Power Simulations Comparing Portmanteau Tests

Esam Mahdi
University of Western Ontario

A. Ian McLeod
University of Western Ontario

Abstract

The results reported by Peña and Rodriguez (2002, Tables 3 and 9), Lin and McLeod (2006, Tables 3-6), and Mahdi and McLeod (2011, Figure 1 and Table 1) could be reproduced using the **portes** package with the following simulation functions. The performance of the generalized variance portmanteau test, `gvtest`, and its competitors, `BoxPierce`, `LjungBox`, `Hosking`, and `LiMcLeod`, for univariate-multivariate time series were compared based on two simulation methods. The first method used the Monte-Carlo techniques and the second one used the approximation asymptotic distribution. The powers of the tests are evaluated under the ARMA , VARMA , GARCH , and FDWN models with some illustrative simulation examples. One can run these simulation functions on PC with single CPU using the default argument `UseRmpi = FALSE` or on cluster computer with multiple CPU's using the argument `UseRmpi = TRUE` where the **Rmpi** package is installed.

Keywords: ARIMA models, GARCH models, FDWN models, Monte-Carlo significance test, Portmanteau test, VARIMA models .

1. Introduction

This vignette contains three sections. In each section we include two R scripts and some simulation examples. The main simulations were run on a computer with double quad core CPU's using the **Rmpi** package (Yu 2002). The empirical significance level and the power of the portmanteau test statistics proposed by Box and Pierce (1970); Ljung and Box (1978); Hosking (1980); Li and McLeod (1981); Peña and Rodriguez (2006); Lin and McLeod (2006); Mahdi and McLeod (2011) are calculated based on the approximation asymptotic distribution and the Monte-Carlo significance test.

The R script function `onesim.varima()` in Section 2 is used for calculating p-values of the test statistics using simulated series from ARMA or VARMA process `NREP` times, where `NREP` represents the number of replications in the Monte-Carlo method. The powers of the test for ARMA or VARMA models are calculated by the main simulation function, `simpower.varima()`, which implements `onesim.varima()` for `NumSim` times, where `NumSim` is the number needed for simulations.

In Section 3, the function `onesim.garch()` is used for calculating p-values of `gvtest` or `LjungBox` test using simulated series from univariate Generalized Autoregressive Conditional Heteroscedasticity, GARCH , process `NREP` times, where `NREP` is described as before. The power of the test is calculated by the main simulation function, `simpower.garch()`, which implements `onesim.garch()` for `NumSim` times.

Section 4 includes the function `onesim.fgwn()` that is used for calculating p-values of `gvtest` or LjungBox test using simulated series from univariate Fractional Difference White Noise, FDWN, process NREP times, where NREP is described as before. The power of the test is calculated by the main simulation function, simpower.fgwn(), which implements onesim.fgwn() for NumSim times.

```
R> library("portes")
R> library("Rmpi")
```

2. Power of Test for ARMA, VARMA Models

In this section we introduce the R functions simpower.varima() and onesim.varima() that can be used to reproduce the results reported by Peña and Rodriguez (2002, Table 3), Lin and McLeod (2006, Tables 3 and 6), and Mahdi and McLeod (2011, Figure 1 and Table 1).

The main simulation function in which we enter the parameters of the ARMA (p,q) or VARMA (p,q) models needed for generating a simulated data is the simpower.varima() function. This function calls the function onesim.varima() to fit AR (1) or VAR (1) models for this simulated data from ARMA (p,q) or VARMA (p,q) model respectively. The p-values of the portmanteau test statistics based on the Monte-Carlo method or the asymptotic distribution method is calculated by the function onesim.varima() where NREP is the number of Monte-Carlo replications and NREP equals to 1 in the asymptotic distribution case. These steps are repeated NumSim times, where NumSim is the number of simulations, and the powers of the test corresponding to the lag values are the final output of the function simpower.varima().

```
R> "simpower.varima" <- function(phi = NULL, theta = NULL, d = NA,
+     sigma, n, constant = NA, trend = NA, demean = NA, lags = seq(5,
+         30, 5), NREP = 1000, NumSim = 1000, test = c("gvtest",
+         "BoxPierce", "LjungBox", "Hosking", "LiMcLeod"), MonteCarlo = TRUE,
+     UseRmpi = FALSE, SquaredQ = FALSE, StableParameters = NA,
+     sig.Level = c(0.01, 0.05, 0.1)) {
+     test <- match.arg(test)
+     test <<- test
+     if (UseRmpi == FALSE) {
+         set.seed(2159734)
+         sim.stat <- replicate(NumSim, onesim.varima(phi = phi,
+             theta = theta, d = d, sigma = sigma, n = n, constant = constant,
+             trend = trend, demean = demean, lags = lags, NREP = NREP,
+             test = test, MonteCarlo = MonteCarlo, SquaredQ = SquaredQ,
+             StableParameters = StableParameters))
+     }
+     else {
+         mpi.spawn.Rslaves()
+         mpi.setup.rngstream(2159734)
+         mpi.bcast.Robj2slave(NumSim)
+         mpi.bcast.Robj2slave(NREP)
+         mpi.bcast.Robj2slave(phi)
```

```
+         mpi.bcast.Robj2slave(theta)
+         mpi.bcast.Robj2slave(sigma)
+         mpi.bcast.Robj2slave(n)
+         mpi.bcast.Robj2slave(d)
+         mpi.bcast.Robj2slave(constant)
+         mpi.bcast.Robj2slave(trend)
+         mpi.bcast.Robj2slave(demean)
+         mpi.bcast.Robj2slave(lags)
+         mpi.bcast.Robj2slave(test)
+         mpi.bcast.Robj2slave(MonteCarlo)
+         mpi.bcast.Robj2slave(SquaredQ)
+         mpi.bcast.Robj2slave(StableParameters)
+         if (all(!is.na(StableParameters))) {
+             mpi.bcast.cmd(library("akima"))
+             mpi.bcast.Robj2slave(interpp)
+             mpi.bcast.Robj2slave(interpp.old)
+             mpi.bcast.Robj2slave(fitstable)
+             mpi.bcast.Robj2slave(rstable)
+         }
+         mpi.bcast.Robj2slave(ToeplitzBlock)
+         mpi.bcast.Robj2slave(gvtest)
+         mpi.bcast.Robj2slave(BoxPierce)
+         mpi.bcast.Robj2slave(LjungBox)
+         mpi.bcast.Robj2slave(Hosking)
+         mpi.bcast.Robj2slave(LiMcLeod)
+         mpi.bcast.Robj2slave(ImpulseVMA)
+         mpi.bcast.Robj2slave(InvertQ)
+         mpi.bcast.Robj2slave(varima.sim)
+         mpi.bcast.Robj2slave(vma.sim)
+         mpi.bcast.Robj2slave(onesim.varima)
+         sim.stat <- mpi.parReplicate(NumSim, onesim.varima(phi = phi,
+             theta = theta, d = d, sigma = sigma, n = n, constant = constant,
+             trend = trend, demean = demean, lags = lags, NREP = NREP,
+             test = test, MonteCarlo = MonteCarlo, SquaredQ = SquaredQ,
+             StableParameters = StableParameters))
+         mpi.close.Rslaves()
+     }
+     m <- length(lags)
+     l <- length(sig.Level)
+     out <- matrix(numeric(m * l), ncol = m, nrow = l)
+     for (i in 1:l) {
+         if (is.matrix(sim.stat))
+             out[i, ] <- rowMeans(sim.stat <= sig.Level[i])
+         else out[i, ] <- mean(sim.stat <= sig.Level[i])
+     }
+     colnames(out) <- paste("Lag", lags, sep = " ")
+     rownames(out) <- paste(100 * sig.Level, "%", sep = "")
```

```
+     return(t(out))
+ }
R> "onesim.varima" <- function(phi = NULL, theta = NULL, d = NA,
+     sigma, n, constant = NA, trend = NA, demean = NA, lags = seq(5,
+         30, 5), NREP = 1000, test = c("gvtest", "BoxPierce",
+         "LjungBox", "Hosking", "LiMcLeod"), MonteCarlo = TRUE,
+     SquaredQ = FALSE, StableParameters = NA) {
+     test <- match.arg(test)
+     Trunc.Series <- min(100, ceiling(n/3))
+     sigma <- as.matrix(sigma)
+     k <- NCOL(sigma)
+     sim.data <- varima.sim(phi = phi, theta = theta, d = d, sigma = sigma,
+         n = n, constant = constant, trend = trend, demean = demean,
+         StableParameters = StableParameters, Trunc.Series = Trunc.Series)
+     if (all(phi == 0))
+         phi <- NULL
+     if (all(theta == 0))
+         theta <- NULL
+     p <- ifelse(is.null(phi), 0, length(phi))
+     q <- ifelse(is.null(theta), 0, length(theta))
+     if (p == 0 && q == 0) {
+         Order <- 0
+         res <- sim.data
+     }
+     else {
+         Order <- 1
+         fitvar1 <- ar.ols(sim.data, aic = FALSE, order.max = Order)
+         res <- ts(as.matrix(fitvar1$resid)[-Order, ])
+     }
+     if (MonteCarlo == FALSE) {
+         if (test == "gvtest")
+             sim.stat <- gvtest(res, lags, Order, SquaredQ)[,
+                 4]
+         else if (test == "BoxPierce")
+             sim.stat <- BoxPierce(res, lags, Order, SquaredQ)[,
+                 4]
+         else if (test == "LjungBox")
+             sim.stat <- LjungBox(res, lags, Order, SquaredQ)[,
+                 4]
+         else if (test == "Hosking")
+             sim.stat <- Hosking(res, lags, Order, SquaredQ)[,
+                 4]
+         else if (test == "LiMcLeod")
+             sim.stat <- LiMcLeod(res, lags, Order, SquaredQ)[,
+                 4]
+         ans <- sim.stat
+     }
```

```
+     else {
+         if (test == "gvtest")
+             obs.stat <- gvtest(res, lags, Order, SquaredQ)[,
+                 2]
+         else if (test == "BoxPierce")
+             obs.stat <- BoxPierce(res, lags, Order, SquaredQ)[,
+                 2]
+         else if (test == "LjungBox")
+             obs.stat <- LjungBox(res, lags, Order, SquaredQ)[,
+                 2]
+         else if (test == "Hosking")
+             obs.stat <- Hosking(res, lags, Order, SquaredQ)[,
+                 2]
+         else if (test == "LiMcLeod")
+             obs.stat <- LiMcLeod(res, lags, Order, SquaredQ)[,
+                 2]
+         if (Order == 0) {
+             phi <- NULL
+             theta <- NULL
+             sigma <- matrix(acf(res, lag.max = 1, plot = FALSE,
+                 type = "covariance")$acf[1, , ], k, k)
+         }
+         else {
+             theta <- NULL
+             sigma <- as.matrix(fitvar1$var.pred)
+             if (is.array(fitvar1$ar)) {
+                 arrayphi <- array(numeric(k * k * 1), dim = c(k^2,
+                   1))
+                 arrayphi[, 1] <- c(fitvar1$ar[1, , ])
+                 phi <- array(c(arrayphi), dim = c(k, k, 1))
+             }
+             else phi <- fitvar1$ar
+             if (!is.null(fitvar1$x.intercept))
+                 constant <- fitvar1$x.intercept
+             else constant <- rep(0, k)
+             if (is.na(trend))
+                 trend <- rep(0, k)
+             demean <- fitvar1$x.mean
+         }
+         count <- rep(0, length(lags))
+         for (i in 1:NREP) {
+             bootdata <- varima.sim(phi = phi, theta = theta,
+                 sigma = sigma, d = d, n = n, constant = constant,
+                 trend = trend, demean = demean, StableParameters = StableParameters,
+                 Trunc.Series = Trunc.Series)
+             p <- ifelse(is.null(phi), 0, length(phi))
+             q <- ifelse(is.null(theta), 0, length(theta))
```

```
+             if (p == 0 && q == 0) {
+                 Order <- 0
+                 rboot <- bootdata
+             }
+             else {
+                 Order <- 1
+                 FitSimModel <- ar.ols(bootdata, aic = FALSE,
+                   order.max = Order)
+                 rboot <- ts(as.matrix(FitSimModel$resid)[-Order,
+                   ])
+             }
+             if (test == "gvtest")
+                 sim.stat <- gvtest(rboot, lags, Order, SquaredQ)[,
+                   2]
+             else if (test == "BoxPierce")
+                 sim.stat <- BoxPierce(rboot, lags, Order, SquaredQ)[,
+                   2]
+             else if (test == "LjungBox")
+                 sim.stat <- LjungBox(rboot, lags, Order, SquaredQ)[,
+                   2]
+             else if (test == "Hosking")
+                 sim.stat <- Hosking(rboot, lags, Order, SquaredQ)[,
+                   2]
+             else if (test == "LiMcLeod")
+                 sim.stat <- LiMcLeod(rboot, lags, Order, SquaredQ)[,
+                   2]
+             count <- count + (sim.stat >= obs.stat)
+         }
+         ans <- (count + 1)/(NREP + 1)
+     }
+     names(ans) <- lags
+     return(ans)
+ }
```

where

phi is a numeric or an array of `AR` or an array of `VAR` parameters with order p.

theta is a numeric or an array of `MA` or an array of `VMA` parameters with order q.

d is an integer or a vector representing the order of the difference.

sigma is the variance of white noise series and must be entered as matrix in case of bivariate or multivariate time series.

n is the length of the series.

constant a numeric vector represents the intercept in the deterministic equation.

trend a numeric vector represents the slop in the deterministic equation.

demean a numeric vector represents the mean of the series.

lags is the vector of lag values.

NREP is the number of Monte-Carlo replications.

NumSim is the number of simulations.

test is the test statistic to be used.

MonteCarlo if `TRUE` then apply the Monte-Carlo version of the test statistic. Otherwise, apply the asymptotic chi-square distribution test.

UseRmpi If `TRUE` then use parallel computing implemented in `"Rmpi"` package. Otherwise use only one CPU.

SquaredQ when it is `TRUE` then apply the test to the squared values in the simulation procedures. Otherwise apply for the usual residuals.

StableParameters is the four stable parameters, `ALPHA, BETA, GAMMA,` and `DELTA`.

2.1. Simulation Example 1

In this example, we study the empirical significance level of `gvtest` test under the first-order autoregressive models using the Monte-Carlo method as given in Lin and McLeod (2006, Table 3). In addition, we consider the empirical significance level of `gvtest` test using the asymptotic chi-square distribution method. For simplicity we introduce the following two codes for Monte-Carlo test method and asymptotic distribution test method respectively. We choose $\phi = 0.5$ with series length $n = 200$ and `NREP = 100`, `NumSim = 1000` to get some timings,

Monte-Carlo of gvtest Test

```
R> NREP <- 10^2
R> NumSim <- 10^3
R> phi <- 0.5
R> theta <- NULL
R> d <- NA
R> sigma <- 1
R> n <- 200
R> constant <- NA
R> trend <- NA
R> demean <- NA
R> UseRmpi <- TRUE
R> lags <- c(10, 20)
R> Start1 <- proc.time()[3]
R> simpower.varima(phi, theta, d, sigma, n, constant, trend, demean,
+     lags, NREP, NumSim, "gvtest", MonteCarlo = TRUE, UseRmpi = UseRmpi)
```

```
        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          1%    5%   10%
Lag 10 0.011 0.050 0.100
Lag 20 0.012 0.056 0.106
```

```
R> End1 <- proc.time()[3]
R> Total1 <- End1 - Start1
R> Total1
```

```
elapsed
   1465
```

Asymptotic Distribution of gvtest Test

```
R> Start2 <- proc.time()[3]
R> simpower.varima(phi, theta, d, sigma, n, constant, trend, demean,
+     lags, NREP, NumSim, "gvtest", MonteCarlo = FALSE, UseRmpi = UseRmpi)
```

```
        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          1%    5%   10%
Lag 10 0.007 0.035 0.079
Lag 20 0.004 0.042 0.085
```

```
R> End2 <- proc.time()[3]
R> Total2 <- End2 - Start2
R> Total2
```

```
elapsed
  15.51
```

2.2. Simulation Example 2

The power of the generalized variance portmanteau test, `gvtest`, is compared with its competitor `LjungBox` using the twelve ARMA (2,2) models given in Peña and Rodriguez (2002, Table 3) and Lin and McLeod (2006, Table 6). For simplicity, we introduce the following R code based on 10^3 simulations, and 10^2 replications and full results of simulations are given in Table 1.

```
R> Model <- "Model 1"
R> NREP <- 10^2
R> NumSim <- 10^3
R> phi <- NULL
R> theta <- c(-0.5)
R> d <- NA
R> sigma <- 1
R> n <- 100
R> lags <- c(10, 20)
R> constant <- NA
R> trend <- NA
R> demean <- NA
R> sig.Level <- 0.05
R> Start3 <- proc.time()[3]
R> simpower.varima(phi, theta, d, sigma, n, constant, trend, demean,
+     lags, NREP, NumSim, "gvtest", MonteCarlo = TRUE, UseRmpi = UseRmpi,
+     sig.Level = sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          5%
Lag 10 0.421
Lag 20 0.340

R> End3 <- proc.time()[3]
R> Total3 <- End3 - Start3
R> Total3
```

Table 1: Power levels of `gvtest`, $\mathfrak{D}_m$, and `LjungBox`, Q_m, tests when AR (1) model is fitted to the twelve ARMA (2,2) models given by Peña and Rodriguez (2002, Table 3) and Lin and McLeod (2006, Table 6) based on 1000 simulations. Each simulation used 1000 Monte-Carlo test.

	Monte-Carlo Method				Asymptotic Distribution Method			
	$m = 10$		$m = 20$		$m = 10$		$m = 20$	
Model	$\mathfrak{D}_m$	Q_m	$\mathfrak{D}_m$	Q_m	$\mathfrak{D}_m$	Q_m	$\mathfrak{D}_m$	Q_m
1	0.421	0.264	0.341	0.225	0.378	0.253	0.257	0.210
2	0.991	0.758	0.968	0.608	0.988	0.768	0.959	0.615
3	0.998	0.795	0.994	0.608	0.994	0.783	0.985	0.628
4	0.624	0.461	0.516	0.387	0.531	0.433	0.390	0.339
5	0.840	0.692	0.771	0.615	0.852	0.691	0.704	0.607
6	0.805	0.548	0.716	0.427	0.792	0.567	0.625	0.460
7	1.000	0.987	1.000	0.940	1.000	0.990	1.000	0.929
8	0.997	0.814	0.992	0.669	0.999	0.844	0.994	0.667
9	0.257	0.189	0.202	0.154	0.186	0.170	0.115	0.128
10	0.879	0.765	0.813	0.652	0.821	0.745	0.712	0.628
11	0.601	0.341	0.490	0.264	0.582	0.340	0.408	0.271
12	0.910	0.626	0.869	0.495	0.880	0.614	0.803	0.479

```
elapsed
 745.71
```

2.3. Simulation Example 3

In the multivariate time series, we consider Model 1 from the eight models given by Mahdi and McLeod (2011). The length series $n = 100$, with 10^3 simulations, and 10^2 replications are chosen to get some timings.

Monte-Carlo of gvtest Test

```
R> NREP <- 10^2
R> NumSim <- 10^3
R> phi <- array(c(0.5, 0.4, 0.1, 0.5, 0, 0.3, 0, 0), dim = c(2,
+     2, 2))
R> theta <- NULL
R> d <- NA
R> sigma <- matrix(c(1, 0.71, 0.71, 1), 2, 2)
R> n <- 100
R> constant <- NA
R> trend <- NA
R> demean <- NA
R> UseRmpi <- TRUE
```

```
R> lags <- seq(5, 30, 5)
R> sig.Level <- c(0.01, 0.05, 0.1)
R> Start4 <- proc.time()[3]
R> simpower.varima(phi, theta, d, sigma, n, constant, trend, demean,
+     lags, NREP, NumSim, "gvtest", MonteCarlo = TRUE, UseRmpi = UseRmpi,
+     sig.Level = sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          1%    5%   10%
Lag 5  0.407 0.684 0.791
Lag 10 0.296 0.565 0.679
Lag 15 0.225 0.472 0.609
Lag 20 0.175 0.393 0.540
Lag 25 0.137 0.346 0.485
Lag 30 0.115 0.321 0.442

R> End4 <- proc.time()[3]
R> Total4 <- End4 - Start4
R> Total4

elapsed
 1196.5
```

Monte-Carlo of Hosking Test

```
R> Start5 <- proc.time()[3]
R> simpower.varima(phi, theta, d, sigma, n, constant, trend, demean,
+     lags, NREP, NumSim, "Hosking", MonteCarlo = TRUE, UseRmpi = UseRmpi,
+     sig.Level = sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
```

```
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          1%    5%   10%
Lag 5  0.266 0.507 0.642
Lag 10 0.143 0.333 0.476
Lag 15 0.100 0.257 0.375
Lag 20 0.074 0.216 0.320
Lag 25 0.058 0.196 0.292
Lag 30 0.053 0.162 0.260
```

```
R> End5 <- proc.time()[3]
R> Total5 <- End5 - Start5
R> Total5
```

```
elapsed
 535.05
```

3. Power of Test for GARCH Models

In this section we introduce the two R functions, `simpower.garch()` and `onesim.garch()`. The idea of these functions is similar to that idea of `simpower.varima()` and `onesim.varima()` respectively. The power of `gvtest` and `LjungBox` tests for GARCH models based on the two mentioned methods, the Monte-Carlo method and the asymptotic distribution method, can be evaluated using these two functions. Users need to have **fGarch** library installed in order to reproduce the results reported by Lin and McLeod (2006, Table 4).

```
R> library("fGarch")
```

```
R> "simpower.garch" <- function(omega = 1, alpha = 0.05, beta = 0.9,
+     n, lags = seq(5, 30, 5), NREP = 1000, NumSim = 1000, statistic = c("gvtest",
+         "LjungBox"), MonteCarlo = TRUE, UseRmpi = FALSE, SquaredQ = FALSE,
+     sig.Level = c(0.01, 0.05, 0.1)) {
+     statistic <- match.arg(statistic)
+     statistic <<- statistic
+     if (UseRmpi == FALSE) {
+         set.seed(2159734)
+         sim.stat <- replicate(NumSim, onesim.garch(omega = omega,
+             alpha = alpha, beta = beta, n = n, lags = lags, NREP = NREP,
+             statistic = statistic, MonteCarlo = MonteCarlo, SquaredQ = SquaredQ))
+     }
+     else {
+         mpi.spawn.Rslaves()
+         mpi.setup.rngstream(2159734)
```

```
+         mpi.bcast.cmd(library("fGarch"))
+         mpi.bcast.Robj2slave(NumSim)
+         mpi.bcast.Robj2slave(NREP)
+         mpi.bcast.Robj2slave(omega)
+         mpi.bcast.Robj2slave(alpha)
+         mpi.bcast.Robj2slave(beta)
+         mpi.bcast.Robj2slave(n)
+         mpi.bcast.Robj2slave(lags)
+         mpi.bcast.Robj2slave(statistic)
+         mpi.bcast.Robj2slave(MonteCarlo)
+         mpi.bcast.Robj2slave(SquaredQ)
+         mpi.bcast.Robj2slave(ToeplitzBlock)
+         mpi.bcast.Robj2slave(gvtest)
+         mpi.bcast.Robj2slave(LjungBox)
+         mpi.bcast.Robj2slave(ImpulseVMA)
+         mpi.bcast.Robj2slave(onesim.garch)
+         sim.stat <- mpi.parReplicate(NumSim, onesim.garch(omega = omega,
+             alpha = alpha, beta = beta, n = n, lags = lags, NREP = NREP,
+             statistic = statistic, MonteCarlo = MonteCarlo, SquaredQ = SquaredQ))
+         mpi.close.Rslaves()
+     }
+     m <- length(lags)
+     l <- length(sig.Level)
+     out <- matrix(numeric(m * l), ncol = m, nrow = l)
+     for (i in 1:l) {
+         if (is.matrix(sim.stat))
+             out[i, ] <- rowMeans(sim.stat <= sig.Level[i])
+         else out[i, ] <- mean(sim.stat <= sig.Level[i])
+     }
+     colnames(out) <- paste("Lag", lags, sep = " ")
+     rownames(out) <- paste(100 * sig.Level, "%", sep = "")
+     return(t(out))
+ }
R> "onesim.garch" <- function(omega = 1, alpha = 0.05, beta = 0.9,
+     n, lags = seq(5, 30, 5), NREP = 1000, statistic = c("gvtest",
+         "LjungBox"), MonteCarlo = TRUE, SquaredQ = FALSE) {
+     statistic <- match.arg(statistic)
+     spec = garchSpec(model = list(omega = omega, alpha = alpha,
+         beta = beta))
+     sim.data <- garchSim(spec, n = n)$garch
+     if (MonteCarlo == FALSE) {
+         if (statistic == "gvtest")
+             sim.stat <- gvtest(sim.data, lags, 0, SquaredQ)[,
+                 4]
+         else sim.stat <- LjungBox(sim.data, lags, 0, SquaredQ)[,
+             4]
+         ans <- sim.stat
```

```
+     }
+     else {
+         if (statistic == "gvtest")
+             obs.stat <- gvtest(sim.data, lags, 0, SquaredQ)[,
+                 2]
+         else obs.stat <- LjungBox(sim.data, lags, 0, SquaredQ)[,
+             2]
+         count <- rep(0, length(lags))
+         for (i in 1:NREP) {
+             bootdata <- rnorm(n)
+             if (statistic == "gvtest")
+                 sim.stat <- gvtest(bootdata, lags, 0, SquaredQ)[,
+                   2]
+             else sim.stat <- LjungBox(bootdata, lags, 0, SquaredQ)[,
+                 2]
+             count <- count + (sim.stat >= obs.stat)
+         }
+         ans <- (count + 1)/(NREP + 1)
+     }
+     names(ans) <- lags
+     return(ans)
+ }
```

where

omega is the constant coefficient of the variance equation in GARCH model.

alpha is the value or vector of autoregressive coefficients.

beta is the value or vector of variance coefficients.

The other arguments are described as before.

3.1. Simulation Example 4

In this example we introduce the codes that can be implemented for reproducing the results reported by Peña and Rodriguez (2002, Table 9) and Lin and McLeod (2006, Table 4). Table 2 shows the results of comparing the power of `gvtest` based on the Monte-Carlo method with the corresponding one based on the asymptotic chi-square distribution for the two GARCH models given in Peña and Rodriguez (2002, Table 9) and Lin and McLeod (2006, Table 4). For simplicity, we choose model **A** with $n = 250$ in the the following two codes for both methods and users can use these codes to replicate the results of Table 2 by changing the parameters of the model and the sample size.

Monte-Carlo of gvtest Test

```
R> NREP <- 10^3
R> NumSim <- 10^3
```

Table 2: Power comparison of Monte-Carlo test and asymptotic chi-square distribution of `gvtest` portmanteau test for GARCH models as given by Peña and Rodriguez (2002, Table 9) and Lin and McLeod (2006, Table 4). The Monte-Carlo test is based 1000 simulations and 1000 replications. The asymptotic chi-square distribution is based on 1000 simulations.

Model	n	Monte-Carlo Method			Asymptotic Distribution Method		
		$m = 12$	$m = 24$	$m = 32$	$m = 12$	$m = 24$	$m = 32$
A	250	0.302	0.303	0.304	0.289	0.277	0.258
B	250	0.855	0.823	0.808	0.842	0.814	0.779
A	500	0.505	0.513	0.495	0.513	0.518	0.493
B	500	0.989	0.985	0.983	0.985	0.980	0.976
A	1000	0.820	0.818	0.800	0.812	0.816	0.807
B	1000	1.000	1.000	1.000	1.000	1.000	1.000

```
R> alpha <- 0.05
R> beta <- 0.9
R> n <- 250
R> lags <- c(12, 24, 32)
R> sig.Level <- 0.05
R> UseRmpi <- TRUE
R> Start6 <- proc.time()[3]
R> simpower.garch(omega = 1, alpha = alpha, beta = beta, n, lags,
+     NREP, NumSim, "gvtest", MonteCarlo = TRUE, UseRmpi = UseRmpi,
+     SquaredQ = TRUE, sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          5%
Lag 12 0.302
Lag 24 0.303
Lag 32 0.304

R> End6 <- proc.time()[3]
R> Total6 <- End6 - Start6
R> Total6
```

```
elapsed
8405.58
```

Asymptotic Distribution of gvtest Test

```
R> Start7 <- proc.time()[3]
R> simpower.garch(omega = 1, alpha = alpha, beta = beta, n, lags,
+     NREP, NumSim, "gvtest", MonteCarlo = FALSE, UseRmpi = UseRmpi,
+     SquaredQ = TRUE, sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          5%
Lag 12 0.289
Lag 24 0.277
Lag 32 0.258

R> End7 <- proc.time()[3]
R> Total7 <- End7 - Start7
R> Total7

elapsed
  20.82
```

4. Power of Test for FDWN Models

Finally we introduce the following two R functions, `simpower.fgwn()` and `onesim.fgwn()` that can be used to reproduce the results reported by Lin and McLeod (2006, Tables 5). The idea of these functions is similar to that of `simpower.varima()` and `onesim.varima()` respectively. The power of `gvtest` and `LjungBox` tests for FDWN models based on the two methods, the Monte-Carlo method and the asymptotic distribution method, can be evaluated using these two functions.

```
R> "simpower.fgwn" <- function(d, n, lags = seq(5, 30, 5), NREP = 1000,
+     NumSim = 1000, statistic = c("gvtest", "LjungBox"), MonteCarlo = TRUE,
+     UseRmpi = FALSE, SquaredQ = FALSE, sig.Level = c(0.01, 0.05,
```

```
+         0.1)) {
+     statistic <- match.arg(statistic)
+     statistic <<- statistic
+     if (UseRmpi == FALSE) {
+         set.seed(2159734)
+         sim.stat <- replicate(NumSim, onesim.fgwn(d = d, n = n,
+             lags = lags, NREP = NREP, statistic = statistic,
+             MonteCarlo = MonteCarlo, SquaredQ = SquaredQ))
+     }
+     else {
+         mpi.spawn.Rslaves()
+         mpi.setup.rngstream(2159734)
+         mpi.bcast.cmd(library("ltsa"))
+         mpi.bcast.Robj2slave(NumSim)
+         mpi.bcast.Robj2slave(NREP)
+         mpi.bcast.Robj2slave(d)
+         mpi.bcast.Robj2slave(n)
+         mpi.bcast.Robj2slave(lags)
+         mpi.bcast.Robj2slave(statistic)
+         mpi.bcast.Robj2slave(MonteCarlo)
+         mpi.bcast.Robj2slave(SquaredQ)
+         mpi.bcast.Robj2slave(ToeplitzBlock)
+         mpi.bcast.Robj2slave(gvtest)
+         mpi.bcast.Robj2slave(LjungBox)
+         mpi.bcast.Robj2slave(ImpulseVMA)
+         mpi.bcast.Robj2slave(onesim.fgwn)
+         sim.stat <- mpi.parReplicate(NumSim, onesim.fgwn(d = d,
+             n = n, lags = lags, NREP = NREP, statistic = statistic,
+             MonteCarlo = MonteCarlo, SquaredQ = SquaredQ))
+         mpi.close.Rslaves()
+     }
+     m <- length(lags)
+     l <- length(sig.Level)
+     out <- matrix(numeric(m * l), ncol = m, nrow = l)
+     for (i in 1:l) {
+         if (is.matrix(sim.stat))
+             out[i, ] <- rowMeans(sim.stat <= sig.Level[i])
+         else out[i, ] <- mean(sim.stat <= sig.Level[i])
+     }
+     colnames(out) <- paste("Lag", lags, sep = " ")
+     rownames(out) <- paste(100 * sig.Level, "%", sep = "")
+     return(t(out))
+ }
R> "onesim.fgwn" <- function(d, n, lags = seq(5, 30, 5), NREP = 1000,
+     statistic = c("gvtest", "LjungBox"), MonteCarlo = TRUE, SquaredQ = FALSE) {
+     statistic <- match.arg(statistic)
+     r <- numeric(n)
```

```
+     r[1] <- gamma(1 - 2 * d)/gamma(1 - d)^2
+     for (i in 1:(n - 1)) r[i + 1] <- ((i - 1 + d)/(i - d)) *
+         r[i]
+     sim.data <- DHSimulate(n, r)
+     if (MonteCarlo == FALSE) {
+         if (statistic == "gvtest")
+             sim.stat <- gvtest(sim.data, lags, 0, SquaredQ)[,
+                 4]
+         else sim.stat <- LjungBox(sim.data, lags, 0, SquaredQ)[,
+             4]
+         ans <- sim.stat
+     }
+     else {
+         if (statistic == "gvtest")
+             obs.stat <- gvtest(sim.data, lags, 0, SquaredQ)[,
+                 2]
+         else obs.stat <- LjungBox(sim.data, lags, 0, SquaredQ)[,
+             2]
+         count <- rep(0, length(lags))
+         for (i in 1:NREP) {
+             bootdata <- rnorm(n)
+             if (statistic == "gvtest")
+                 sim.stat <- gvtest(bootdata, lags, 0, SquaredQ)[,
+                   2]
+             else sim.stat <- LjungBox(bootdata, lags, 0, SquaredQ)[,
+                 2]
+             count <- count + (sim.stat >= obs.stat)
+         }
+         ans <- (count + 1)/(NREP + 1)
+     }
+     names(ans) <- lags
+     return(ans)
+ }
```

where d is the fractional difference parameter.

4.1. Simulation Example 5

This example presents the R code which can be used to reproduce the results reported by Lin and McLeod (2006, Table 5). For simplicity we pick $d = 0.2$ with $n = 256$ and users need to re-run the code after replacing d and n by all possible values given in Table 3 in order to replicate the results of this table.

Monte-Carlo gvtest Test

```
R> NREP <- 10^3
R> NumSim <- 10^3
```

Table 3: The empirical power of `gvtest` and `LjungBox` test statistics for Fractional Difference White Noise, FDWN model as described in Lin and McLeod (2006, Table 5), based on 1000 simulations and 1000 replications. The first entry corresponds to `gvtest` and the second one to `LjungBox`, where $n = 256, 512$ and $d = 0.2, 0.3$.

d	n	$m = 5$	$m = 10$	$m = 20$	$m = 30$	$m = 40$
0.2	256	0.212/0.188	0.202/0.16	0.170/0.134	0.155/0.122	0.149/0.111
0.3	256	0.692/0.644	0.644/0.593	0.597/0.526	0.560/0.497	0.532/0.460
0.2	512	0.346/0.300	0.306/0.246	0.256/0.198	0.218/0.175	0.213/0.162
0.3	512	0.914/0.884	0.892/0.837	0.847/0.790	0.818/0.750	0.801/0.729

```
R> n <- 256
R> d <- 0.2
R> lags <- c(5, seq(10, 40, 10))
R> sig.Level <- 0.05
R> UseRmpi <- TRUE
R> Start8 <- proc.time()[3]
R> simpower.fgwn(d, n, lags, NREP, NumSim, "gvtest", MonteCarlo = TRUE,
+     UseRmpi = UseRmpi, SquaredQ = TRUE, sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          5%
Lag 5  0.212
Lag 10 0.202
Lag 20 0.170
Lag 30 0.155
Lag 40 0.149

R> End8 <- proc.time()[3]
R> Total8 <- End8 - Start8
R> Total8

 elapsed
12294.42
```

Monte-Carlo LjungBox Test

```
R> Start9 <- proc.time()[3]
R> simpower.fgwn(d, n, lags, NREP, NumSim, "LjungBox", MonteCarlo = TRUE,
+     UseRmpi = UseRmpi, SquaredQ = TRUE, sig.Level)

        8 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 9 is running on: stats-c-emim
slave1 (rank 1, comm 1) of size 9 is running on: stats-c-emim
slave2 (rank 2, comm 1) of size 9 is running on: stats-c-emim
slave3 (rank 3, comm 1) of size 9 is running on: stats-c-emim
slave4 (rank 4, comm 1) of size 9 is running on: stats-c-emim
slave5 (rank 5, comm 1) of size 9 is running on: stats-c-emim
slave6 (rank 6, comm 1) of size 9 is running on: stats-c-emim
slave7 (rank 7, comm 1) of size 9 is running on: stats-c-emim
slave8 (rank 8, comm 1) of size 9 is running on: stats-c-emim
          5%
Lag 5  0.188
Lag 10 0.160
Lag 20 0.134
Lag 30 0.126
Lag 40 0.111

R> End9 <- proc.time()[3]
R> Total9 <- End9 - Start9
R> Total9

elapsed
1378.85
```

References

Box G, Pierce D (1970). "Distribution of Residual Autocorrelation in Autoregressive-Integrated Moving Average Time Series Models." *Journal of American Statistical Association*, **65**, 1509–1526.

Hosking JRM (1980). "The Multivariate Portmanteau Statistic." *Journal of American Statistical Association*, **75**(371), 602–607.

Li WK, McLeod AI (1981). "Distribution of the Residual Autocorrelation in Multivariate ARMA Time Series Models." *Journal of the Royal Statistical Society, Series B*, **43**(2), 231–239.

Lin J, McLeod AI (2006). "Improved Peña-Rodriguez Portmanteau Test." *Computational Statistics and Data Analysis*, **51**(3), 1731–1738.

Ljung G, Box G (1978). "On a Measure of Lack of Fit in Time Series Models." *Biometrika*, **65**, 297–303.

Mahdi E, McLeod AI (2011). "Improved Multivariate Portmanteau Diagnostic Test." Submitted.

Peña D, Rodriguez J (2002). "A Powerful Portmanteau Test of Lack of Test for Time Series." *Journal of American Statistical Association*, **97**(458), 601–610.

Peña D, Rodriguez J (2006). "The Log of the Determinant of the Autocorrelation Matrix for Testing Goodness of Fit in Time Series." *Journal of Statistical Planning and Inference*, **8**(136), 2706–2718.

Yu H (2002). "Rmpi: Parallel Statistical Computing in R." *R News*, **2**(2), 10–14. URL `http://CRAN.R-project.org/doc/Rnews/`.

Affiliation:

A. Ian McLeod
Department of Statistical and Actuarial Sciences
University of Western Ontario
E-mail: `aim@stats.uwo.ca`
URL: `http://www.stats.uwo.ca/mcleod`
Esam Mahdi
Department of Statistical and Actuarial Sciences
University of Western Ontario
E-mail: `emahdi@uwo.ca`