Appendix D

Functions for Simulating Data by Using Fleishman's Transformation

Contents

D.1 Overview of Fleishman's Cubic Transformation Method

This supplemental material describes SAS/IML functions that implement Fleishman's cubic transformation method (Fleishman 1978). Fleishman's method is used in Section 16.8 of *Simulating Data with SAS* (Wicklin 2013) to simulate data from a univariate distribution that has specified values for the skewness and kurtosis. For feasible values of the skewness and kurtosis, you can find coefficients of a cubic polynomial that transform the standard normal distribution into a distribution with the specified moments.

The functions are divided into the following general categories:

- Functions that implement Newton's method. These functions find the cubic coefficients that correspond to a distribution with target values for the skewness and kurtosis.
- A function that computes the sample moments of data and returns the Fleishman coefficients for those empirical moments.
- A function that simulates random draws from the Fleishman distribution by sampling from a standard normal distribution and transforming each variate by a cubic polynomial.

The implementation of the Fleishman method uses the MOMENTS function, which is described in Section A.10 in Appendix A, "A SAS/IML Primer." The following statements load the MOMENTS function into a SAS/IML session:

```
%include "C:\<path>\Utility.sas";
proc iml;
load module=(Moments);
```

D.2 Computing Parameters for Given Skewness and Kurtosis Values

Fleishman's method applies a cubic transformation to a standard univariate normal distribution to obtain a nonnormal distribution with known moments. If Z is a standard normal random variable, the new nonnormal random variable is defined by the polynomial

$$Y = -c_2 + c_1 Z + c_2 Z^2 + c_3 Z^3$$

Given the values of the cubic coefficients (c_1, c_2, c_3), you can compute the variance, skewness, and kurtosis of the nonnormal distribution in terms of the coefficients. Similarly, you can use root-finding methods to solve the inverse problem: Given feasible skewness and kurtosis values, find the parameters (coefficients) that produce a nonnormal distribution with the specified skewness and kurtosis.

The following function takes cubic coefficients as an input argument, and returns the variance, skewness, and kurtosis of the corresponding nonnormal distribution. The mean of the distribution is zero, and is not returned.

```
/* Compute variance, skewness, and kurtosis from cubic coefficients */
start Fleishman(coef);
   b = coef[1]; c = coef[2]; d = coef[3];
   b2 = b##2; c2 = c##2; d2 = d##2; bd = b#d;
   var = b2 + 6#bd + 2#c2 + 15#d2;              /* variance */
   skew = 2#c#(b2 + 24#bd + 105#d2 + 2);        /* skewness */
   kurt = 24#(bd + c2#(1 + b2 + 28#bd) + d2 #
       (12 + 48#bd + 141#c2 + 225#d2));         /* excess kurtosis */
   return( var // skew // kurt );
finish;
```

You can define two functions that help to solve the inverse problem. The FLFUNC function calls the FLEISHMAN function and subtracts the vector $(1, \gamma_0, \kappa_0)$, where γ_0 is the target skewness, and κ_0 is the target kurtosis. Consequently, if you can find coefficients for which the FLFUNC function is zero, then the corresponding nonnormal distribution has unit variance, skewness γ_0, and kurtosis κ_0.

```
/* Find the root of this function */
start FlFunc(x) global (g_target); /* g_target=(skewness, kurtosis) */
   return ( Fleishman(x) - (1 // g_target[1] // g_target[2]) );
finish FlFunc;
```

The FLDERIV function is the derivative of the FLFUNC function with respect to the parameters (c_1, c_2, c_3). The FLDERIV function is used by Newton's method to solve the inverse problem.

```
/* derivatives of the Fleishman function */
start FlDeriv(x);
   b = x[1]; c = x[2]; d = x[3];
   b2 = b##2; c2 = c##2; d2 = d##2; bd = b#d;
   df1db = 2#b + 6#d;
   df1dc = 4#c;
   df1dd = 6#b + 30#d;
   df2db = 4#c#(b + 12#d);
   df2dc = 2#(b2 + 24#bd + 105#d2 + 2);
   df2dd = 4#c#(12#b + 105#d);
   df3db = 24#(d + c2#(2#b + 28#d) + 48#d##3);
   df3dc = 48#c#(1 + b2 + 28#bd + 141#d2);
   df3dd = 24#(b + 28#b#c2 + 2#d#(12 + 48#bd + 141#c2 + 225#d2)
           + d2#(48#b + 450#d));
   J = (df1db || df1dc || df1dd) //
       (df2db || df2dc || df2dd) //
       (df3db || df3dc || df3dd);
   return( J );
finish FlDeriv;
```

The following function implements Newton's method. It calls the FLFUNC and FLDERIV functions to find a zero of the FLFUNC function. Notice that the skewness and kurtosis are specified by using a global variable, `g_target`.

```
/* Newton's method to find roots of a function.
    You must supply the functions that compute
    the function and the Jacobian matrix.
    Input: x0 is the starting guess
           optn[1] = max number of iterations
           optn[2] = convergence criterion for || f ||
    Output: x contains the approximation to the root */
start Newton(x, x0, optn);
   maxIter = optn[1]; converge = optn[2];
   x = x0;
   f = FlFunc(x);
   do iter = 1 to maxIter while(max(abs(f)) > converge);
      J = FlDeriv(x);
      delta = -solve(J, f);                    /* correction vector */
      x = x + delta;                           /* new approximation */
      f = FlFunc(x);
   end;
   /* return missing if no convergence */
   if iter > maxIter then x = j(nrow(x0),ncol(x0),.);
finish Newton;
```

When you call the NEWTON function, you need to provide an initial guess for the cubic coefficients. The FLEISHMANIC function is a helper function that provides an initial guess. The equations in the FLEISHMANIC function were derived by using polynomial regression.

```
/* Given (skew, kurt), produce an initial guess of the Fleishman
   coefficients to use for Newton's algorithm.  This guess is
   produced by a polynomial regression. */
start FleishmanIC(skew, kurt);
   c = j(3,1);
   c[1] = 0.95357 -0.05679*kurt +          /* c1 = Quad(skew, kurt) */
          0.03520*skew##2 + 0.00133*kurt##2;
   c[2] = 0.10007*skew + 0.00844*skew##3;       /* c2 = Cubic(skew) */
   c[3] = 0.30978 -0.31655*c[1];                /* c3 = Linear(c1)  */
   return (c);
finish;
```

All of the pieces are in place for solving the inverse problem. The following function is a driver function that finds the Fleishman coefficients that produce a distribution with the specified skewness and kurtosis. The FITFLEISHMANFROMSK function first checks that the skewness and kurtosis are in the feasible region (see Section 16.8 of Wicklin (2013)). The function then calls the FLEISHMANIC function to obtain an initial guess for the coefficients. Lastly, the function assigns the `g_target` global variable and calls Newton's method.

```
start FitFleishmanFromSK(skew, kurt) global (g_target);
   /* 1. check that (skew,kurt) is in the feasible region */
   if kurt < -1.13168+1.58837*skew##2 then return({. . . .});
   /* 2. Initial guess for nonlinear root finding */
   x0 = FleishmanIC(skew, kurt);
   optn = {25, 1e-5};   /* maximum iterations, convergence criterion */

   /* 3. Find cubic coefficients (c1, c2, c3) so that
         -c2+c1*Z+c2*Z##2+c3*Z##3 has target (skew,kurt). */
   g_target = skew || kurt;                  /* set global variable */
   run Newton(coef, x0, optn);               /* find (c1, c2, c3)   */
   return( -coef[2] // coef );
finish;
```

You can test the FITFLEISHMANFROMSK function by specifying values for the skewness and kurtosis. The following statements compute the Fleishman coefficients that produce a distribution that has a skewness of 1.15 and an excess kurtosis of 2. The coefficients are shown in Figure D.1.

```
/* test the Fleishman functions */
c = FitFleishmanFromSK(1.15, 2);
print c[rowname=("c0":"c3")];
```

Figure D.1 Fleishman Coefficients That Produce a Distribution with Specified Skewness and Kurtosis

	c
c0	-0.18582
c1	0.9368777
c2	0.1858204
c3	0.0092367

D.3 Fitting Fleishman's Model to Data

Given a set of data, you can use the data to estimate the parameters in a Fleishman distribution. The MOMENTS function is used to obtain the empirical moments for a data vector. Then the FITFLEISHMANFROMSK function is called to return Fleishman coefficients that produce a distribution with the empirical skewness and kurtosis.

```
/* Fit Fleishman model. Given data x, find coefficients
   c = {c0, c1, c2, c3} so that the standardized data is modeled by
   c0+c1*Z+c2*Z##2+c3*Z##3 for Z~N(0,1) */
start FitFleishman(x);
   m = Moments(x);                          /* compute sample moments */
   return( FitFleishmanFromSK(m[3], m[4]) );
finish;
```

D.4 Simulating Data from the Fleishman Distribution

After you fit Fleishman's model to data, you can simulate the data by generating many samples from the model. The RANDFLEISHMAN function returns a matrix of random values from a Fleishman model. This is done by simulating data from a standard normal distribution and transforming the variates by using the cubic coefficients, as follows:

```
/* return N x NumSamples matrix of samples from Fleishman distrib */
start RandFleishman(N, NumSamples, coef);
   /* fill each element of X with sample from Fleishman distribution
      with given coefficients */
   Z = j(N, NumSamples);
   call randgen(Z, "Normal");
   X = coef[1] + Z#(coef[2] + Z#(coef[3]+Z#coef[4]));
   return( X );
finish;
```

For example, suppose that you want to simulate 10,000 values from a distribution that has the same mean, standard deviation, skewness, and kurtosis as the data in the MPG_City variable in the Sashelp.Cars data set. The following statements simulate the data:

```
use Sashelp.Cars;
read all var {MPG_City};
close Sashelp.Cars;

call randseed(12345);
c = FitFleishman(MPG_City);       /* fit model; obtain coefficients */
Y = RandFleishman(10000, 1, c);
Y = mean(MPG_City) + std(MPG_City)*Y;      /* translate and scale Y */

varNames = {"Mean" "Var" "Skew" "Kurt"};
Moments = T(Moments(MPG_City)) // T(Moments(Y));
print Moments[r={"Observed" "Simulated"} c=varNames];
```

Figure D.2 Statistics from Observed and Simulated Data

Moments				
	Mean	**Var**	**Skew**	**Kurt**
Observed	20.060748	27.438924	2.7820718	15.791147
Simulated	20.069402	26.638441	2.8428958	16.652077

Fleishman's method is powerful, but moment-matching is not a panacea. In the current example, you can check that several of the 10,000 simulated values are negative, even though the MPG_City variable represents a positive quantity (miles per gallon). Similarly, several simulated values are greater than 70, even though no vehicles in the original data are that fuel efficient. Lastly, a kernel density estimate of the original data shows minor peaks in density near 25 and 32 miles per gallon. The Fleishman distribution does not capture these peaks. Like many other distributional models, the Fleishman distribution captures only major features in the data density.

D.5 A Multivariate Fleishman Transformation

Vale and Maurelli (1983) extended Fleishman's method to multivariate nonnormal data with a specified correlation, as described in Section 16.11 of Wicklin (2013). The implementation of the Vale-Maurelli algorithm has the following components:

- A function, SOLVECORR, that finds the correlation, r, between two bivariate normal variables X_1 and X_2 such that when you apply Fleishman's cubic transformation to the X_i, the resulting variables have a specified correlation, ρ.
- A function, VMTARGETCORR, that calls SOLVECORR for each pair of variables.
- A function, RANDVALEMAURELLI, that simulates random values from a multivariate nonnormal distribution such that
  - each marginal distribution has a specified skewness and kurtosis
  - the marginal variables have a specified correlation matrix

The SOLVECORR function is implemented as follows. The SAS/IML POLYROOT function is used to find the real root of the cubic polynomial.

```
/*** Vale-Maurelli method of generating multivariate nonnormal data ***/
/* Solve the Vale-Maurelli cubic equation to find the intermediate
   correlation between two normal variables that gives rise to a target
   correlation (rho) between the two transformed nonnormal variables. */
start SolveCorr(rho, coef1, coef2);
   a1 = coef1[1]; a1 = coef2[1]; b1 = coef1[2]; b2 = coef2[2];
   c1 = coef1[3]; c2 = coef2[3]; d1 = coef1[4]; d2 = coef2[4];
   coef = (6*d1*d2) ||
          (2*c1*c2) ||
          (b1*b2+3*b1*d2+3*d1*b2+9*d1*d2) ||
          -rho;
   roots = polyroot(coef); /* solve for zero of cubic polynomial */
   /* roots is a 3x2 matrix of complex roots */
   realIdx = loc(abs(roots[,2]<1e-8)); /* extract the real root(s) */
   r = roots[realIdx,1][1];            /* return smallest real root */
   return (r);
finish;
```

The VMTARGETCORR function calls the SOLVECORR function in a nested double loop. The function returns the intermediate correlation matrix, V.

```
/* Given a target correlation matrix, R, and target values of skewness
   and kurtosis for each marginal distribution, find the "intermediate"
   correlation matrix, V */
start VMTargetCorr(R, skew, kurt);
   V = j( nrow(R), ncol(R), 1);
   do i = 2 to nrow(R);
      ci = FitFleishmanFromSK(skew[i], kurt[i]);
      do j = 1 to i-1;
         cj = FitFleishmanFromSK(skew[j], kurt[j]);
         V[i,j] = SolveCorr(R[i,j], ci, cj);
         V[j,i] = V[i,j];
      end;
   end;
   return (V);
finish;
```

You can call the VMTARGETCORR function as part of the simulation of multivariate data, but it is slightly more efficient to embed the computations into the simulation algorithm. The following function simulates data from a multivariate nonnormal distribution by implementing the Vale-Maurelli algorithm that is described in Section 16.11 of Wicklin (2013). The Fleishman coefficients for each marginal distribution are computed, and they are used to compute the intermediate correlation matrix. The spectral decomposition is used to generate nonnormal data with the specified correlation.

```
/* Simulate data from a multivariate nonnormal distribution such that
   1) Each marginal distribution has a specified skewness and kurtosis
   2) The marginal variables have the correlation matrix R */
start RandValeMaurelli(N, R, skew, kurt);
   /* compute Fleishman coefficients that match marginal moments */
   c = j(ncol(R), 4);
   do i = 1 to ncol(R);
      c[i,] = T( FitFleishmanFromSK(skew[i], kurt[i]) );
   end;
   /* adjust correlation matrix */
   V = j(nrow(R), ncol(R), 1);
   do i = 2 to nrow(R);
      do j = 1 to i-1;
         V[i,j] = SolveCorr(R[i,j], c[i,], c[j,]);
         V[j,i] = V[i,j];
      end;
   end;

   call eigen(D, U, V);      /* eigenvector decomposition: V=U*D*U` */
   F = sqrt(D`) # U;         /* F is a square root matrix for V     */
   X = j(N, ncol(R));
   call randgen(X, "Normal");              /* uncorrelated normals */
   Y = X*F`;                               /* correlated normals   */
   do i = 1 to ncol(R);
      w = Y[,i];             /* apply Fleishman transformation      */
      X[,i] = c[i,1] + w#(c[i,2] + w#(c[i,3] + w#c[i,4]));
   end;
   return(X);
finish;
```

You can test the algorithm by simulating multivariate data. The following statements simulate 10,000 trivariate observations from correlated nonnormal data. The marginal distributions have zero mean and unit covariance, and have specified values for the skewness and kurtosis.

```
/* Find target correlation for given skew, kurtosis, and corr.
   The (skew,kurt) correspond to Gamma(4), Exp(1), and t5. */
skew = {1   2 0};
kurt = {1.5 6 6};
R = {1.0 0.3 -0.4,
     0.3 1.0  0.5,
    -0.4 0.5  1.0 };

/* generate samples from Vale-Maurelli distribution */
call randseed(54321);
X = RandValeMaurelli(10000, R, skew, kurt);
```

You can use the MOMENTS function and the CORR function to verify that the simulated data have moments and correlations that are close to the specified values.

D.6 References

Fleishman, A. (1978), "A Method for Simulating Non-normal Distributions," *Psychometrika*, 43, 521–532.

Vale, C. and Maurelli, V. (1983), "Simulating Multivariate Nonnormal Distributions," *Psychometrika*, 48, 465–471.

Wicklin, R. (2013), *Simulating Data with SAS*, Cary, NC: SAS Institute Inc.