:)~=yp(:));
  z.good=sum(y(:)==yp(:));
endfunction
```

– **amax.sci**

The function computes argument of maxima of a vector.

```
function [a,m]=amax(x,tx)
  // arg of max
  if argn(2)<2,
    [m,a]=max(x);
  else
    if tx==1, tx='r'; end
    if tx==2, tx='c'; end
    [m,a]=max(x,tx);
  end
endfunction
```

– **binom1.sci**

The function constructs the probability function of the modified binomial distribution in a discretized form (as a vector of values).

```
function p=binom1(x,p,N1)
  // modified binomial pf
  // p    probability, i.e. f(x)
```

```
  // x    value 1,2,..,N1 (N1 is thr number of values)
  // p    Bernoulli probability
  // N1   number of values (N for binomial is N1-1)
  p=binomial(p,N1-1)(x);
endfunction
```

– **c2c.sci**

The function commutes the values of the estimated variable so that it corresponds with the simulated (real) one.
Remark: *The problem occurs when estimating the values of a pointer in mixture estimation. The existing clusters can be captured by components in a different order then we defined in simulation (in dependence on their initial centers). Then we look for the combination of the values which give the best result and reorder the results in this way.*

```
function [q,T]=c2c(ct,Ect)
  // cc=c2c(ct,Ect)   permutation of pointer values for plot
  // ct      simulated pointer
  // Ect     estimated pointer
  // q       order vector for Ect
  // T       transf. matrix
  //
  // USSAGE:       ct=q(Ect)
  // set:  [q T]=c2c(ct,Ect);      simul and estim pointer
  // plot(1:nd,ct,1:nd,q(Ect))     plotting
  //
  n=min([length(ct),length(Ect)]);
  if max(ct(1:n))~=max(Ect(1:n))
    disp 'WARNING from c2c.sci: Different numbers of components'
  end

  nc=max(ct);
  T=zeros(nc,nc);

  for t=1:n
    T(ct(t),Ect(t))=T(ct(t),Ect(t))+1;  // transformation matrix
  end

  for i=1:nc
    [xxx,q(i)]=max(T(:,i));             // order vector
  end
```

```
endfunction
```

– **col2xt.sci**

This function computes a vector of values of a multivariate discrete variable from their code. It is an inverse function to `xt2col.sci`

```
function x=col2xt(i,b)
  // x=col2xt(i,b)   generates a discrete rgegression vector xt
  //                  with a base b that corresponds to the
  //                  i-th column of a table of the discrete model
  // it is based on the relation
  //       i=b(n-1)b(n-2)...b(1)(x(n)-1)+...+b(1)(x(2)-1)+x(1)

  if isscalar(b),
    n=fix(log(i+1)/(log(b)))+1;
    b=b*ones(1,n);
  else
    n=length(b);
  end
  if i>prod(b)
    disp('ERROR: The row number is too big')
    return
  end
  i=i-1;
  for j=1:n
    i=i/b(n-j+1);
    x(n-j+1)=round((i-fix(i))*b(n-j+1)+1);
    i=fix(i);
  end
  x=x(:)';
endfunction
```

– **fnorm.sci**

The function performs probabilistic normalization of a vector or matrix of nonnegative numbers.

```
function fn=fnorm(f,i)
  // fn=fnorm(f,i)   normalization of probabilistic table
  // fn  normalized table
  // f   table
```

```
  // i   direction i=1 norm colunms, i=2 norm rows

  if argn(2)==1,
    sf=sum(f);
    fn=f/sf;
  else
    [m n]=size(f);
    if i==1
      f1=sum(f,1);
      fn=f./(ones(m,1)*f1);
    else
      f2=sum(f,2);
      fn=f./(f2*ones(1,n));
    end
  end
endfunction
```

– **genThs.sci**

The function generates parameters of a categorical model - i.e. a table, suitably normalized that can generate discrete data which are more or less close to a deterministic case (one value of the generating vector is near to one).

```
function th=genThs(n1,n2,a)
  // Generation of discrete model for simulation
  c=zeros(n1,1); c(1)=1;
  th=[];
  for j=1:n2
    th(:,j)=mixData(c);
  end
  th=fnorm(th+a*rand(n1,n2,'u'),1);
endfunction
```

– **mixData.sci**

The function commutes the rows of a data matrix to avoid similar data in the beginning of estimation.

```
function [dd,mm]=mixData(d)
  // reorder data records
  // - variables in columns
```

```
  [nr,nc]=size(d);
  if nr<nc, d=d'; end
  n=size(d,1);
  s=1:n;
  mm=samwr(n,1,s');
  dd=d(mm,:);
endfunction
```

- pfXY.sci

This function construct empirical conditional probability function $f(y|x)$ from data vectors x and y.

```
function [f,t]=pfXY(x,y,kx,ky)
  // probab.fc. for two variables x and y
  // x = 0,1..kx,  y = 1,2..ky !!!
  // x,y       variables x and y
  // kx,ky     max. value of x and y
  if argn(2)<4, ky=max(x); ky=max(y); end
  if argn(2)<3, kx=max(x); kx=max(y); end
  v1=vals(x);
  v2=vals(y);
  t=1e-8*ones(kx+1,ky);
  [tb,xv,yv]=table(x+1,y);            // table T(x,y)
  t(v1(1,:)+1,v2(1,:))=tb;
  f=fnorm(t,1);               // normalization in columns
endfunction
```

– pfY.sci

This function construct empirical probability function $f(y)$ from data vector y.

```
function [f,h]=pfY(d,m)
  // probab. func. from y = 1,2..m
  if argn(2)<2, m=max(d); end
  if isempty(d), h=ones(1,m);
  else
  v=vals(d);
  h=zeros(1,m);
  h(v(1,:))=v(2,:);
  end
```

```
  f=fnorm(h);
endfunction
```

– **printm.sci**

The function prints a matrix on the screen.

```
function printm(a,b)
  // concise print of matrix
  if argn(2)<2, b='res'; end
  [m,n]=size(a);
  disp(b+' = ');
  for i=1:m
    for j=1:n
      printf('  %5.3f',a(i,j));
    end
    printf('\n');
  end
endfunction
```

– **randu.sci**

The function generates a value from the uniform distribution on $(0, 1)$.

```
function y=randu(m,n)
  // uniform distribution (0,1)
  if argn(2)<1, m=1; n=1; end
  if argn(2)<2, n=1; end
  y=grand(m,n,'unf',0,1);
endfunction
```

– **sampBin.sci**

The function generates a value from the binomial distribution.

```
function x=sampBin(p,N)
  // generation of binomial values
  // x    values
  // p    probability in Bernoulli trial
  // N    number of Bernoulli trials
  x=0;
```

```
  for i=1:N
    x=x+(rand(1,1,'u')<p)+0; // Bernoulli trials
  end
endfunction
```

– **sampBin1.sci**

The function generates a value from the modified binomial distribution.

```
function x=sampBin1(p,N1)
  // sampling from modified binomial dist.
  // x    values 1,2,..,N1
  // p    Bernoulli probability
  // N1   number of values (= maximum value)
  b=binomial(p,N1-1);
  x=sum(cumsum(b)<rand(1,1,'u'))+1;
endfunction
```

– **sampCat.sci**

The function generates a value from the categorical distribution.

```
function x=sampCat(p,xs)
  // sample from categorical dist. with par. p
  // for model with y vertical and x horizontal
  // y=1 | p1
  // y=2 | p2
  // y=3 | p3
  // p    is the column k=xt2col([x1 x2],[2 2])
  // Example: x(t)=sampCat(thE(:,k))
  // Remark: can generate also dynamic pointer
  //         c(t)=dampCat(p,c(t-1))
  if argn(2)<2
    p=p(:);
    xs=1;
  end
    x=sum(cumsum(p(:,xs))<rand(1,1,'u'))+1;
endfunction
```

– **table.sci**

The function generates a frequency table fro two discrete data vectors.

```
function [T,x1,y1]=table(x,y)
  // T=table(x,y)   contingency table T(nx,ny)
  // x,y       data

  if argn(2)<2
    y=x(:,2);
    x(:,2)=[];
  end

  nx = length(x);
  ny = length(y);
  nn = min(nx,ny);
  xx = vals(x);
  x1=xx(1,:);
  yy = vals(y);
  y1=yy(1,:);
  mx = size(xx,2);
  my = size(yy,2);
  T = zeros(mx, my);
  for t = 1:nn
    i=find(x1==x(t));
    j=find(y1==y(t));
    T(i, j) = T(i, j) + 1;
  end
endfunction
```

– **vals.sci**

This function gives a two-row table where the first row contains different values of a discrete data vector and the second one their frequencies.

```
function [h,f]=vals(a)
  // [h f]=vals(a)  find different nalues of a
  //                and their frequencies
  // h       values and frequencies  [vals;abs_freq]
  // f       relative frequencies

  a=a(:)';
  b=gsort(a,'g','i');
  [v,m]=unique(b);
  dm=diff(m);
```

```
  n1=length(b)+1;
  n=[dm n1-m($)];
  f=n/sum(n);
  h=[v(:)';n];

  if sum(n)~=max(size(a))
    disp('Error: in vals.sci')
    return
  end
endfunction
```

– **xt2col.sci**

This function constructs codes for a multivariate discrete variable. The coding works by indexing where the rightmost index changes with the highest frequency.

```
function i=xt2col(x,b)
  // i=xt2col(x,b)  i is the column number of a model table with
  //                 the regression vector xt with the base b;
  //                 elements of x(i) are 1,2,...,nb(i)
  // it is based on the relation
  //     i=b(n-1)b(n-2)...b(1)(x(n)-1)+...+b(1)(x(2)-1)+x(1)

  n=length(x);
  bb=b(2:n);
  bb=bb(:)';
  b=[bb 1];
  i=0;
  for j=1:n
    i=(i+x(j)-1)*b(j);
  end
  i=i+1;
//  pause
endfunction
```