The first part of these section notes is a variation on Max Auffhammer's final exam for ARE212 in 2011. If you don't like this treatment of IV estimation, then blame him. Actually, I like this pattern: if you don't like something I do, then blame Max. He is probably at fault. I am faultless. Consider the following data generating process:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \eta_i \quad \text{with} \quad \eta_i \sim N(0,1) \tag{1}$$

And assume that the covariance matrix of the covariates, an additional instrument, and the idiosyncratic error (x_{1i}, x_{2i}, x_{3i}, z_i, and η_i) is defined to be

$$\Sigma = \begin{bmatrix} 1 & 0 & \rho_{13} & 0 & 0 \\ 0 & 1 & \rho_{23} & \rho_{2z} & 0 \\ \rho_{13} & \rho_{23} & 1 & 0 & 0 \\ 0 & \rho_{2z} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where each variable is assumed to have zero mean for simplicity. Note that Σ is consistent with η_i being independently and identically distributed with constant variance. We will explore the properties of IV (e.g., weak instruments and the exclusion restriction) via Monte Carlo simulation.

The first step, then, is to figure out how to generate random data with the appropriate covariance. For this, a useful function is `rmvn.chol`, which returns a random $n \times k$ multivariate normal matrix $\mathbf{X}$, based on the supplied covariance matrix `vcov.mat`. In general, $\mathbf{X} = Q + \mu'$, where $\mathbf{Q'Q} = \Sigma$ and μ is a vector of means for each of the k columns of $\mathbf{X}$. We have assumed away $\mu = \mathbf{0}$ and so the problem becomes a simple application of matrix decomposition.

```
rmvn.chol <- function(n, vcov.mat) {
  k <- ncol(vcov.mat)
  Q <- chol(vcov.mat)
  Z <- matrix(rnorm(k*n), nrow=n, ncol=k)
  return(Z %*% Q)
}
```

It will be handy to have a simple function to generate Σ with three arguments representing the three non-zero correlations across the covariates and the instrument.

```
vcov.fn <- function(rho.13, rho.23, rho.2z) {
  mat <- diag(5)
  mat[3,1] <- rho.13; mat[1,3] <- rho.13
  mat[2,3] <- rho.23; mat[3,2] <- rho.23
  mat[2,4] <- rho.2z; mat[4,2] <- rho.2z
  return(mat)
}
```

The result is a way to generate the random data according to the specified data generating process. The following generates the covariance matrix and a random multivariate normal matrix with 500 observations, printing Σ for reference:

```
(vcov <- vcov.fn(0, 0.5, 0.5))
X <- rmvn.chol(500, vcov)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1  0.0  0.0  0.0    0
[2,]    0  1.0  0.5  0.5    0
[3,]    0  0.5  1.0  0.0    0
[4,]    0  0.5  0.0  1.0    0
[5,]    0  0.0  0.0  0.0    1
```

A quick check to ensure that the variance of each variable is 1, as specified by Σ. There will be some variation around the true variance, but even with 500 observations, it's clear that we are approaching the true variance:

```
apply(X, 2, function(i){var(i)})
```

```
[1] 0.9901410 1.0367531 1.0763512 0.9626240 0.9186701
```

Calculate bias in β and it's standard error

We will now write a couple of functions to help examine the bias of the parameter vector and it's standard error, using an array of estimation techniques, including OLS, 2SLS, and estimation by proxy variable. First, let's put together a very simple function to calculate the parameter vector and it's standard error for both direct regression and two-stage least squares, when a first-stage covariate matrix is provided.

```
ols.results <- function(y, X, first=FALSE) {
  Xt <- t(X)
  xtxi <- solve(Xt %*% X)
  beta <- xtxi %*% Xt %*% y
  if (class(first)=="logical") {
    e <- y - X %*% beta
  }
  else {
    e <- y - first %*% beta
  }
  s2 <- (t(e) %*% e)/(nrow(X) - ncol(X))
  se <- sqrt(diag(xtxi)*s2)
  return(cbind(beta, se))
}
```

Now comes the serious stuff.

```
est.bias <- function(vcov, n=500, B=10000, two.stage=FALSE, proxy=FALSE) {
  true.beta <- c(1, 2, -4, 1)
  res.beta <- mat.or.vec(3,B); res.se <- mat.or.vec(3,B)

  for (i in 1:B) {

    data <- rmvn.chol(n, vcov)

    X <- cbind(1, data[,1:3]); eta <- data[,5]
    y <- X %*% true.beta + eta
    full.ols <- ols.results(y, X)

    if (two.stage==TRUE) {
      endog <- data[,2]
      first  <- cbind(1, data[,c(1,4)])
      predict <- first %*% solve(t(first) %*% first) %*% t(first) %*% endog
      exog <- cbind(1, data[,1], predict)
      limited.ols <- ols.results(y, exog, first=first)
    }
    else if (proxy==TRUE) {
      exog <- cbind(1, data[,c(1, 4)])
      limited.ols <- ols.results(y, exog)
```

```
    }
    else {
      exog <- cbind(1, data[,1:2])
      limited.ols <- ols.results(y, exog)
    }

    res.beta[,i] <- limited.ols[,1] - true.beta[1:3]
    res.se[,i]   <- limited.ols[,2] - full.ols[1:3,2]
  }
  results <- cbind(rowMeans(res.beta), rowMeans(res.se))
  colnames(results) <- c("beta bias", "se bias")
  print(results)
}
```