

# Useful Transformations

Paul L. Fackler\*

January 27, 2007

These notes describe a number of ways by which a parameter vector (or matrix)  $\beta$  can be expressed as a function of an unrestricted parameter vector  $\theta$  in such a way that any value of  $\theta$  produces a  $\beta$  with a desired characteristic and such that any  $\beta$  with that characteristic can be obtained with some value of  $\theta$ . Alternatively, it may be useful to have a transform that defines  $\theta$  on a simple bounded region. This can facilitate direct search routines (grid or randomized searches).

## Parametric Linear Restrictions

For example, in many situations, one wants to restrict a parameter to be nonnegative. A simple transform to accomplish this is to let  $\beta = \exp(\theta)$ , where  $\theta$  is unrestricted. The value  $\beta = 0$  is obtained as  $\theta \rightarrow -\infty$ . In statistical estimation, this can accomplish two desirable goals. First, it may allow unrestricted optimization to be used with respect to  $\theta$ . Second, it may result in a parameter space of lower dimension and/or one that is bounded. In both cases, searches over the parameter space may be facilitated.

In addition to  $\beta \geq 0$ , the following restrictions on  $\beta$  are described:  $R\beta = r$ ,  $\beta$  is positive-semi definite,  $\beta \succeq 0$  with  $\beta^\top \underline{1} = 1$ ,  $\beta^\top \beta = 1$  and  $\beta$  is orthogonal (i.e.,  $\beta$  is  $n \times n$  and  $\beta^\top \beta = I_n$ ).

---

\*The author is an associate professor at North Carolina State University.

Mail: Paul L. Fackler

Department of Agricultural and Resource Economics

NCSU, Box 8109

Raleigh NC, 27695, USA

e-mail: [paul\\_fackler@ncsu.edu](mailto:paul_fackler@ncsu.edu)

Web-site: <http://www4.ncsu.edu/~pfackler/>

© 2004, Paul L. Fackler

Linear restrictions are perhaps the most common for of equality constraints. Generally these are written in direct form:  $R\beta = r$ , where  $R$  is  $m \times n$ . The number of restrictions this imposes on  $\beta$  is equal to  $p = \text{rank}(R)$  (which may be less than  $m$  if there are redundant constraints). We would like to express  $\beta$  as a function of free parameters of the form  $\beta = Q\theta + q$ . The constraints are now written as  $RQ\theta + Rq = r$ . In order for  $\beta$  to satisfy the constraints for arbitrary  $\theta$ , it must be the case that  $Q$  and  $q$  are chosen such that  $RQ = 0$  and  $Rq = r$ . There is not a unique way to chose  $Q$  and  $q$  but one method that will always work is to compute the decomposition

$$R^\top = [ V_1 \quad V_2 ] \begin{bmatrix} U \\ 0 \end{bmatrix}$$

where  $V$  is orthogonal, with  $V_1$   $n \times p$  and  $U$  is  $p \times m$  (this is the so-called QR decomposition of  $R^\top$ ). Thus setting  $Q = V_2$  and  $q = V_1(UU^\top)^{-1}Ur$  accomplishes the desired goal. A MATLAB implementation that assumes  $\text{rank}(R) = m$  is given in the appendix (`lnconst`).

If there is a question about the rank of  $R$ , it may be better to use the singular value decomposition (SVD):

$$R = [ U_1 \quad U_2 ] \begin{bmatrix} S_1 & 0 \\ 0 & 0 \end{bmatrix} [ V_1 \quad V_2 ]^\top = U_1 S_1 V_1^\top$$

where  $U$  and  $V$  are both orthogonal and  $S$  is diagonal. Set  $Q = V_2$  and  $q = V_1 S_1^{-1} U_1^\top r$ . The SVD is the preferred method for determining the rank of a matrix numerically, with values of the diagonal of  $S$  (the so-called singular values of  $R$ ) treated as 0 if they are less than some tolerance times the maximal singular value (a useful default tolerance is  $2^{-52}n$ ).

## Positive Definite Restrictions

Another very common restriction is that a matrix of parameters must be positive semi definite. Positive semi-definite matrices arise in estimation of variance matrices. This is most easily accomplished by parameterizing in terms of a lower triangular matrix  $L$  and using  $LL^\top$ , which is guaranteed to be positive semi-definite and positive definite if the diagonal elements of  $L$  are all non-zero. Furthermore, the signs of the diagonal elements of  $L$  are arbitrary and hence can be normalized to be non-negative. Any  $n \times n$  positive semi definite matrix can be obtained with  $n(n+1)/2$  free parameters.

An alternative is to use  $LDL^\top$  where  $L$  is a lower triangular matrix with unit diagonal and  $D$  is a diagonal matrix.  $L$  contains  $n(n-1)/2$  free parameters and  $D$  contains  $n$ . To ensure that  $LDL^\top$  is positive-definite, the diagonal elements of  $D$  must be positive.<sup>1</sup>

## Mappings to Unbounded Regions

Direct search algorithms often are used to search a bounded region for an optimum. It is, therefore, sometimes useful to map an interval to an unbounded region. The search over the bounded region thus effectively becomes a search over an unbounded region. The main difficulty with this is the search should be concentrated in an area that is likely to contain the optimum.

A simple way to think of a useful transformation is to consider functions that map an unbounded domain into a bounded one, which for simplicity can be normalized to the interval  $[0,1]$ . Probability theory is full of such mappings and it is convenient to pick one for which the mapping in either direction has a closed form expression.

For example, suppose one is interested in searching the whole real line, i.e.,  $x \in (-\infty, \infty)$ . Define  $u \in [0, 1]$  and use the mapping

$$x = c \ln \left( \frac{u}{1-u} \right)$$

The derivative of the mapping is  $dx/du = c(1/u - 1/(1-u))$ . The mapping is concentrated around  $x = 0$ . The parameter  $c$  determines how diffuse the mapping is, with larger values of  $c$  making larger values of  $x$  more likely (if  $u$  is selected randomly, i.e., if  $u \sim \text{Uniform}(0, 1)$ ).

For obtaining nonnegative values one can use

$$c \left( \frac{u}{1-u} \right)^\beta$$

with derivative

$$\beta c \left( \frac{u}{1-u} \right)^{\beta-1} \frac{1}{(1-u)^2}$$

---

<sup>1</sup>The operations  $\text{vech}$  and  $\text{vech}^{-1}$  are very useful in working with triangular matrices.  $\text{vech}$  extracts the elements of a square matrix on or below the diagonal and stacks each column as a vector.  $\text{vech}^{-1}$  reverses this process (alternatively  $\text{vech}^{-1}$  could create a symmetric matrix from a vector).

The mapping is concentrated around  $x = c$ . Specifically, half of the values will be below  $c$  if  $u$  is selected randomly. A similar transformation for non-negative values is

$$x = c \ln \left( \frac{1+u}{1-u} \right)$$

The derivative of the mapping is  $dx/du = c(1/(1+u) - 1/(1-u))$ .

## Restrictions to a Simplex

Another restriction is that a vector be composed of non-negative elements that sum to one. Such vectors arise when they are interpreted as shares or as probability vectors. A simple way to accomplish this is to start with an  $n$ -vector  $\theta$  composed of numbers on  $[0,1]$ . Set  $\beta_1 = \theta_1$ ,  $\beta_k = \theta_k(1 - \sum_{i=1}^{k-1} \beta_i)$  for  $1 < k < n$  and  $\beta_{n+1} = (1 - \sum_{i=1}^n \beta_i)$ . This transform takes values of  $\theta$  on  $[0, 1]^n$ . If one wants unrestricted values of  $\theta$ , a mapping from  $R^n$  to  $[0, 1]^n$  can be used applied first. For example, let  $f(x) = \tan^{-1}(x)/\pi + 1/2$  and define  $\beta_1 = f(\theta_1)$  and  $\beta_k = f(\theta_k)(1 - \sum_{i=1}^{k-1} \beta_i)$  for  $1 < k < n$ .

One problem with this transform is that it does not cover the space of interest evenly. For example, if you took a grid of points in  $[0, 1]^n$  and used the above mapping to obtain points in  $[0, 1]^{n+1}$  that sum to one, these points would not be uniformly distributed in the new space. In particular, the values in the lowest dimensions would tend to be larger than the values in the higher dimensions. This is not necessarily bad but it might be, especially for global optimization algorithms.

As an alternative consider the probability associated with points that are uniformly distributed on a simplex. The marginal CDF for any specific element is given by

$$Prob(X \leq x) = 1 - (1 - x)^n$$

Furthermore the conditional probability is

$$Prob \left( X_i \leq \frac{x_i}{1 - \sum_{j=1}^{i-1} x_j} \middle| X_1 = x_1, \dots, X_{i-1} = x_{i-1} \right) = 1 - (1 - x_i)^{n+1-i}$$

This leads to the following algorithm. Set  $\beta_1 = 1 - \theta_1^{1/n}$  and  $s = 1 - \beta_1$ . Then, for  $j = 2, \dots, n$  set  $\beta_j = s(1 - \theta_j^{1/(n+1-j)})$  and  $s = s - \beta_j$ . Finally set  $\beta_{n+1} = s$ . A MATLAB implementation (`square2simplex`) appears in the appendix.

## Normalization Restrictions

In many situations one has an  $n$ -vector that is of arbitrary scale. A common normalization is to arbitrarily set one of the values equal to 1. Unfortunately, this can be problematic if the size of the normalized parameter is small relative to other parameter values. This strategy also does not allow one to set the normalized parameter to 0 and hence may impose a restriction on the behavior of a model.

Instead, suppose that one normalizes by making the sum of the squared values equal 1. This is less restrictive because it imposes only that one of the  $n$  values of  $\beta$  is non-zero, rather than that a particular one is non-zero. If there are  $n + 1$  model parameters, the single restriction means that there are  $n$  free parameters. Let  $\theta$  be the  $n$  vector of free parameters and  $\beta$  the  $n + 1$ -vector of model parameters. Set

$$\beta_i = \cos(\theta_i) \prod_{j=1}^{i-1} \sin(\theta_j)$$

and

$$\beta_{n+1} = \prod_{j=1}^n \sin(\theta_j)$$

The mapping is periodic and every point on the hypersphere can be obtained from a point in the space  $\theta \in [0, \pi]^{n-1} \times [0, 2\pi]$  ( $\theta \in [0, \pi]^{n-1}$  will always yield a positive value for  $\beta_n$ ). If the value of  $\beta$  is only determined up to a sign normalization, then  $\theta \in [0, \pi]^n$  is sufficient to span the relevant space of  $\beta$ . Furthermore, any nonnegative point on the hypersphere can be obtained with a point  $\theta \in [0, \pi/2]^n$ .

It is worth noting that sign changes in  $\beta$  are obtained by  $180^\circ$  rotations in  $\theta$ , with their direction of rotation altered for the last element. Defining

$$\tilde{\theta}_i = \begin{cases} \pi - \theta_i & \text{if } i < n \\ \theta_i \pm \pi & \text{if } i = n \end{cases}$$

yields  $\tilde{\beta} = -\beta$ . Notice that the sign transformations map  $[0, \pi]^{n-1} \times [0, 2\pi]$  into itself so long as the last element subtracts  $\pi$  when  $\theta_n > \pi$  and adds  $\pi$  otherwise.

The derivative of the mapping is easily computed:

$$\frac{\partial \beta_i}{\partial \theta_j} = \begin{cases} 0 & \text{for } i < j \\ -\prod_{k=1}^j \sin(\theta_k) & \text{for } i = j \\ \frac{\cos(\theta_j)}{\sin(\theta_j)} \beta_i & \text{for } i > j \end{cases}$$

(with  $\partial \beta_i / \partial \theta_j = 0$  if  $\theta_j = 0$ ).

For some problems the  $\beta$  is determined only up to a sign normalization. In this case all relevant values can be obtained by defining  $\theta \in [-\pi/2, \pi/2]^n$ , which normalizes the lead term to be positive. This is implemented in the MATLAB function `r2sphere` in the appendix.

The inverse of this mapping can also be defined:

$$\theta_i = \text{sign} \left( \beta_{i+1} \prod_{j=1}^{i-1} \sin(\theta_j) \right) \cos^{-1} \left( \frac{\beta_i}{\prod_{j=1}^{i-1} \sin(\theta_j)} \right)$$

for  $i < n$  and

$$\theta_{n-1} = \sin^{-1} \left( \frac{\beta_n}{\prod_{j=1}^{n-2} \sin(\theta_j)} \right)$$

(see appendix function `sphere2r`).

This transform does not map uniformly from the unit hypercube to the unit sphere. In particular, it causes a set of points that is random in the hypercube to map to a set of points with the low dimensional values more likely to be large.

If a uniform mapping is desirable, one must find the probability distribution that is uniform on a hyper-sphere. It can be shown that the desired conditional probability is

$$\text{Prob} \left( X_i \leq \frac{x_i}{\sqrt{1 - \sum_{j=1}^{i-1} x_j^2}} \mid X_1 = x_1, \dots, X_{i-1} = x_{i-1} \right) = \text{sign}(v_i) \sqrt{B(|v_i|; 1/2, (n+1-i)/2)}$$

where  $v_i = (x_i + 1)/2$  and  $B(x; a, b)$  is the CDF of the Beta distribution. An algorithm that maps  $\theta \in [-1, 1]^n$  onto the unit sphere in  $n + 1$  dimensions can be obtained by setting

$$\beta_i = \text{sign}(\theta_i) \sqrt{\left( 1 - \sum_{j=1}^{i-1} \beta_j^2 \right) B^{-1}(|\theta_i|; 1/2, (n+1-i)/2)}$$

for  $i = 1, \dots, n$  and

$$\beta_{n+1} = \sqrt{1 - \sum_{j=1}^n \beta_j^2}$$

Notice that this imposes the normalization that the last element is always non-negative, i.e., that the mapping is to the half hyper-sphere. Reversing the order of generation will lead to the sign normalization of the first element, if desired.

The difficulty with this method is that closed form expressions for

$$\sqrt{B^{-1}(x; 1/2, j/2)}$$

don't exist except for  $j = 1$  and  $j = 2$ :

$$\sqrt{B^{-1}(x; 1/2, 1/2)} = \sin\left(\frac{\pi}{2}x\right)$$

and

$$\sqrt{B^{-1}(x; 1/2, 2/2)} = x$$

For other values of  $j$  the inverse must be found numerically using the explicit expression for the CDF. The CDF can be expressed in a closed, recursive, form. It is, however, unsuited for numerical work as the recursion is unstable and expensive to evaluate for large values of  $j$ .<sup>2</sup> A MATLAB implementation (`square2sphere`) appears in the appendix (this implementation maps  $[0, 1]^n$  into the  $n + 1$  dimensional unit hypersphere).

If computation of  $\sqrt{B^{-1}}$  causes unacceptable slowing of a procedure, an approximation can be used. The approximation

$$\sqrt{B^{-1}(1/2, i/2)} \approx \frac{\sum_{j=1}^m a_{ij} x^j}{1 + \left(\sum_{j=1}^m a_{ij} - 1\right) x}$$

---

<sup>2</sup>The derivative of the mapping involves the complete Beta function, which follows the recursion

$$B(1/2, i/2) = B(1/2, (i-2)/2) \frac{i-2}{i-1}$$

with  $B(1/2, 1/2) = \pi$  and  $B(1/2, 1) = 2$ .

does a good job, even for low values of  $m$  (5-9 appear quite good) if the  $a_{ij}$  values are picked appropriately. As these values can be precomputed, the approximation and thus the transform can be evaluated quickly. For the purposes of defining a transformation it enough that uniformity be approximately preserved by the transformation; hence a high degree of accuracy is not required.

An alternative is the use the Kumaraswamy distribution: to approximate the Beta. The density of this distribution is

$$f(x) = abx^{a-1}(1-x^a)^{b-1}$$

with associated CDF

$$F(x) = 1 - (1 - x^a)^b$$

and inverse CDF

$$F^{-1}(u) = (1 - (1 - u)^{1/b})^{1/a}$$

A simple way to obtain an approximation is by matching the first two moments of the distributions. The uncentered moments for the Beta( $\alpha, \beta$ ) are

$$\mu_i = \frac{\alpha(\alpha + 1) \dots (\alpha + i - 1)}{(\alpha + \beta)(\alpha + \beta + 1) \dots (\alpha + \beta + i - 1)}$$

and for the Kumaraswamy( $a, b$ ) are

$$\mu_i = \frac{b\Gamma(b)\Gamma(1 + i/a)}{\Gamma(1 + i/a + b)}$$

## Rank Restrictions

Suppose one were interested in parameterizing a model in such a way that restrictions could be placed on the rank of a matrix. The rank is equal to the number of non-zero eigenvalues but it can be a bit tricky to parameterize a model in term of eigenvalues and eigenvectors because these can come in pairs of complex conjugates. It is more convenient to work with a singular value decomposition, in which any matrix can be decomposed into

$$A = USV^\top$$

where  $U$  and  $V$  are both orthogonal and  $S$  is diagonal with  $S \geq 0$ . The diagonal elements of  $S$  are the unique singular values of  $A$  and the number of non-zero singular values equals the rank of  $A$ .

The question then is how to parameterize the matrices  $U$  and  $V$  to ensure that they are orthogonal. First, count the number of restrictions. Suppose the first column is an arbitrary length 1 vector, and hence has 1 restriction. The second column must be length 1 and orthogonal to the first. Hence there are two restrictions on the second column. By similar reasoning, there are  $j$  restrictions on the  $j$ th column, making a total of  $n(n+1)/2$  restrictions. The number of free parameters is thus  $n(n-1)/2$ .

Essentially the most straightforward method of creating an arbitrary orthogonal matrix is to begin with a order  $N$  identity matrix. Then set the first column of  $U$  equal to an arbitrary length 1 vector by mapping an  $n-1$ -vector  $\theta_1$  onto the unit hypersphere (as explained above) and orthogonalize the remaining columns with respect to the new first column. Then multiply columns 2 through  $n$  by an arbitrary length 1  $n-2$  vector, replacing column 2 with the resulting vector and orthogonalize the remaining columns with respect to the first two. Continue this process  $n-1$  times.

Specifically, initialize  $A = I_n$ . For each  $i$  from 1 to  $n-1$ , let  $\alpha$  be a length 1  $n$ -vector with zeros in the first  $i-1$  elements and define the recursion

$$U \leftarrow U - Uuu^T/\beta$$

for some  $n$ -vector  $u$  and scalar  $\beta$ .

If we pick

$$u_k = \begin{cases} \text{sign}(\alpha_i)(1 + |\alpha_i|) & \text{for } k = i \\ \alpha_k & \text{for } k \neq i \end{cases}$$

and

$$\beta = 1 + |\alpha_i|$$

it is easily verified that if  $A$  is orthogonal before each step, it is orthogonal after each step. Furthermore, it is easily verified that the  $i$ th column of the updated  $A$  is

$$Ae_i = -\text{sign}(\alpha_i)\alpha$$

As we desire this to equal  $\alpha$ , we reverse the sign of this column if  $\alpha_i > 0$ .

After  $n - 1$  iterations we have  $n$  columns, each of length 1 and each picked to be an arbitrary element in the null space of the preceding columns. Thus we have an arbitrary orthogonal matrix.

Actually, there is still a sign issue in regards of the last column. Multiplying the last column by  $-1$  also results in a valid orthogonal matrix but one that will not be found by this method. Furthermore, if the values of  $\theta$  all lie in  $[0, \pi]$  an implicit sign normalization will have been imposed because the last values of the  $\alpha$  vectors will be positive.

In practice this may not matter and, in fact, some sign normalization may be useful. Consider the problem of generating an arbitrary matrix  $A$  from decomposition  $A = USV^\top$ , where  $U$  and  $V$  are orthogonal and  $S$  is diagonal. This is like a singular value decomposition except the diagonal values of  $S$  can be negative. The sign restrictions imposed in generating  $U$  and  $V$  are irrelevant. To see this, consider the SVD of  $A = \tilde{U}\tilde{S}\tilde{V}^\top$ . For some free parameters we can obtain  $U = \tilde{U}D_u$  and  $V = \tilde{V}D_v$ , where  $D_u$  and  $D_v$  are diagonal matrices with diagonal elements equal to either 1 or  $-1$ . If  $S$  is defined as  $S = D_uD_v\tilde{S}$  it is clear that  $USV^\top = \tilde{U}\tilde{S}\tilde{V}^\top = A$ .

## Appendix: MATLAB Code

```
% LINCONST Converts linear constraints from direct to parametric form
% USAGE
%   [P,q]=linconst(R,r);
% INPUTS
%   R : mxn matrix
%   r : mx1 vector
% OUTPUTS
%   P : nx(n-m) matrix
%   q : nx1 vector
%
% A constraint of the form  $Rx=r$  is converted to one of the form
%  $x=Py+q$ , such that for any  $(n-m)\times 1$  vector  $y$ ,  $R(Py+q)=r$ .
%
% It is required that  $\text{rank}(R)=m$ . No check is made but method will return
% infs or NaNs if this is not true.
%
% P and q are not unique. This function returns a P with  $P'P=I$ .

function [P,q]=linconst(R,r)
    [m,n]=size(R);
    [V,U]=qr(R');
    P=V(:,m+1:end);
    q=V(:,1:m)*(U(1:m,1:m)\r);
```

```

% SQUARE2SIMPLEX Maps points in the unit square to points in the unit simplex
% The mapping preserves uniformity. If u is U(0,1) then v is uniformly
% distributed on the unit simplex
% USAGE
%   v=square2simplex(u);
% INPUT
%   u : dxn matrix of points in [0,1]^d
% OUTPUTS
%   v : (d+1)xn matrix of points on [0,1]^(d+1) with sum(v)=1
%   dv : dv/du (n=1 for this option) (d+1)xd Jacobian matrix

function [v,dv]=square2simplex(u) \
[d,n]=size(u);

if nargin>1
    if n>1, error('u must be a column vector to compute Jacobian'); end
    z0=(1-u).^(1./(d:-1:1)'-1);
    z1=z0.*(1-u);
    v=[1-z1;0];
    s=1-v(1);
    for i=2:d
        v(i)=v(i)*s;
        s=s-v(i);
    end
    v(d+1)=s;
    dv=zeros(d+1,d);
    ss=1-cumsum([0;v]);
    for j=1:d
        s=ss(j)*z0(j)/(d+1-j);
        dv(j,j)=s;
        for i=j+1:d
            dv(i,j)=(z1(i)-1)*s;
            s=s+dv(i,j);
        end
        dv(d+1,j)=-s;
    end
end

```

```
else
    v=zeros(d+1,n);
    vi=1-(1-u(1,:)).^(1./d);
    s=1-vi;
    v(1,:)=vi;
    for i=2:d
        vi=(1-(1-u(i,:)).^(1./(d-i+1))).*s;
        v(i,:)=vi;
        s=s-vi;
    end
    v(d+1,:)=s;
end
```

```

% R2SPHERE Maps points in  $R^n$  onto the unit sphere in  $R^{n+1}$ 
% USAGE
% [beta,dbeta]=r2sphere(theta);
% INPUT
%   theta : an n-vector
% OUTPUTS
%   beta   : an (n+1)-vector such that beta'*beta=1
%   dbeta  : (n+1)xn matrix of partial derivatives (d beta/d theta)
%
% Note the mapping is periodic with period  $2\pi$ 
% The points in  $[-\pi/2,\pi/2]^n$  map onto a half sphere
%   with positive 1st coordinate

function [beta,dbeta]=r2sphere(theta)
c=[cos(theta);ones(1,size(theta,2))];
s=[ones(1,size(theta,2));sin(theta)];
S=cumprod(s);
beta=c.*S;
if nargin>1
    n=size(theta,1);
    d1=[diag(S(2:end));zeros(1,n)];
    % to avoid divide by 0 and consequent NaNs
    s(s==0)=eps;
    d2=tril(beta*(c(1:end-1)./s(2:end))',-1);
    dbeta=d2-d1;
end

```

```

% SPHERE2R Maps points on the unit sphere in  $R^{\{n+1\}}$  into  $R^n$ 
% USAGE
%   beta=r2sphere(alpha);
% INPUT
%   alpha : an n+1-vector with alpha'alpha=1
% OUTPUTS
%   beta  : an n-vector normalized to  $[-\pi/2,\pi/2]^n$ 
%
% SPHERE2R is the inverse of R2SPHERE, i.e., given beta in  $[-\pi/2,\pi/2]^n$ 
%   sphere2r(r2sphere(beta)) returns beta
% and given a point alpha with alpha'alpha=1
%   r2sphere(sphere2r(alpha)) returns alpha

function beta=sphere2r(alpha)
K=length(alpha);
beta=zeros(K-1,1);
beta(1)=sign(alpha(2))*acos(alpha(1));
for i=2:K-2
    tmp=prod(sin(beta(1:i-1)));
    if tmp==0, return; end
    beta(i)=sign(alpha(i+1))*sign(tmp)*acos(alpha(i)/tmp);
end
tmp=alpha(K)/prod(sin(beta(1:K-2)));
beta(K-1)=asin(tmp);

```

```

% SQUARE2SPHERE Maps points in the unit square to points in the unit sphere
% The mapping preserves uniformity. If u is U(0,1) then v is uniformly
% distributed on the unit sphere
% USAGE
%   v=square2sphere(u);
% INPUT
%   u : (d-1)xn matrix of points in [0,1]^(d-1)
% OUTPUTS
%   v : dxn matrix of points with sum(v.^2)=1
%   dv : dv/du (n=1 for this option) dx(d-1) Jacobian matrix

```

```
function [v,dv]=square2sphere(u)
```

```
persistent betavals
```

```

d=size(u,1)+1;
n=size(u,2);

if d==1, v=ones(1,n); return; end % trivial case

v=zeros(d,n);
uu=2*u-1;
b=zeros(d-1,n);
s=ones(d,n);
for i=d-1:-1:2
    b(i,:)=icdfbeta(abs(uu(i,:)),0.5,i/2);
    vi2=b(i,:).*s(i+1,:);
    s(i,:)=s(i+1,:)-vi2;
    v(i+1,:)=sign(uu(i,:)).*sqrt(vi2);
end
b(1,:)=sin(uu(1,:)*(pi/2)).^2;
vi2=b(1,:).*s(2,:);
s(1,:)=s(2,:)-vi2;
v(2,:)=sign(uu(1,:)).*sqrt(vi2);
v(1,:)=sqrt(s(1,:));

```

```

% Compute the Jacobian if requested
if nargin>1
    if n>1, error('u must be a column vector to compute Jacobian'); end
    % compute the Beta function values the first time called.
    % Uses the recursion
    % Beta(1/2,i/2)=Beta(1/2,(i-2)/2)(i-1)/(i-1)
    if length(betavals)<d-1
        betavals=zeros(d-1,1);
        betavals(1)=pi;
        betavals(2)=2;
        for j=3:d-1, betavals(j)=betavals(j-2)*((j-2)/(j-1)); end
    end

    dv=zeros(d,d-1);
    for j=d:-1:2
        dv(j,j-1)=sign(uu(j-1))*s(j)./v(j)*betavals(j-1).* ...
            sqrt(b(j-1,:))./(1-b(j-1,:))^(j-1/2-1);
        for i=j-1:-1:1
            dv(i,j-1)=-v(i)/s(i)*sum(v(i+1:end).*dv(i+1:end,j-1));
        end
    end
end
end

```