EXERCISE ON VAR AND VARMA MODELS

On the course website there is a link to a directory that contains this exercise and some data files and programs you can use to do the exercise.

Unfortunately, I somehow put up on the course web site slightly out of date versions of some programs that contained bugs. The huge gap people were finding between the VARMA and VAR likelihoods arose from one of these errors. Corrected versions are now on the web site

(1) Estimate a four-lag VAR in quarterly data on the unemployment rate and the log of the personal consumption expenditure price deflator. Make the starting date for the model data 1960.0 (first quarter of 1960). `trimts(window(pcdata, start=1960))` will do this for you. (The `trimts()` function throws out initial and final observations that contain any NA values for either series; it may not be necessary for this data set.) Multiply the log PCEP data by 100 so that its changes are quarterly inflation rates in percent. (Note that in the US, the convention is to report quarterly inflation rates at annual rates, so quarterly changes in logs, scaled by 100, will be four times smaller than what you are used to seeing in the newspaper.) Use a flat prior and condition on initial observations. An R data file, with a time series object containing these data, already logged and scaled, is on the web site as `pcdata.RData`. If you use the `rfvar3()` function, you will need to set `lambda=NULL` and `mu=NULL` in its argument list to accomplish this. Otherwise, the default is to use the "cointegration prior" and "unit root prior" dummy observations of the Minnesota prior.

This was straightforward:

```
voutex <- rfvar3(trimts(window(pcdata, start=1960)), lags=4,
            lambda=NULL, mu=NULL)
```

(2) Generate 1000 draws from the posterior distribution of the VAR parameters. This can be done exactly, without MCMC accept-reject steps, because the posterior with a flat prior and conditioned on initial observations has a standard form — inverse-Wishart as the marginal distribution of the covariance matrix of residuals, with a normal distribution for the parameters conditional on that covariance matrix. The function `postdraw()` will generate draws from the posterior, given the maximum likelihood estimates that come out of `rfvar3()`

```
vexdraw <- postdraw(voutex, n=1000, nosigprior=TRUE)
```

(3) Use your draws from the posterior to generate 1000 draws of 40-quarter impulse responses to triangularly orthogonalized shocks. The `impulsdtrf()` function generates such responses for a single set of parameter values. For this AR model, you can leave the `smat` argument at the `NULL` default. You will need to invoke it 1000 times to generate a $2 \times 2 \times 40 \times 1000$ array containing the draws of the impulse responses.

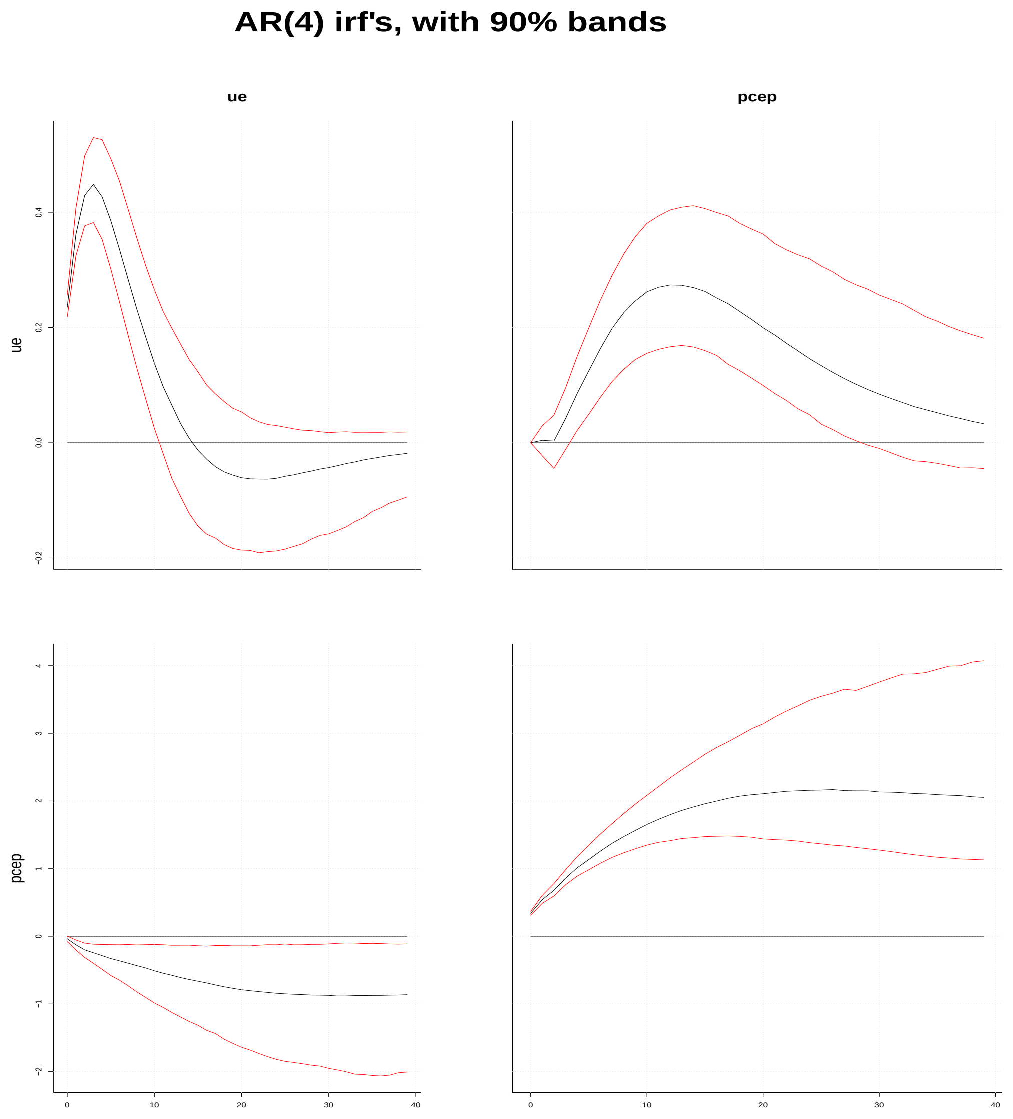
Date: November 16, 2015.

```
respexdraw <- array(0, c(2,2, 40, 1000))
for (id in 1:1000) respexdraw[ , , , id] <-
   with(vexdraw, impulsdtrf(list(By=By[ , , , id], smat = t(smat[ , , id]))))
```

(4) Sort the impulse response draws (use `apply(resp, c(1,2,3), sort)`) and then plot the 50th, 500th, and 950th elements of the sorted responses to generate 90% bands for the impulse responses around the posterior median. The function `plotir()` will do this for you.

```
srespexdraw <- apply(respexdraw, c(1,2), sort)
srespexdraw <- aperm(respexdraw, c(2,3,4,1))  # needed because apply() puts
                                               # its results in first dimension
plotir(srespexdraw[ , , ,c(50,500,950)], 1:2, 1:2,
         main="AR(4)␣irf’s,␣with␣90%␣bands", file="respexBand.pdf")
## I found that I needed to give a dev.flush() command in order to get the
## onscreen display to finish drawing, but that is likely peculiar to my
## computer and operating system.
```

Note that my claim that you could leave `smat` at the default NULL value was not correct. If `smat==NULL`, the impulsdtrf function tries to form a covariance matrix from residuals it expects to find as part of the initial list argument.



(5) Use the program `estVARMA()` to find the MLE estimates for a vector ARMA(2,2) model. The `estVARMA()` program returns, as list element `xh`, a single vector containing all the model parameters. Use

```
armlst <- param2b(armout$xh, 2,2,2,2)
```

to convert this to coefficient matrix arrays and the estimated covariance matrix.

```
armout22 <- estVARMA(trimts(window(pcdata, start=1960.5)),
   By=array(c(1,0,0,1,rep(0,4)), c(2,2,2)), Be= array(0, c(2,2,2)),
           Bx=c(0,0), Sig=diag(2), H0=NULL, nit=200)
```

The 1960.5 start value is needed to make the sample period match that for the AR(4) model.

(6) Calculate and plot the impulse responses for this ARMA(2,2) model, with the same orthogonalization you used for the VAR. This is not a single-call calculation. You will need to first use `impulsdtrf()` on the AR coefficients of the model alone, then form linear combinations of that result using the coefficient matrices from the moving average part of the model. Figuring out how to do this is up to you. The first step of the calculation is done by

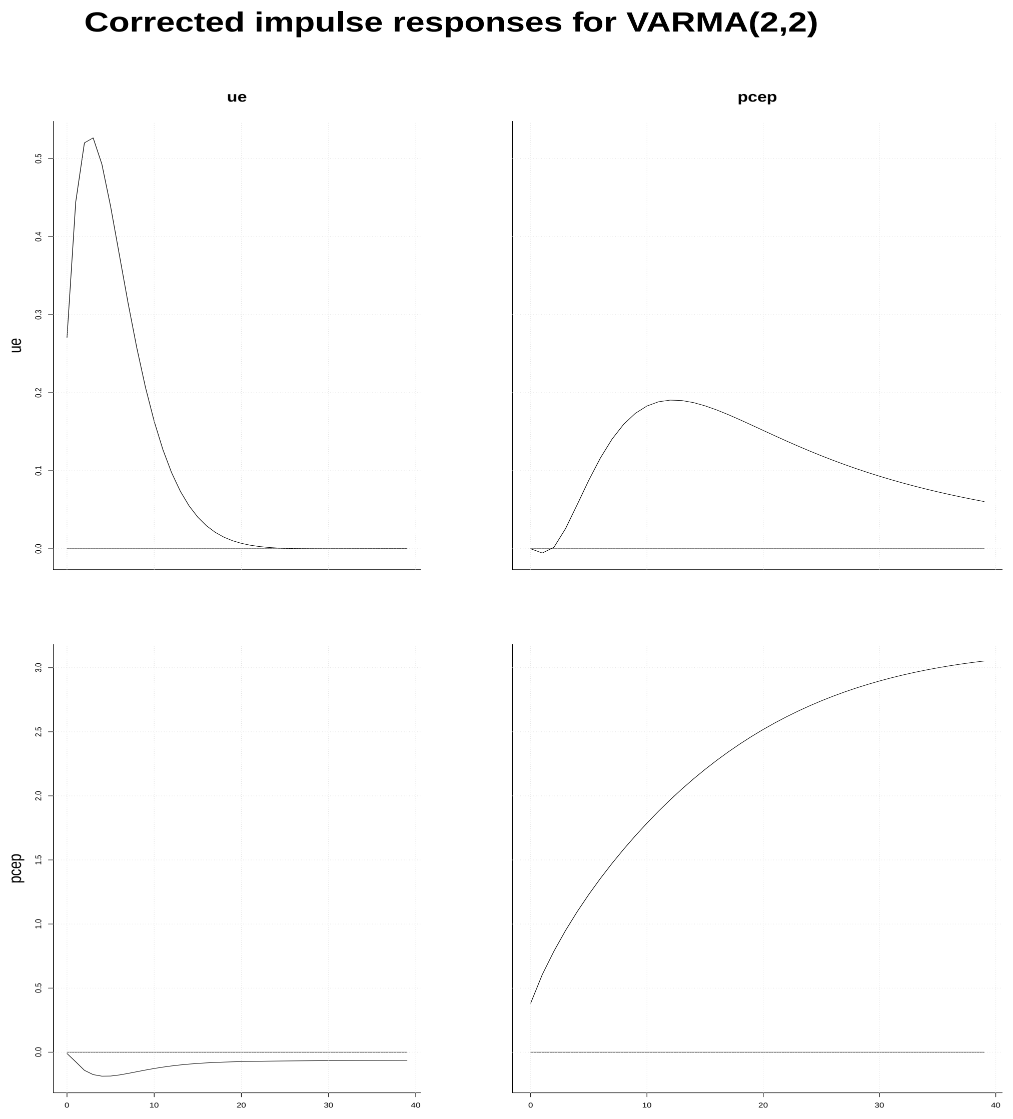
From a student's exercise answer, I realized I'd missed something below. Using smat in the call to `impulsdtrf()` is not correct. It needs to postmultiply the whole reduced form impulse response *after* it's been convoluted with the MA component. Assuming you used `estVARMA()` and created `armlst` as suggested above, the next steps can be:

```
## mistake:
## resparm22 <- impulsdtrf(vout=armlst22, smat = t(chol(armlst22$Sig)))
## correct, responses to unorthogonalized shocks:
resparm22 <- impulsdtrf(vout=armlst22, smat = diag(2))
resparm22[ , , 2:40] <- resparm22[ , , 2:40] +
   aperm(tensor(resparm22[ , , 1:39], arlmst22$Be[ , , 1], 2, 1), c(1,3,2))
resparm22[ , , 3:40] <- resparm22[ , , 3:40] +
   aperm(tensor(resparm22[ , , 1:38], armlst22$Be[ , , 2], 2, 1), c(1,3,2))
## Now convert to responses to orthogonalized shocks:
resparm22 <- tensor(resparm22, t(chol(armlst22$Sig)), 2, 1)
resparm22 <- aperm(resparm22, c(1,3,2)
## the algebra loses the dimension names, so for handy graph labeling:
dimnames(resparm22) <- dimnames(respar) # assuming respar is a properly
                                         # labeled output from impulsdtrf()
```

This uses the `tensor` package, which is often handy, but there are other ways it could have been done.

Since you give an explicit `smat`, the program needs only the `$By` component of a complete output list from `rfvar3()`.

The impulse responses don't look very different from those implied by the AR(4).



(7) Calculate the log likelihoods for the ARMA(2,2) and the VAR(4) models and compare them. For the ARMA model, the log likelihood is directly available as `armout$fh`. For the VAR model, you can compute the log likelihood from the covariance matrix of least squares residuals (available as `arout$u` if you have earlier set `arout` as the return value from `rfvar3()`. Figuring out exactly what the formula is is again up to you. A cautionary hint: the R command **`cov()`** calculates the "unbiased" estimate of the covariance matrix, dividing by $T-1$, whereas the MLE divides by T. This apparently minor difference can matter for this model comparison.

The likelihood for the AR model is

$$-\tfrac{T}{2}\log(2\pi) - \tfrac{T}{2}\log(|\Sigma|) - \tfrac{1}{2}\operatorname{trace}(\Sigma^{-1}S)\,,$$

where $S = u'u$, the matrix of cross-products of the residuals. Since Σ is unconstrained, its MLE is S/T, where T is sample size, so the likelihood at the maximum is

$$-\tfrac{T}{2}\log(2\pi) - \tfrac{T}{2}\log(|S/T|) - \frac{Tn}{2},$$

where n is the length of the $y(t)$ vector.

With the correct code, I found a log likelihood of -55.65 for the VARMA(2,2) and -53.108 for the VAR(4). This implies an odds ratio of $e^{-2.64} = .071$, or about 14 to 1 in favor of the VAR(4). This may seem like a strong result, but as we discussed in class, odds ratios on small collections of models tend to be unreasonably decisive.

(8) For both models, calculate unconditional sample means and covariance matrices and compare them to the initial conditions. For both types of model, this can be done with the function `doubleic()`. For the ARMA model you may find it useful to use `armasysmat()` to put the model's parameters into the form that `doubleic()` needs for the calculation. `doubleic()` calculates the distribution after `2^log2T` periods, conditional on zero initial conditions. The default value for `log2T` is 8, so the distribution is what emerges after 256 quarters. For models with roots well below one, this will be indistinguishable from the unconditional distribution. If the roots are close to one, whether above or below, the calculated distribution will not be a good approximation to the unconditional one (which doesn't even exist if roots are larger than one in absolute value), but arguably what `doubleic()` returns makes sense as a distribution to use for the initial conditions in any case.

```
> AS <- with(armlst22, armasysmat(By,  Be, Bx, Sig))
> icarma <- doubleic(AS$sysmat, AS$sigkf, const=NULL, y0=c(rep(0,8), 1))
    # using doubleic in pure first-order mode.
> icdev <- window(pcdata, start=1960)[3:4, ]  # leaving out first two
                          # observations, because ARMA(2,2) has only two
                                       #  initial conditions, not four
> icdev
       ue     pcep
[1,] 4.9 286.6933
[2,] 5.5 287.1359
> icdev <- c(t(icdev))
> icdev
[1]   4.9000 286.6933   5.5000 287.1359
## geting ic's lined up with the way they are indexed in icarmaj
> icdev <- icdev - icarma$mu[1:4]
> icdev %*% solve(icarma$Omega[1:4, 1:4], icdev)
         [,1]
[1,] 9.647742
```

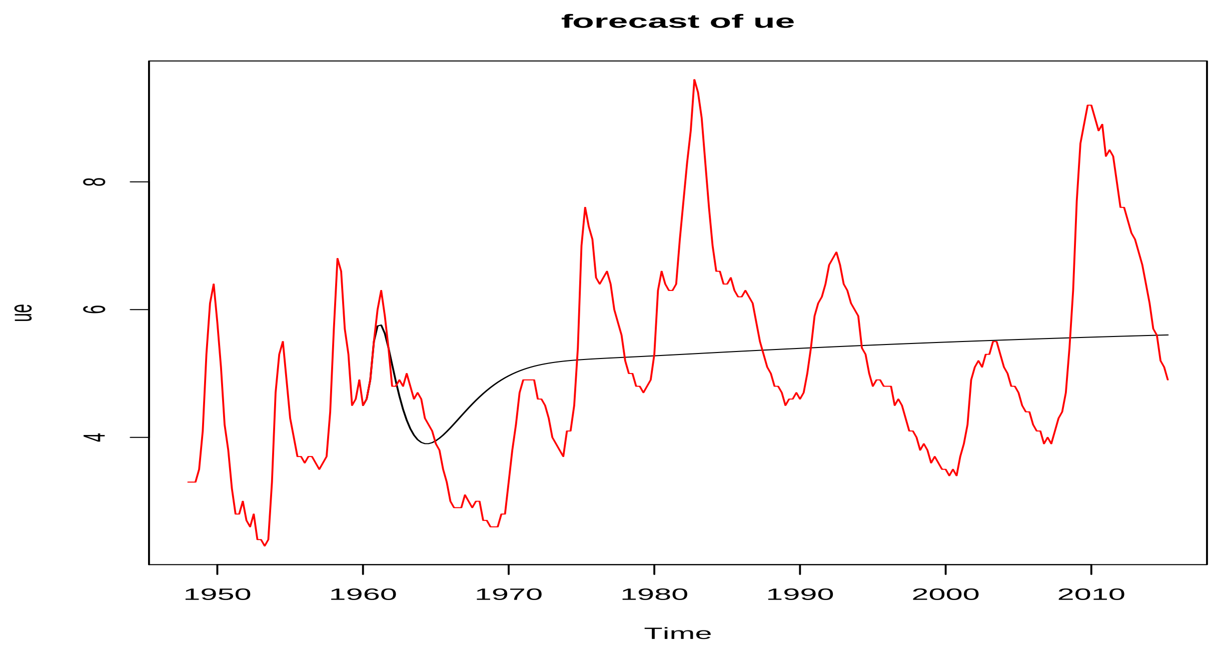
This is scaled like a chi-squared(4), which puts it just inside the 5 per cent tail of that distribution, big but not as bad as one sometimes finds. For the AR model,

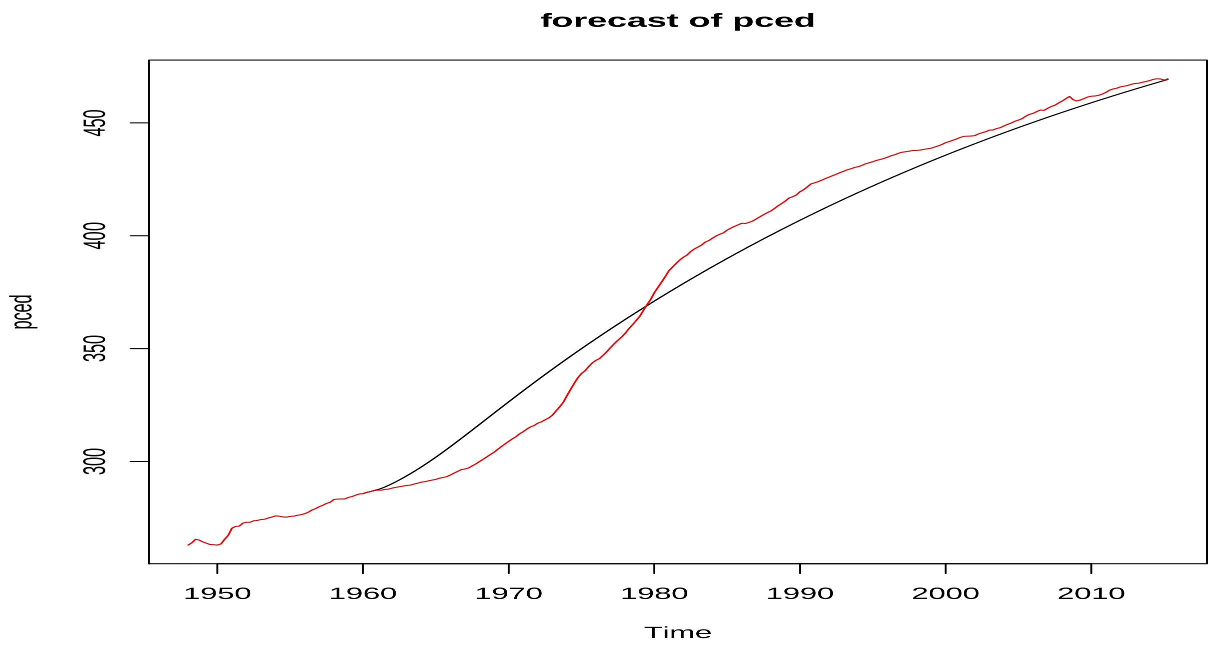
```
> icar <- doubleic(voutex$By, crossprod(voutex$u)/dim(voutex$u)[1],
          const=voutex$Bx, y0=rep(0,8))
> icardev <- window(pcdata, start=1960)[1:4, ]
> icardev <- c(t(icardev))
> icardev <- icardev - icar$mu[-9]
> icardev
[1]   -0.9044194 -124.2387995   -0.8017754 -122.9256835   -0.4991171
[6] -121.7531343    0.1035557 -120.5154367
> icardev %*% solve(icar$Omega[-9,-9], icardev)
         [,1]
[1,] 42.31066
> pchisq(42.311, 8)
[1] 0.9999988
```

from which we see that the AR model has put the initial conditions much farther out in the tail of the steady state distribution.

(9) For the AR(4) model, calculate the forecast over the entire sample based only on initial conditions, using the MLE parameters, and compare this (on the same plot) to the actual time path of each of the two variables. Assess whether the estimates are implying unreasonably accurate long term forecasts. For this part of the problem `fcast()` is likely to be useful.

```
> arfcst <- with(voutex, fcast(y0=window(pcdata, start=1960)[1:4, ],
    By, Bx, horiz=218))
> arfcst <- ts(arfcst, start=1960, freq=4)
> plot(cbind(arfcst[ ,1], pcdata[ , 1]), plot.type="single", col=1:2,
    main="forecast␣of␣ue", ylab="ue")
> dev.copy2pdf(file="uefcst.pdf")
> plot(cbind(arfcst[ ,2], pcdata[ , 2]), plot.type="single", col=1:2,
    main="forecast␣of␣pced", ylab="pced")
> dev.copy2pdf(file="pcedfcst.pdf")
```





(10) Resample from the draws you made from the flat-prior posterior on the AR(4) model, to obtain a sample from the posterior with a flat prior, but using the initial conditions, treating them as having the distribution delivered by `doubleic()`. As we discussed in class, this can be done by sampling from the initial set of draws, with replacement, and using the likelihood for the initial conditions (which is the ratio of the target pdf to the pdf you are drawing from in this case) to implement the independence Metropolis-Hastings accept/reject rule.

I should know better than to include an exercise I haven't tried yet myself, even when it looks like it should be simple. Unfortunately, the steady state distribution for initial conditions is *extremely* informative here, meaning that it is not flat relative to the rest of the likelihood, as I was thinking it would be. The result is that the draw from the normalized likelihood with the highest log pdf for the initial conditions is so much greater than the next highest value that, once that draw was resampled, the accept-reject rule would leave the algorithm stuck there indefinitely. I tried using inflation (differenced log p) in place of the price level. That did result in a less wide range of values for the initial condition density draws, but the ratio of largest to next largest pdf value was still on the order of 10^{12}, so the same issue arose. It appears no short cut is available here. To draw from the posterior using the steady state distribution for the initial conditions seems to require direct MCMC on the full likelihood.