generating correlated RVs with any probability distribution

Steven Brawer
11/2017

introduction

two random variables

We consider generating two correlated random variables (RVs) x_0, x_1. First generate x_0 such that

$$\langle x_0 \rangle = 0, \quad \langle {x_0}^2 \rangle = 1 \tag{A}$$

Let

$$x_1 = \pm x_0 + \varepsilon \tag{B}$$

where

$$\langle \varepsilon \rangle = 0, \quad \langle \varepsilon^2 \rangle = \sigma^2, \quad \langle \varepsilon x_0 \rangle = 0 \tag{C}$$

There is no restriction on the PDFs for x_0, ε. Then the correlation coefficient is

$$\rho \equiv \frac{\langle x_0 x_1 \rangle}{\sqrt{\langle {x_0}^2 \rangle \langle {x_1}^2 \rangle}} \tag{D}$$

It is easy to see that

$$\rho = \frac{\pm 1}{\sqrt{(1+\sigma^2)}} \tag{E}$$

Inversely

$$\sigma^2 = \frac{1}{\rho^2} - 1 \tag{F}$$

Either equation (E) or (F) can be used to generate correlated RVs x_0, x_1 from eq (B).

multiple random variables

We can generalize the above case for N RVs $x_0, x_1, ..., x_{N-1}$. First generate x_0 such that eqs (A) hold. Then set

$$x_i = x_0 + \sum_{\substack{k=0 \\ (k \neq i)}}^{N-1} \varepsilon_{i,k} \qquad \text{(G)}$$

which is a generalization of (B). There is no restriction on the PDF for $x_0, \varepsilon_{i,k}$. For now we assume all signs are positive, so all correlation coefficients will be positive. In the end, one can set $x_i = -x_i$ for any desired values of i to generate negative correlations.

In eq (G) we have the restrictions

$$\varepsilon_{i,i} = 0, \quad \langle \varepsilon_{i,k} \rangle = 0, \quad \langle \varepsilon_{i,k}{}^2 \rangle = \sigma_{i,k}{}^2 \qquad \text{(H)}$$

$$\langle \varepsilon_{i,k} \varepsilon_{j,m} \rangle = 0 \quad \text{when} \quad i,k \neq j,m \qquad \text{(I)}$$

$$\langle x_0 \varepsilon_{i,k} \rangle = 0 \quad \text{for any i,k} \qquad \text{(J)}$$

There is no restriction on the values of the variances $\sigma_{i,k}{}^2$ .First we note that

$$\rho_{i,0} = \rho_{0,i} \equiv \frac{1}{|x_i|} = \frac{1}{\sqrt{1 + \sum_{k=0} \sigma_{i,k}{}^2}} \qquad \text{(K)}$$

where, because of the first of eqs (H)

$$\sigma_{i,i}^2 = 0 \qquad \text{(L)}$$

Next we have

$$\rho_{i,k} \equiv \frac{\langle x_i x_k \rangle}{|x_i||x_k|} \qquad \text{(M)}$$

It turns out, using eqs (H), (I), (J), (K) that

$$\rho_{i,k} = \rho_{i,0} \rho_{k,0} \left(1 + \sigma_{i,k}{}^2\right) \qquad \text{(N)}$$

from which we have the important consistency relation

$$\rho_{i,k} \geq \rho_{i,0}\rho_{k,0} \qquad \text{(O)}$$

If any correlation coefficients are negative, eq (O) holds for absolute values.

Using the above equations, there are two procedures for generating correlated RVs:

(1). Define the variances $\sigma_{i,k}^{\ 2}$, then generate the RVs from eq (G). Eqs (K) and (N) determine the exact values of the correlation coefficients.
(2). Assign some correlation coefficients, compute the variances by inverting eqs (K), (N), then use eq (G) to generate the RVs.

We will demonstrate both approaches, including algorithms and numerical results. There are issues with each technique which must be taken into account in order for the results to be useful (or even feasible).

generate correlation matrix from given variances

A python program for this procedure is given in appendix 1.

This method is quite straightforward. Once the variances are created, then an exact correlation matrix is created from eqs (N), (K) and RVs may be generated from eq (G).

The problem is that, for more than two RVs, it is difficult to know what the correlation coefficients will be for a given set of variances. Here we generate the array of standard deviations statistically.

Here is an example. Generate the N x N array of standard deviations $\sigma_{i,k}$ statistically and look at the range of exact correlation coefficients (from eqs (K), (N)) . In our example, we generate the standard deviations from a square distribution, with values equally distributed between **smin** and **smax**. The number of variables N = 50 in all cases.

The python program for this procedure is given in appendix 1. Here are results for the minimum and maximum values in the correlation matrix (with 1.0 not included) for given smin and smax. The function for generating RVs is also in appendix 1.

smin	smax,	min $\rho_{i,j}$	max $\rho_{i,j}$
0.010	0.050	0.945	0.981
0.030	0.070	0.876	0.947
0.050	0.090	0.790	0.904
0.070	0.110	0.705	0.853
0.090	0.130	0.619	0.803
0.110	0.150	0.543	0.747
0.130	0.170	0.470	0.705
0.150	0.190	0.414	0.652
0.170	0.210	0.364	0.609
0.190	0.230	0.323	0.567

0.005	0.010	0.997	0.999
0.010	0.015	0.992	0.996
0.015	0.020	0.985	0.993
0.020	0.025	0.975	0.988
0.025	0.030	0.963	0.982
0.030	0.035	0.950	0.976
0.035	0.040	0.935	0.968
0.040	0.045	0.919	0.959
0.045	0.050	0.900	0.950
0.050	0.055	0.882	0.939
0.100	0.300	0.303	0.595
0.300	0.500	0.118	0.344
0.500	0.700	0.064	0.238
0.700	0.900	0.044	0.179
0.900	1.100	0.036	0.143
1.100	1.300	0.030	0.119
1.300	1.500	0.027	0.103
1.500	1.700	0.025	0.090
1.700	1.900	0.024	0.080
1.900	2.100	0.023	0.072

We see that the smaller the variances, the larger the correlation coefficients.

Here is the correlation coefficient array for smin = 0 .05, smax= 0.6, for 5 RVs. The range of coefficients is different from above because the number of variables is much smaller and the smaller number of statistics of variances.

```
[[ 0.76663738  0.76663738  0.9071237   0.90929623  0.79884166]
 [ 0.76663738  0.76663738  0.7767657   0.82275173  0.69996997]
 [ 0.9071237   0.7767657   0.76663738  0.83575352  0.77703674]
 [ 0.90929623  0.82275173  0.83575352  0.76663738  0.7282731 ]
 [ 0.79884166  0.69996997  0.77703674  0.7282731   0.76663738]]
```

generate RVs from given correlation coefficients

A python program for this procedure is given in appendix 2.

The procedure is as follows:

1. Create the correlation matrix (not described in this document).
2. Make sure that condition (O) is satisfied.
3. Compute $\sigma_{i,k}$ from eq (N), for $i,k > 0, i \neq k$.
4. Compute $\sigma_{0,k}$ from eq (K). The other variances are known from the previous step, but it may be that the sum over other variances is too large, so the variances might have to be scaled down.
5. Generate the RVs from eq (G).

6. Also generate the final correlation matrix, which can be different from the input one because of rescaling, as discussed above.

A sample output is shown in appendix 2.

appendix 1 - python pgm for RVs from variances

```
import numpy as np
import math

# generate numPoints values  randomly distributed between
# xmin and xmax, with given mean and standard deviation sd.
# Return numpy array.
def genX(numPoints, xmin, xmax, mean, sd):
    x = xmin + (xmax - xmin)*np.random.ranf(numPoints)
    x = standardize(x)  # has variance = 1
    x *= sd    # has variance = sd**2
    return x

# standardize x to have mean zero, standard deviation 1
def standardize(x):
    mn = np.mean(x)
    var = np.var(x)
    x=x-mn
    x /= np.sqrt(var)
    return x


# Generate sig array, dimension dim, as values randomly distributed between
# smin and smax. sig[i,i] = 0.
def genSig(dim, smin,smax):
    sig=np.zeros( (dim,dim),float)
    vals= smin + (smax - smin)*np.random.ranf(dim**2)
    k=0
    for i in np.arange(0,dim):
        for j in np.arange(i,dim):
            if(i==j): continue;
            sig[i,j]=vals[k]
            k+=1
            sig[j,i]=sig[i,j]
    return sig

# Compute correlation coefficients from sig array.
def corrsFromSigs(sig):
    shp = sig.shape
    p = shp[0]
    rhoExact = np.ones((p,p),float)
    for i in np.arange(1,p):
        sum=1.
        for j in np.arange(0,p):
            if(i==j): continue
```

```
            sum += sig[i,j]**2
        rhoExact[0,i] = 1./math.sqrt(sum)
        rhoExact[i,0]=rhoExact[0,i]
    for i in np.arange(1,p):
        for j in np.arange(1,p):
            if(i==j): continue
            rhoExact[i,j] = rhoExact[0,i]*rhoExact[0,j]*(1.+sig[i,j]**2)
            rhoExact[j,i] = rhoExact[i,j]
    return rhoExact

# Return the min and max values in the correlation matrix,
# neglecting diagonal elements.
def minMaxCorrelation(rho):
    sz = rho.shape[0]
    tem=rho
    for k in np.arange(sz):
        tem[k,k] = rho[0,1]
    mn = np.amin(tem)
    mx = np.amax(tem)
    return (mn,mx)

def corrsFromRVs(x):
    # Calculate the correlation matrix from random variables
    # given in numpy array x.
    # x must have shape (numVars, numSamples)
    xshape = x.shape
    p = xshape[0]
    numx = xshape[1]
    rho = np.ones((p,p),float)
    for i in np.arange(0,p):
        for j in np.arange(i,p):
            if(i==j): continue
            cr = np.corrcoef(x[i,:],x[j,:])
            rho[i,j]=cr[0,1]
            rho[j,i] = rho[i,j]
    return rho

def genRVsFromSig(sig, numSamples):
    #Generate random variables x from the sig matrix.
    # x has shape (sig.shape[0], numSamples)
    # Return the tuple (x, rhoFromX, rhoExact)

    p = sig.shape[0]
    numx = numSamples
    x = np.zeros( (p,numx),float)  # The p RVs to be generated
    x[0,:] = genX( numx,-1.,1.,0.,1.)
    sig = genSig(p,smin,smax)  # std array sigma[i,j]
    eps = np.zeros( (p,p,numx),float)
    for i in np.arange(0,p):
        for j in np.arange(i,p):
            if(i==j): continue
            e = genX(numx,-1.,1.,0.,sig[i,j])
            eps[i,j,:]=e
```

```
        eps[j,i,:]=e
    for i in np.arange(1,p):
        sum = x[0,:]
        for j in np.arange(0,p):
            if(i==j): continue
            sum = sum + eps[j,i,:]
        x[i,:] = standardize(sum)

    rho = corrsFromRVs(x)
    rhoExact = corrsFromSigs(sig)
    return (x, rho,rhoExact)
```

Below is a main:

```
# The following generates standard deviations,
# and the tables in the text and also RVs

numVars = 50  # number of RVs
smin=.1
smax=.3
dels=.2
p=numVars
for i in np.arange(0,10):
    sig = genSig(numVars, smin,smax)
    rho = corrsFromSigs(sig)
    mnn,mxx=minMaxCorrelation(rho)
    print("%6.3f   %6.3f   %6.3f   %6.3f"  % (smin,smax,mnn,mxx))
    smin += dels
    smax += dels

#To generate RVs from a given sig
numSamples = 5000
x, rhoFromX, rhoExact=genRVsFromSig(sig,numSamples)
```

appendix 2-python pgm for RVs from correlations

def sigsFromInputCorrs(inputCorrs):

```
    # This routine takes an array of values
    # inputCorrs=[ 1., rho[0,1], rho[0,2], ....., rho[0,p-1] ]
    # These values can be positive or negative. However,
    # negative values are converted to positive values here.
    # The other correlation elements are computed as
    # rho[i,j] = abs(rho[0,i]*rho[0,j])*(1. + resetRho)
    # but the magnitude is truncated if necessary to be
    # no larger than 0.99.
    # It computes a sig array which tries to be consistent
    # with these initial rho values. A final rho is computed from
    # the sig array. The
    # final rho will in general have different numbers than the initial one
    # because the sig matrix elements must be positive.
```

```
# -------
# returns tuple (initialRho, finalRho, sigArray)

p = len(inputCorrs)
rho = np.ones( (p,p),float)  # initial rho
resetRho = .2

# create full input correlation matrix
# The initial correlations are defined as:
# rho[i,j] = rho[0,i]*rho[0,j]*(1+resetRho) as
# an initial version of rho. If resetRho is very
# small, the output rho should be the same.
for i in np.arange(1,p):
    rho[0,i] = abs(inputCorrs[i])
    if(rho[0,i] > 0.99):rho[0,i]=0.99
    rho[i,0] = rho[0,i]

for i in np.arange(1,p):
    for j in np.arange(i,p):
        if(i==j):continue
        rho[i,j] = rho[0,i]*rho[0,j]*(1. + resetRho)
        if(rho[i,j] > .99): rho[i,j]= .99
        rho[j,i]=rho[i,j]

# compute sigs from input rho. rho may be inconsistent, so
# it my have to be modified in order to generate
# final sigs. This can be used in case ther above
# algorithm for initial rho is modified

sig = np.zeros( (p,p),float )

# first sig[i,j] for i,j>=1, i != j
for i in np.arange(1,p):
    for j in np.arange(i,p):
        if(i==j): continue

        # make sure rho[i,j] is large enough.
        if(rho[i,j] < rho[0,i]*rho[0,j]):
            print("resetting rho: " + str(i) + "," + str(j))
            rho[i,j]=rho[0,i]*rho[0,j] + .02
            if(rho[i,j]) >= 0.99: rho[i,j]=.99
            rho[j,i]=rho[i,j]

        # the ratio in the sqrt should be non-negative
        sig[i,j] = np.sqrt( (rho[i,j]/(rho[0,i]*rho[0,j]))-1.)
        sig[j,i] = sig[i,j]

# Compute sig[0,i] = sig[i,0],i=1,2,...
# The below might modify sig[i,j] computed above. That could modify
# rho[i,j] from values at this point.
for i in np.arange(1,p):
    sum=1.
    for j in np.arange(1,p):
        if(j==i): continue
```

```
            sum += sig[i,j]**2
            newSum=sum
        if(sum  > 1./(rho[0,i]**2)):
            # decrease sig[i,1], sig[i,2],....
            print("modifying sigs to get rho[0,i],i=  " + str(i))
            x=np.sqrt( ( (1/(rho[0,i]**2)) -1. - .01)/sum )
            newSum=1.
            for j in np.arange(1,p):
                if(j==i): continue
                sig[i,j] *= x
                sig[j,i]=sig[i,j]
                newSum += sig[i,j]**2
        sig[0,i]=np.sqrt( (1./(rho[0,i]**2))-newSum )
        sig[i,0]=sig[0,i]

    #print("\n\n    final sig-----")
    #print(sig)

    #recalculate rho from final sig
    finalRho = corrsFromSigs(sig)
    return (rho, finalRho,sig)
```

Here is a main:

```
# First generate sig from input correlations

inputCorrs = [1.0, .9, .85, .8, .95]
(initialRho,finalRho,sig)=sigsFromInputCorrs(inputCorrs)
print("\n--------  initial rho")
print(initialRho)
print("\n\n ------- final rho, computed from sig array")
print(finalRho)

# generate RVs from the sig
numSamples = 5000
x, rhoFromX, rhoExact=genRVsFromSig(sig,numSamples)
```

==========

The result is:

```
modifying sigs to get rho[0,i],i=  1
modifying sigs to get rho[0,i],i=  2
modifying sigs to get rho[0,i],i=  4

--------  initial rho
[[ 1.     0.9    0.85   0.8    0.95 ]
 [ 0.9    1.     0.918  0.864  0.99 ]
 [ 0.85   0.918  1.     0.816  0.969]
 [ 0.8    0.864  0.816  1.     0.912]
 [ 0.95   0.99   0.969  0.912  1.   ]]
```

------- final rho, computed from sig array
[[1.   0.91582321 0.86524275 0.8519123   0.95   ]
 [0.91582321 1.   0.79839043 0.80269399 0.87155447]
 [0.86524275 0.79839043 1.   0.77570766 0.82528964]
 [0.8519123   0.80269399 0.77570766 1.   0.82176097]
 [0.95   0.87155447 0.82528964 0.82176097 1.   ]]