Simulation and Imputation

Rob Leonard (robleonard@tamu.edu (mailto:robleonard@tamu.edu))

Load some packages

Random Vector Generation

In this problem, we are going to be looking at the generating random vectors. Here are some simulation parameters we are going to use:

```
set.seed(1)
n           = 100
p           = 3
rho         = 0.3
Sigma       = matrix(rho,nrow=p,ncol=p)
diag(Sigma) = 1
mu          = rep(5,p)

#Draw feature matrix from standard normal dist.
x = matrix(rnorm(n*p),nrow=n)
```

Simulation

We want to simulate general random variables from standard ones, as in MMA 4.2, 2. (really it is the opposite. The book is discussing standardizing a general random vector. We want to generate a standard random vector and make it more general).

Form the square root of a matrix via the Cholesky decomposition in order to generate multivariate normal draws with a given mean and covariance.

```
#Get square root here
SigmaSqRt_chol   = chol(Sigma)

# Transform x so that it's rows, X_i, are drawn from N(mu,Sigma)
x_chol = x %*% SigmaSqRt_chol
x_chol = x_chol + mu

#Print the sample covariance matrix
cov(x_chol)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.8067621 0.2412126 0.2576907
## [2,] 0.2412126 0.9070733 0.2262908
## [3,] 0.2576907 0.2262908 1.0280075
```

Compare Sigma to the sample covariance matrix.

The sample covariance matrix values aren't terrible, but they aren't close (say within 10%). For example, we want values of 1.0 in the diagonals but one value is only 0.81. I would expect the values to be closer as the sample size n increases to a much larger value.

SVD

Now form the square root via the svd.

```
SigmaSVD  = svd(Sigma)

SigmaSqRt_svd = SigmaSVD$u %*% diag(sqrt(SigmaSVD$d)) %*% t(SigmaSVD$u)

# Transform x so that it's rows, X_i, are drawn from N(mu,Sigma)
x_svd = x %*% SigmaSqRt_svd
x_svd = x_svd + mu

#Print the sample covariance matrix
cov(x_svd)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.8169132 0.2569098 0.2897785
## [2,] 0.2569098 0.9051101 0.2488008
## [3,] 0.2897785 0.2488008 1.0522635
```

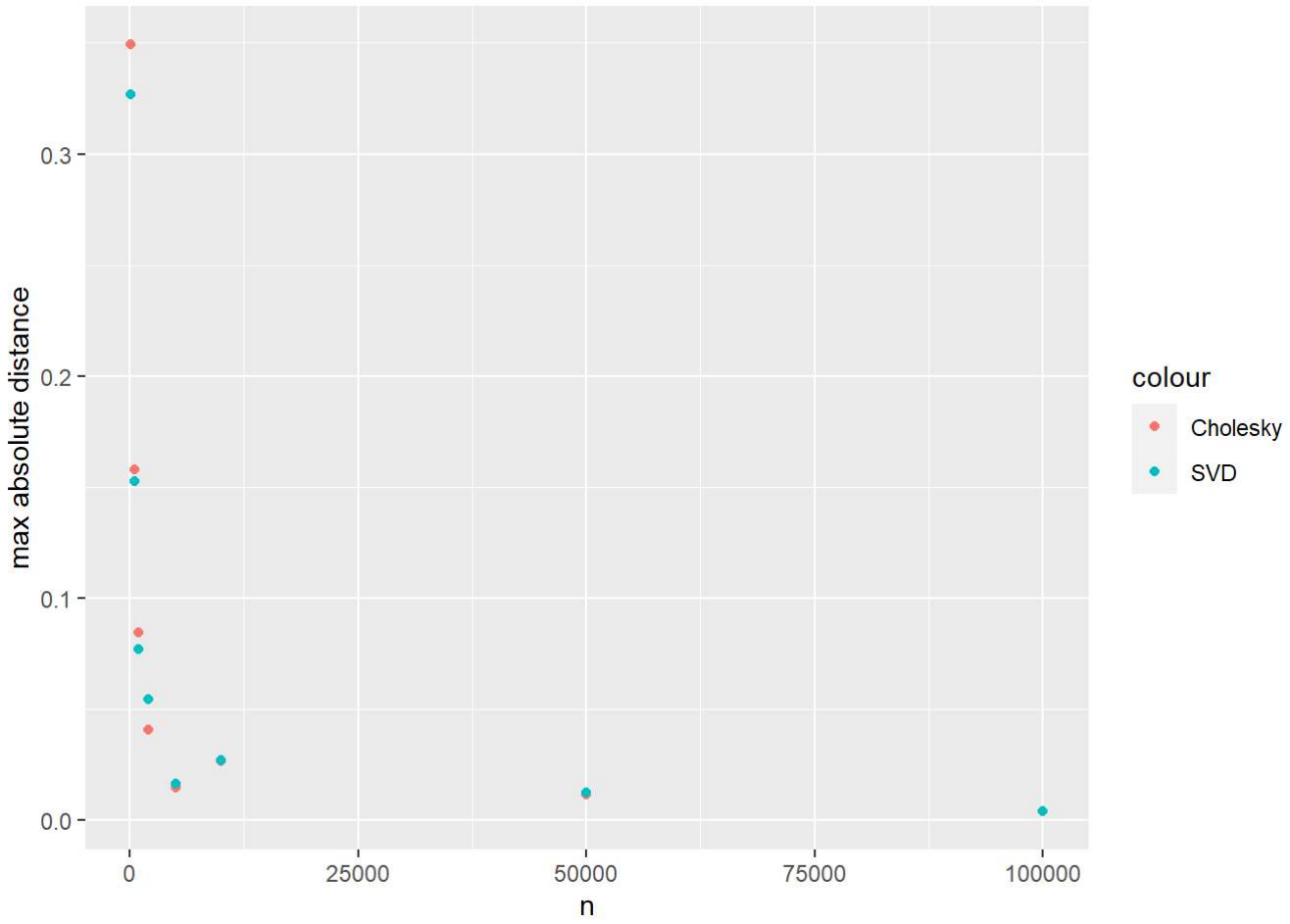
Compare Sigma to this sample covariance matrix.

The sample covariance from svd is similar to Cholesky, and isn't terribly close (within 10%). Again, the first element of the matrix is only 0.82 when it should be 1.0.

Comparison via the law of large numbers

Let's demonstrate via simulation that the two methods come up with the same square root. Note that we will ignore the mean `mu' for this comparison as it doesn't impact the covariance.

```
nGrid = c(100,500,1000,2000,5000,10000,50000,100000)

results = data.frame('nGrid' = nGrid, 'svd' = rep(0, length(nGrid), 'chol' = rep(0, length(nGri
d))))

nIter = 0
for(nSweep in nGrid){
  nIter    = nIter + 1
  xSweep   = matrix(rnorm(nSweep*p),nrow=nSweep)

  cov_chol = cov(xSweep %*% SigmaSqRt_chol)
  cov_svd  = cov(xSweep %*% SigmaSqRt_svd)

  results$chol[nIter] = max( abs(cov_chol - Sigma) )
  results$svd[nIter]  = max( abs(cov_svd - Sigma) )
}

ggplot(results) +
  geom_point(aes(nGrid, chol, color='Cholesky')) +
  geom_point(aes(nGrid, svd, color='SVD')) +
  labs(x = 'n', y = 'max absolute distance')
```



Missing Data & Imputation

For this problem, extend the function explored in class to accomodate qualitative features.
This should look like the following. Let's adjust meanImputeF to just call a single column of the feature matrix:

```
meanImputeF = function(x_j){
  M      = is.na(x_j)
  x_j[M] = mean(x_j,na.rm=TRUE)
  return(x_j)
}
```

Add a new function called modeImputeF that mimics meanImputeF:

```
modeImputeF = function(x_j){
  M      = is.na(x_j)
  tbl = table(x_j)
  x_j[M] = names(tbl)[which.max(tbl)]
  return(x_j)
}
```

Now, we would want a third function called meanModeImputeF:

```
meanModeImputeF = function(X){
  Ximpute = X
  M       = is.na(X)
  for(j in 1:ncol(X)){
    x_j = Ximpute[,j]
    if(is.character(x_j) | is.factor(x_j)){
      Ximpute[,j] = modeImputeF(X[,j])
    }
    else if(is.numeric(x_j)){
      Ximpute[,j] = meanImputeF(X[,j])
    }else{
      break
      warning('Not compatible data type detected')
    }
  }
  return(list('M' = M, 'Ximpute' = Ximpute))
}
```

```
iterativeF = function(X, maxIters = 10, threshold = 1, verbose = TRUE){
  meanModeImpute  = meanModeImputeF(X)
  Ximpute         = meanModeImpute[['Ximpute']]
  M               = meanModeImpute[['M']]
  numericFeatures = sapply(Ximpute,is.numeric)

  converged = FALSE
  nIters    = 0
  while(!converged){
    nIters = nIters + 1
    Ximpute_old = Ximpute
    for(j in 1:ncol(X)){
      x_j = Ximpute[,j]
      if(is.character(x_j) | is.factor(x_j)){
        fitX_j           = glm(factor(x_j) ~ ., data = Ximpute[,-j], family = 'binomial')
        fitX_j_probs     = fitX_j$fitted.values[M[,j]]
        Ximpute[M[,j],j] = ifelse(fitX_j_probs > 1/2, 'fraud','notFraud')
      }
      if(is.numeric(x_j)){
        fitX_j = lm(Ximpute[,j] ~ ., data = Ximpute[,-j])
        Ximpute[M[,j],j] = fitX_j$fitted.values[M[,j]]
      }
    }
    status = max(abs(Ximpute_old[,numericFeatures] - Ximpute[,numericFeatures])/abs(Ximpute_old
[,numericFeatures]))
    if(verbose){ print(status) }

    if( status < threshold){
      converged = TRUE
    }
    if( nIters >= maxIters){
      break
      warning('algorithm failed to converge')
    }
  }
  return(Ximpute)
}
```

Simulation

```
set.seed(1)
p               = 2
n               = 100
covarianceMat   = matrix(c(1,.75,.75,1),nrow=2)
X_cQuantitative = matrix(rnorm(n*p),nrow=n) %*% chol(covarianceMat)
X_cQualitative  = ifelse(rbinom(n,1,.2) == 1,'fraud','notFraud')

X_c             = data.frame('x1' = X_cQuantitative[,1],
                             'x2' = X_cQuantitative[,2],
                             'x3' = X_cQualitative)


X = X_c
X[X$x3 == 'fraud',2]     = NA
X[rbinom(n,1,.2) == 1,3] = NA
```

Impute this data set using the iterative scheme

```
Xhat = iterativeF(X)  # returns Ximpute value
```

```
## [1] 13.94988
## [1] 0.9543573
```

We can check the unknown truth compared with our imputations via ‘table’, which cross-tabulates two qualitative vectors with each other. In this case, the diagonal elements record when we correctly imputed the observation and the off diagonals are where we incorrectly imputed.

```
# only want to look at the results of the imputed values from discussion board post
table(Xhat$x3[is.na(X$x3)],X_c$x3[is.na(X$x3)])
```

```
##
##            fraud notFraud
##   fraud        5        9
##   notFraud     0        7
```

```
correctImputation   = 12 # The number of correctly imputed observations - from main diagonal
```

I correctly imputed 12 observations