STAT 621
Modeling Exchange Rate-Too many models? Probably.

Ben Hulet, Yuqiu Yang, Zhenshan Jin

Introduction

In this report we will be comparing the results from fitting two GARCH volatility models, two stochastic volatility models, and one EGARCH model on two difference exchange rate pairs. The exchange rate pairs are the price of 1 Great British Pound (GBP) in dollars and the price of one Euro (EUR) in dollars. These series are daily exchange rates from January 2005 to March 2016. We will be comparing the out of sample one step ahead prediction performance of these models on the final three months of the series.

Modeling the FX pairs

Before fitting these models to our pairs, we calculated the log return of each series, and split each series into two parts: one from 2005-01-01 to 2016-01-01 and the other from 2016-01-01 to 2016-03-30. And judging from the corresponding Acf and Pacf plots and the Dickey Fuller Test for unit roots, we concluded that our series are random walk processes. In so doing, we can fit these volatility models directly to the log return series.

GARCH

$$a_t = \sigma_t \epsilon_t$$
$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{s} \alpha_i a_{t-i} + \sum_{j=1}^{m} \beta_j \sigma_{t-j}^2$$

Empirical studies have shown that return series are often uncorrelated but dependent. GARCH models try to model non-constant variances conditional on the past, but use constant unconditional variances. We fit GARCH(1,1) models to our exchange rate pairs with both normal distributed errors and t-distributed errors. In order to evaluate the forecasting capability, we calculated 50%, 80% and 90% predictive intervals and checked their actual coverage rates.
Below the corresponding plots are shown:

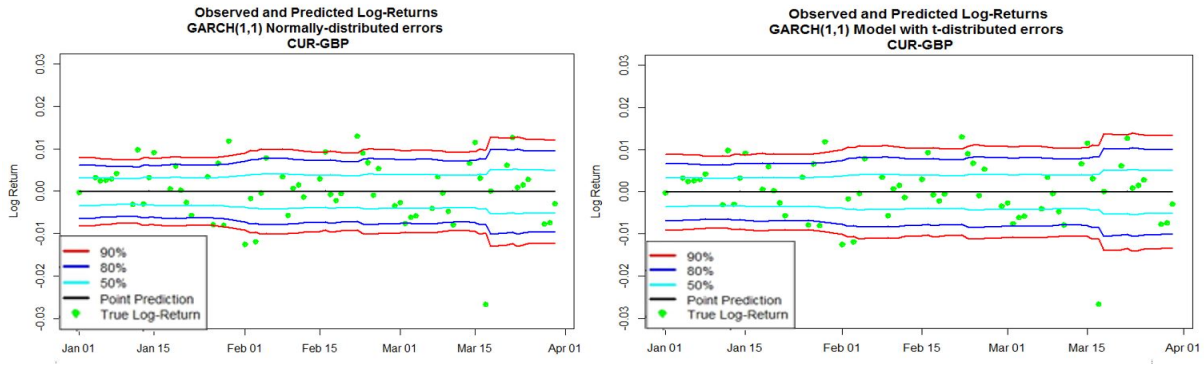

Figure 1: USD/GBP

	90%	80%	50%
Coverage Rate	84.37%	73.43%	50.00%

Table 1: GBP N-Distributed Errors

	90%	80%	50%
Coverage Rate	87.50%	79.70%	51.60%

Table 2: GBP T-distributed Errors

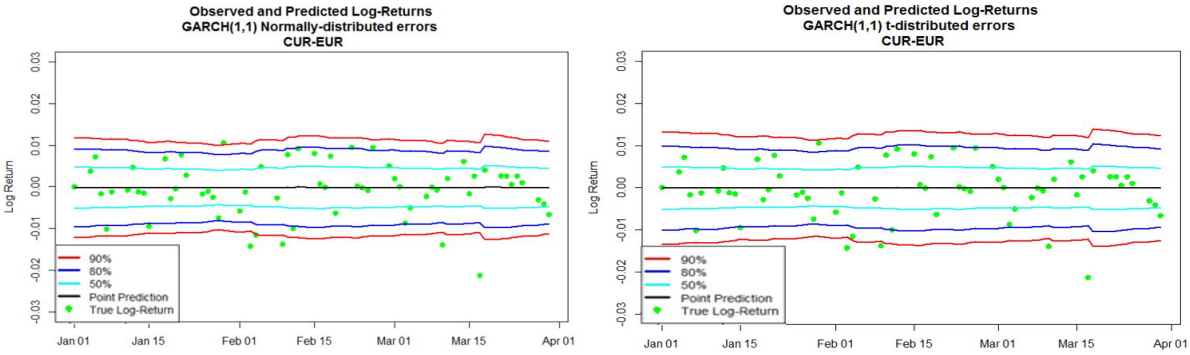

Figure 2: USD/EUR

	90%	80%	50%
Coverage Rate	90.60%	82.80%	56.20%

Table 3: EUR N-Distributed Errors

	90%	80%	50%
Coverage Rate	93.80%	84.40%	56.20%

Table 4: EUR T-distributed Errors

The criterion that we used to determine which model is better is that the closer the coverage rate to the prediction interval, the better the model is. GARCH(1,1) model with t-distributed error is a better choice in the USD-GBP exchange rate. GARCH(1,1) with normally distributed error is a better choice in the USD-EUR exchange rate.

EGARCH

$$a_t = \sigma_t \epsilon_t$$

$$ln(\sigma_t^2) = \alpha_0 + \sum_{i=1}^{s} \alpha_i \frac{|a_{t-i}| + \gamma_i a_{t-i}}{\sigma_{t-i}} + \sum_{j=1}^{m} \beta_j ln(\sigma_{t-j}^2)$$

The EGARCH model was developed by Nelson in 1991 and is often used in financial markets where the volatility corresponding to positive returns is generally smaller than the volatility corresponding to negative returns. EGARCH allows for asymmetries in the relationship between return and volatility which contributes to it's effectiveness in modeling exchange rates. Additionally, EGARCH does not impose the assumption of positive coefficients which allows random oscillatory behavior in the sigma process. For this project we varied the order of the model and the distribution of the errors. For the order of the model we found that it had a negligible effect on the one step ahead predictive performance and for the distribution of errors, normal distribution seems to have better performance. So in the following part we only show the results from EGARCH(1,1) with normally distributed errors.

	90%	80%	50%
Coverage Rate	87.50%	76.60%	53.10%

Table 5: USD/GBP

	90%	80%	50%
Coverage Rate	90.60%	81.20%	56.20%

Table 6: USD/EUR

Comparing the coverage rate from GARCH(1,1) model, we can found that EGARCH(1,1) model with normally distributed errors has the best performance for both exchange rates time series. After fitting the model, we also checked the leverage effect for each time series where the standardized shock magnitude is equal to 2. Shown here:

$$\frac{\sigma_t^2(\epsilon_{t-1} = -2)}{\sigma_t^2(\epsilon_{t-1} = 2)}$$

	USD-GBP	USD-EUR
Leverage Effect	0.836	0.826

Table 7: Leverage Effect

As we can see from the table, there is an obvious up leverage effect. Take USD-GBP exchange rate for example, the impact of a negative shock of size 2 standard deviations is about 16.4% lower than that of a positive shock of the same size.

Stochastic Volatility Model

$$a_t = \sigma_t \epsilon_t$$
$$(1 - \alpha_1 B - \cdots - \alpha_m B^m) ln(\sigma_t^2) = \alpha_0 + \nu_t$$

Stochastic volatility models are those in which the variance of a stochastic process is itself randomly distributed. At time t, the volatility is NOT pre-determined (deterministic) given previous values. This allows these models to be more flexible than GARCH models by allowing the volatility to be a random process.

	90%	80%	50%
Coverage Rate	85.93%	75.00%	45.30%

Table 8: GBP N-Distributed Errors

	90%	80%	50%
Coverage Rate	79.69%	65.63%	40.63%

Table 9: GBP T-distributed Errors

	90%	80%	50%
Coverage Rate	90.63%	76.56%	56.25%

Table 10: EUR N-Distributed Errors

	90%	80%	50%
Coverage Rate	87.50%	75.00%	53.13%

Table 11: EUR T-distributed Errors

We can see that the stochastic volatility models performed moderately well with Normally distributed errors but the performance was noticeably worse with T-Distributed errors. The effect of the error distribution is the opposite of what was observed in the GARCH family. We also see that the stochastic models were similarly ineffective at modeling the GBP exchange rate.

Prior distribution sensitivity checks

For all stochastic volatility models used in this project (those with normally distributed errors and T-distributed errors), we performed prior distribution sensitivity checks. To do this we ran 11 different models where the prior distributions for each of the parameters was systematically varied. Below is the table corresponding to the prior distribution sensitivity checks where the first four columns contain the parameters we used while calling the function svsample.

priormu	priorphi	priorsig	priornu	90%*	80%*	50%*	90%**	80%**	50%**
(0,100)	(5,1.5)	1	NA	0.86	0.75	0.45	0.91	0.77	0.56
(-10,1)	(20,1.5)	0.1	NA	0.86	0.75	0.44	0.91	0.77	0.56
(-10,1)	(5,1.5)	1	NA	0.86	0.75	0.44	0.91	0.77	0.56
(0,100)	(20,1.5)	1	NA	0.88	0.75	0.45	0.91	0.77	0.56
(0,100)	(5,1.5)	0.1	NA	0.84	0.75	0.44	0.91	0.78	0.56
(0,100)	(5,1.5)	1	(2,100)	0.80	0.66	0.41	0.88	0.75	0.53
(-10,1)	(20,1.5)	0.1	(5,20)	0.80	0.70	0.42	0.88	0.77	0.53
(-10,1)	(5,1.5)	1	(2,100)	0.80	0.67	0.42	0.88	0.73	0.53
(0,100)	(20,1.5)	1	(2,100)	0.80	0.70	0.42	0.88	0.75	0.53
(0,100)	(5,1.5)	0.1	(2,100)	0.80	0.70	0.41	0.88	0.75	0.53
(0,100)	(5,1.5)	1	(5,20)	0.80	0.70	0.42	0.88	0.73	0.53

Table 12: Sensitivity Checks Table $* = USD/GBP$ $** = USD/EUR$

Looking at this table, we find that the effect of changing the prior is negligible in terms of the prediction interval and recovery rate.

Conclusion

We came to several conclusions while working on this project. First, that stochastic volatility models are extremely slow to implement and they did not perform better than standard GARCH(1,1) models. Additionally, varying the prior distribution did not effect the model's performance. This may be due to the relatively large sample size used in this project, where the length of the entire series contained 2,929 observations.

In comparing T-Distributed errors versus normally distributed errors, we found that T-Distributed errors provided a wider predictive interval than normally distributed errors for the GARCH family, but the effect is reversed for Stochastic volatility models. At this time, we do not have an understanding for this observation.

Generally the GARCH Family performed well in recovering the mean zero series for 1 step ahead rolling forecasts. Additionally, the models were easy to implement and were computationally time efficient. By implementing EGARCH models, we did improve the recovery rate over the standard GARCH(1,1). Because EGARCH models are similarly computationally inexpensive, we find they are the best choice in our comparison of forecasting coverage rates. We also found that for these series, changing the order of the EGARCH model had no effect on the model performance.

Because all models performed worse when predicting the GBP exchange rate series we can conclude that when the series expresses a higher degree of volatility, we should choose different methods than those explored in this project.

References

[1] Ruey S.Tsay *Analysis of Financial Time Series, Third Edition* Wiley. 2010.

[2] Gregor Kastner *Dealing with Stochastic Volatility in Time Series Using the R Package stochvol* Journal of Statistical Software. 2016

[3] Gregor Kastner *Heavy-Tailed Innovations in the R Package stochvol* 2015

[4] Torben G. Andersen and Tim Bollerslev *Answering the skeptics: Yes, standard volatility models do provide accurate forecasts* International Economic Review. 1998.

[5] Alexios Ghalanos *Introduction to the rugarch package* 2015.

[6] Robert F. Engle *Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation* Econometrica, Vol. 50, No. 4. 1982

[7] Tim Bollerslev *Generalized Autoregressive Conditional Heteroskedasticity* Journal of Econometrics. 1986

[8] Daniel B. Nelson *Conditional Heteroskedasticity in Asset Returns: A New Approach* Economitrica, Vol. 59, No. 2. 1991

[9] Angelo Melino and Stuart Turnbull *Pricing Foreign Currency Option with Stochastic Volatility* Journal of Econometrics. 1990

[10] Spencer Graves *Package 'FinTS'* Methods and tools for analyzing time sires. 2009. https://cran.r-project.org/web/packages/FinTS/FinTS.pdf

[11] Adrian Trapletti et al. *Package 'tseries'* Time series analysis and computational finance. 2015. https://cran.r-project.org/web/packages/tseries/tseries.pdf

[12] Alexios Ghalanos *Package 'rugarch'* Univariate GARCH Models. 2015. https://cran.r-project.org/web/packages/rugarch/rugarch.pdf

[13] Gregor Kastner *Package 'stochvol'* Efficient Bayesian Inference for Stochastic Volatility Models. 2016. https://cran.r-project.org/web/packages/stochvol/stochvol.pdf

[14] Vitalie Spinu *Package 'lubridate'* Make Dealing with Dates a Little Easier. 2016. https://cran.r-project.org/web/packages/lubridate/lubridate.pdf

[15] R-core *Package 'parallel'* Coarse-grained Parallelization. 2015. https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

[16] Rob J Hyndman et al. *Package 'forecast'* Methods and tools for displaying and analysing univariate time series forecasts 2015. https://cran.r-project.org/web/packages/forecast/forecast.pdf

[17] Rich Calaway, et al. *Package 'doParallel'* Provides a parallel backend for the %dopar% function using the parallel package. 2015. https://cran.r-project.org/web/packages/doParallel/doParallel.pdf

[18] Rich Calaway, et al. *Package 'foreach'* Provides Foreach Looping Construct for R. 2015. https://cran.r-project.org/web/packages/foreach/foreach.pdf

Appendix

```
## check the First series
Box.test(DFirst[,2], type="Ljung-Box") ## reject null of independence
kpss.test(DFirst[,2]) ## fail to reject null of independence
Box.test(DFirst[,2]^2,lag=9,type="Ljung-Box") ## significant ARCH effects
adf.test(DFirst[,2]) #Augmented Dickey-Fuller reject null of non stationarity,
## Fitting Models
##############################################################################
## (Non-Bayesian) GARCH(1,1)
sgarch_model <-ugarchspec(variance.model=list(model="sGARCH",garchOrder=c(1,1)),
                          distribution.model="norm",
                          mean.model=list(armaOrder=c(0,0),include.mean=TRUE))
sgarch_fit <- ugarchfit(sgarch_model,DFirst)
mod_sgarch <- ugarchroll(sgarch_model, data = logFX, n.ahead = 1,refit.every = 1,
                         n.start = 2865, refit.window = "recursive",
                         solver = "hybrid", fit.control = list(),
                         calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.025, 0.05),
                         keep.coef = TRUE)
## GARCH(1,1) with t distributed errors
##############################################################################
sgarch_t_model <- ugarchspec(variance.model=list(model="sGARCH",garchOrder=c(1,1)),
                             distribution.model="std",
                             mean.model=list(armaOrder=c(0,0),include.mean=TRUE))
sgarch_t_fit <- ugarchfit(sgarch_t_model,DFirst)
mod_t_sgarch <- ugarchroll(sgarch_t_model, data = logFX, n.ahead = 1,refit.every = 1,
                           n.start = 2865, refit.window = "recursive",
                           solver = "hybrid", fit.control = list(),
                           calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.025, 0.05),
                           keep.coef = TRUE)
## EGARCH(1,1) with normally distributed errors
##############################################################################
egarch_model <- ugarchspec(variance.model=list(model="eGARCH",garchOrder=c(1,1)),
                           distribution.model="norm",
                           mean.model=list(armaOrder=c(0,0),include.mean=TRUE))
egarch_fit <- ugarchfit(egarch_model,DFirst)
mod_egarch <- ugarchroll(egarch_model, data = logFX, n.ahead = 1,refit.every = 1,
                         n.start = 2865, refit.window = "recursive",
                         solver = "hybrid", fit.control = list(),
                         calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.025, 0.05),
                         keep.coef = TRUE)
## Stochastic Volatility Model using Normal and T distribution
##############################################################################
SV <- function(train = DFirst[,2], test = DSecond[,2], primu = c(0,100), priphi = c(5,1.5),
```

```
              prisig = 1 ,prinu = NA, heavytail=FALSE, predict=FALSE){
if(heavytail){
  res <- svsample(train, priormu=primu, priorphi=priphi, priorsigma=prisig, priornu=prinu)
    }else{
  res <- svsample(train, priormu=primu, priorphi=priphi, priorsigma=prisig)
}
if(predict)
{
  coverage <- matrix(nrow=length(train), ncol=3); colnames(coverage) <- c("90%","80%","50%")
  preinterval <- matrix(nrow=length(train), ncol=7)
  for(i in 1:{length(train)})
  {
    if(i==1)
    {
      forecast <- exp(predict(res,1)/2)
    }else{
      if(heavytail){
        res1 <- svsample(c(train, test[1:{i-1}]), priormu = primu, priorphi = priphi,
                         priorsigma = prisig, priornu = prinu)
      }else{
        res1 <- svsample(c(train, test[1:{i-1}]), priormu = primu, priorphi = priphi,
                         priorsigma = prisig)
      }
      forecast <- exp(predict(res1, 1) / 2)
    }
    epsilon <- rnorm(length(forecast), mean = 0, sd = 1)
    at <- forecast * epsilon
    preinterval[i,] <- quantile(at,c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95))
    coverage[i,] <- c({test[i] >= preinterval[i,1] & test[i] <= preinterval[i,7]},
                      {test[i] >= preinterval[i,2] & test[i] <= preinterval[i,6]},
                      {test[i] >= preinterval[i,3] & test[i] <= preinterval[i,5]})
  }
  return(list(CovRate=colMeans(coverage), CovMatr=coverage, Predict=preinterval, First=res,
              Para=list(primu = primu, priphi = priphi, prisig = prisig, prinu = prinu,
                        heavytail= heavytail, predict= predict), Ptitle=title))
}}
```