SC2 Coursework 2: Armadillo and Parallel

Dom Owens

13/05/2020

In this document, we outline an efficient implementation of the Wald-type procedure for multiple change-point analysis of multiple time series, as outlined in chapter 3 of MOSUM Methods for Multiple Change-Point Analysis in Causal Networks.

RcppArmadillo Implementation

The original implementation relies largely on base-R functionality and a few functions from popular packages (e.g. `stats`), and takes an extremely long time even in problems of moderate dimension. The new implementation is written entirely in `RcppArmadillo`, which performs large-dimensional linear algebra calculations at a compiled level. Given our procedure consists of many matrix calculations, we should expect to see a strong improvement in performance.

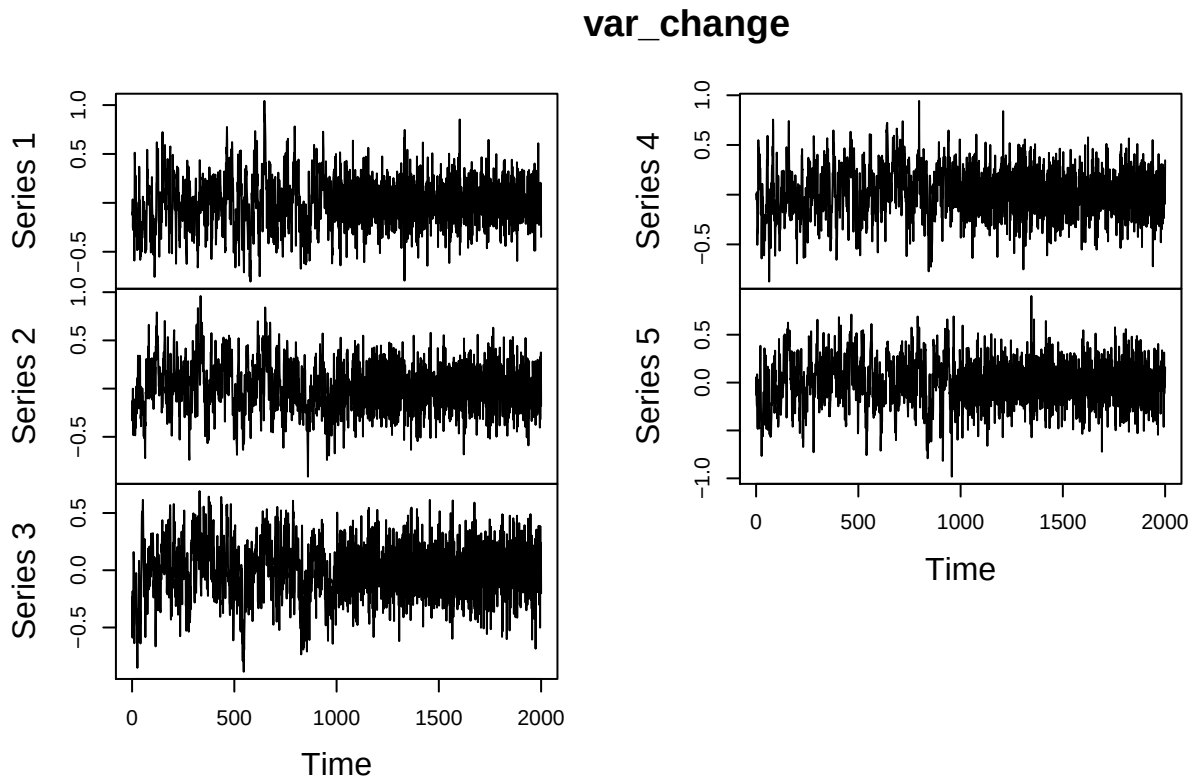
The relevant `Rcpp` code can be viewed here.

Runtime Comparison

We compare how our implementation performs in terms of runtime against the previous base-R implementation.

First, we generate an example set of data from a VAR process, with dimension $d = 5$ and lag $p = 1$, and a change in structure at time $t = 1000$.

```
source("VAR_sim.R") #load VAR data simulator
a1 <- diag(0.6, nrow = 5, ncol = 5 ) + 0.05  #regime 1
e1 <- matrix(rnorm(5 * 1000, 0, 0.2),ncol=5)
a2 <- diag(-0.4, nrow = 5, ncol = 5 ) - 0.03 #regime 2
e2 <- matrix(rnorm(5 * 1000, 0, 0.2),ncol=5)
rData1 <- rSim(a1, e1)
#rData1[1,] <- runif(5, 0, 0.02) #prevent NaN
rData2 <- rSim(a2, e2)
#rData2[1,] <- runif(5, 0, 0.02)
var_change <- ts(rbind(rData1+ 0, rData2- 0) )
plot(var_change)
```

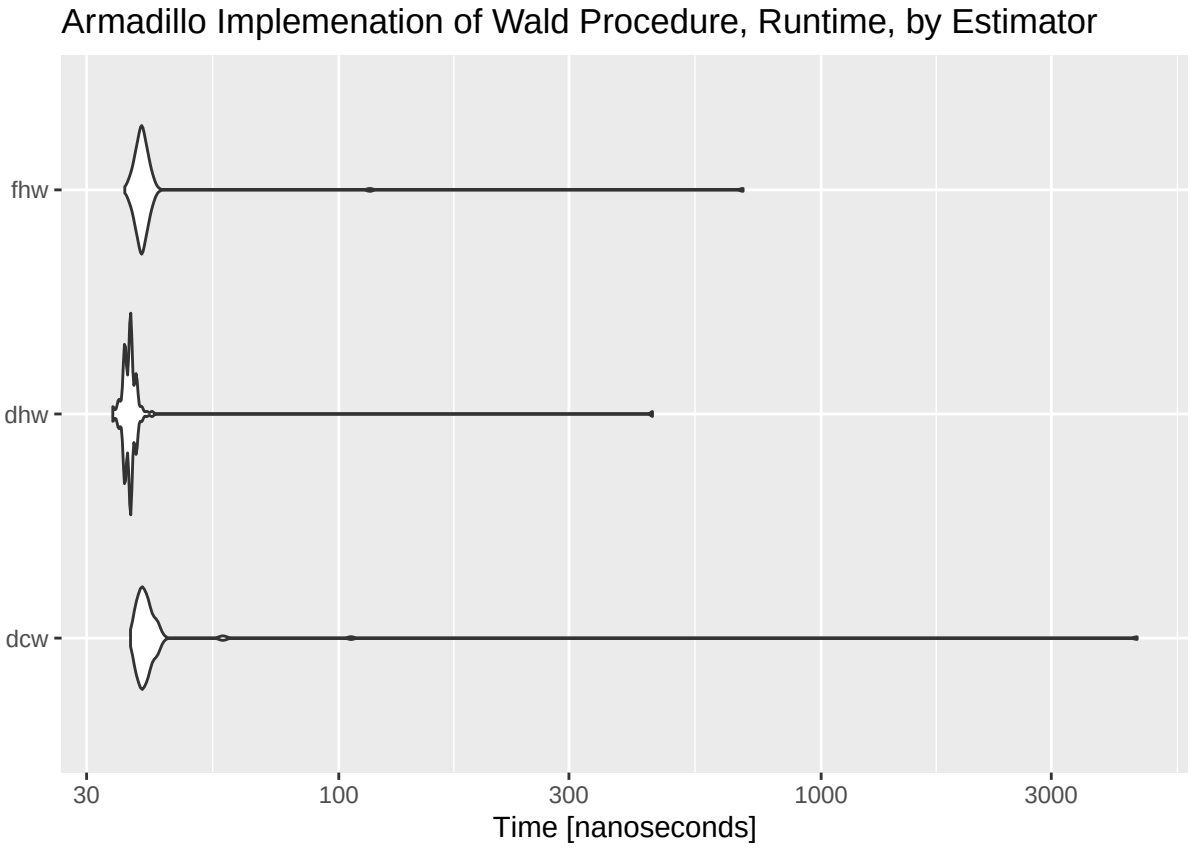



```
sourceCpp(file = "Wald_RcppParallel.cpp")
dcw <- function()test_Wald_RCPP(x=var_change, p=1, G=200, alpha = 0.05, estim = "DiagC")
dhw <- function()test_Wald_RCPP(x=var_change, p=1, G=200, alpha = 0.05, estim = "DiagH")
fhw <- function()test_Wald_RCPP(x=var_change, p=1, G=200, alpha = 0.05, estim = "FullH")
mb <- microbenchmark(dcw, dhw, fhw, times = 100)
print(mb)
```

```
## Unit: nanoseconds
##  expr min lq  mean median uq  max neval
##   dcw  37 39 85.43   39.5 41 4521   100
##   dhw  34 36 40.89   37.0 37  447   100
##   fhw  36 38 46.27   39.0 40  689   100
```

```
autoplot(mb) + ggtitle("Armadillo Implemenation of Wald Procedure, Runtime, by Estimator")
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



We can see that these methods take around 50 nanoseconds to compute.

The base-R methods take between 10 and 150 seconds to compute (see Figure 1), which is of the order of **one billion** times as long. The original code contains multiple embedded for-loops, which makes up the majority of the time taken.

Constituent functions

In this section, we compare each Rcpp function to its' corresponding base R function, to ensure the procedure is doing what we expect it to.

We use a new example with $p = 2$.

```
p2_Data1 <- rSim_p2(a1, a2, e1)
p2_Data2 <- rSim_p2(a2, -a1, e2)
p2_change <- ts(rbind(p2_Data1, p2_Data2) )
plot(p2_change)
```

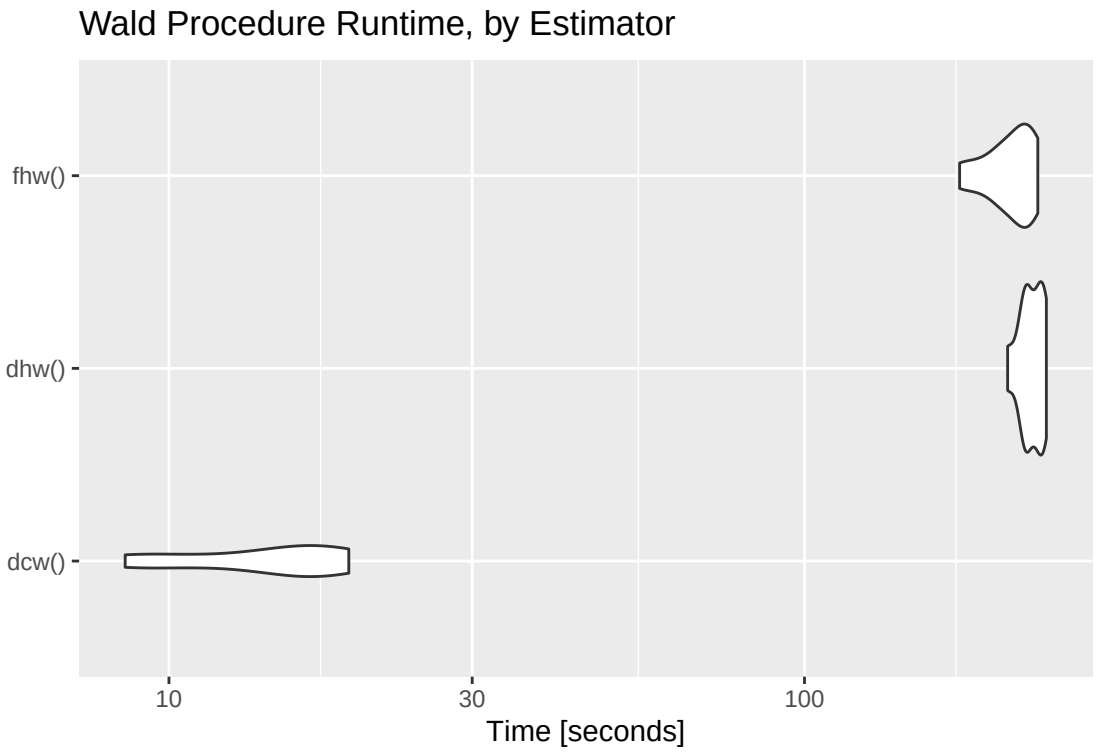
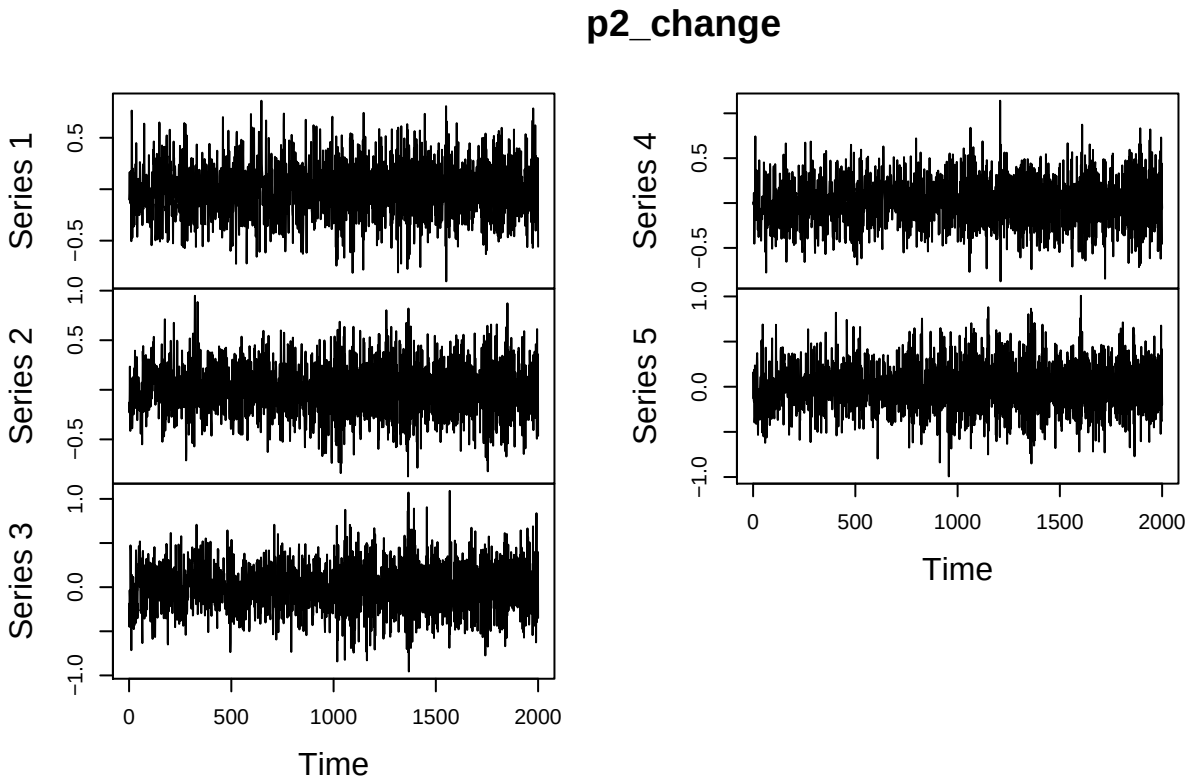


Figure 1: Base R Implementation Runtime



Regression parameter estimators

$$\tilde{\boldsymbol{a}}_{l,u}$$

(3.19)

```
ai <-get_a_lu_i(p2_change, i=1,p=2, 10, 100) ;aiC <-get_a_lu_i_RCPP(p2_change, i=1,p=2, 10, 100)
max(abs(ai-aiC))
```

```
## [1] 2.775558e-16
```

```
head(aiC)
```

```
##             [,1]
## [1,] -0.0237932
## [2,]  0.3757309
## [3,] -0.2089523
## [4,]  0.1111389
## [5,] -0.1295106
## [6,]  0.3141322
```

$$\tilde{\boldsymbol{a}}_{l,u}$$

(3.18)

```
ap2 <- make_a_lu(p2_change, p=2, l= 11, u= 100); ap2C <- make_a_lu_RCPP(p2_change, p=2, 11, 100)
max(abs(ap2-ap2C))
```

```
## [1] 3.885781e-16
```

```
head(ap2C)
```

```
##              [,1]
## [1,] -0.02494047
## [2,]  0.36991097
## [3,] -0.20359665
## [4,]  0.11064483
## [5,] -0.12865064
## [6,]  0.31985505
```

Estimating functions

$$\boldsymbol{H}_i$$

(3.11)

```
H_ik <- getH_ik_Wald(p2_change, i=1, k=100,p=2, a = ap2 ); H_ikC <- getH_ik_Wald_RCPP(p2_change, i=1, k=
max(abs(H_ik-H_ikC))
```

```
## [1] 3.330669e-16
```

```
head(H_ikC)
```

```
##              [,1]
## [1,]  0.49611101
## [2,] -0.10012090
## [3,] -0.08039155
## [4,]  0.04974965
## [5,]  0.05748852
## [6,] -0.01655661
```

$$\boldsymbol{H}$$

(3.12)

```
H_k <- makeH_k_Wald(p2_change, k=100,p=2, a = ap2 );H_kC <- makeH_k_Wald_RCPP(p2_change, k=100,p=2, a =
max(abs(H_k-H_kC))
```

```
## [1] 3.330669e-16
```

```
head(H_kC)
```

```
##             [,1]
## [1,]  0.49611101
## [2,] -0.10012090
## [3,] -0.08039155
## [4,]  0.04974965
## [5,]  0.05748852
## [6,] -0.01655661
```

Lower and upper summands of difference vector

$$\boldsymbol{A}_{\tilde{\boldsymbol{a}},k}$$

(3.14)

```
H_l <- makeH_l_u(p2_change, p=2, l=11, u=100, a = ap2); H_u <- makeH_l_u(p2_change, p=2, l=101, u=190, a
H_l_C <- makeH_l_u_RCPP(p2_change, p=2, l=11, u=100, a = ap2C); H_u_C <- makeH_l_u_RCPP(p2_change, p=2,
max(abs(H_l-H_l_C))
```

```
## [1] 7.771561e-16
```

```
max(abs(H_u-H_u_C))
```

```
## [1] 6.661338e-16
```

```
head(H_l_C[,1:5])
```

```
##             [,1]        [,2]       [,3]        [,4]         [,5]
## [1,]  0.97821387 -0.62375509 -1.6731229 -0.87534993 -0.234654084
## [2,]  0.07572064  0.31516041  0.2474076 -0.66895118 -0.156345505
## [3,] -0.19136609  0.13404466  0.1356538 -0.00856508  0.003368786
## [4,] -0.13371879  0.44546815  0.2726064 -0.10866632 -0.032632261
## [5,]  0.46542413 -0.46345453 -0.6368778 -0.14112599  0.044700466
## [6,]  0.17818999  0.06603104 -0.1040423  0.08735995  0.048814397
```

Outer expectation matrix

estimator

$$\widehat{\boldsymbol{V}}$$

(3.41)

```
V <- get_V_nk(p2_change, p=2, l=10, u=99);V_C <- get_V_nk_RCPP(p2_change, p=2, l=10, u=99)
max(abs(V-V_C))
```

```
## [1] 2.775558e-17
```

```
str(V_C)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:605] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:56] 0 11 22 33 44 55 66 77 88 99 ...
##   ..@ Dim     : int [1:2] 55 55
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:605] 1.0112 -0.0224 0.0071 -0.0159 -0.0121 ...
##   ..@ factors : list()
```

LOCAL1

Channel variance estimator

$$\hat{\sigma}^2_{n,k}(i)$$

(3.60)

```
sigi <- getsigma_i_kLOCAL1(x = p2_change, i=1, k = 100, G= 90, p =2, ai, get_a_lu_i(p2_change, i=1,p=2,
sigi_C <- getsigma_i_kLOCAL1_RCPP(x = p2_change, i=1, k = 100, G= 90, p =2, aiC, get_a_lu_i_RCPP(p2_cha
sigi
```

```
## [1] 0.05716121
```

```
sigi_C
```

```
## [1] 0.05716121
```

All channels

```
sigd <- getsigma_d_kLOCAL1(x = p2_change, k = 100, G= 90, p =2, ap2, make_a_lu_RCPP(p2_change,p=2, 101,
sigd_C <- getsigma_d_kLOCAL1_RCPP(x = p2_change, k = 100, G= 90, p =2, ap2C, make_a_lu_RCPP(p2_change,p
max(abs(sigd-sigd_C))
```

```
## [1] 2.775558e-17
```

Sigma estimators

$$\widehat{\mathbf{\Sigma}}_{n,k}$$

Diagonal-H (3.56)

```
DH <- get_DiagH_Wald(p2_change, G=90, p=2, H_l, H_u)
DH_C <- get_DiagH_Wald_RCPP(p2_change, G=90, p=2, H_l_C, H_u_C)
max(abs(DH-DH_C))
```

```
## [1] 2.664535e-14
```

```
str(DH_C)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:605] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:56] 0 11 22 33 44 55 66 77 88 99 ...
##   ..@ Dim     : int [1:2] 55 55
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:605] 2.0809 -0.2184 -0.2867 -0.0367 -0.1247 ...
##   ..@ factors : list()
```

$$\widehat{\mathbf{\Sigma}}_{n,k}$$

Full-H (3.57)

```
FH <- get_FullH_Wald(p2_change, G=90, H_l, H_u)
FH_C <- get_FullH_Wald_RCPP(p2_change, G=90, H_l_C, H_u_C)
max(abs(FH-FH_C))
```

```
## [1] 1.49214e-13
```

Note this currently leads to overflow errors for larger `G` values; this needs amending.

$$\widehat{\mathbf{\Sigma}}_{n,k}$$

Diagonal-C (3.58)

```
DC <- get_DiagC_Wald_RCPP(p2_change, p=2, sigma_d = sigd, k=100, G=90)
DC_C <- get_DiagC_Wald_RCPP(p2_change, p=2, sigma_d = sigd_C, k=100, G=90)
max(abs(DC-DC_C))
```

```
## [1] 4.440892e-16
```

```
str(DC_C)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:605] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:56] 0 11 22 33 44 55 66 77 88 99 ...
##   ..@ Dim     : int [1:2] 55 55
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:605] 2.9536 -0.0404 0.0678 -0.0195 -0.0193 ...
##   ..@ factors : list()
```

W

$$W_{k,n}(G)$$

(3.17)

```
get_Wkn(p2_change, p=2, k=100, G=90, estim = "DiagC")
```

```
## [1] 4.961614
```

```
get_Wkn_RCPP(p2_change, p=2, k=100, G=90, estim = "DiagC")
```

```
## [1] 4.961614
```

```
get_Wkn(p2_change, p=2, k=100, G=90, estim = "DiagH")
```

```
## [1] 3.74541
```

```
get_Wkn_RCPP(p2_change, p=2, k=100, G=90, estim = "DiagH")
```

```
## [1] 3.74541
```

```
get_Wkn(p2_change, p=2, k=100, G=90, estim = "FullH")
```

```
## [1] 4.202006
```

```
get_Wkn_RCPP(p2_change, p=2, k=100, G=90, estim = "FullH")
```

```
## [1] 4.202006
```

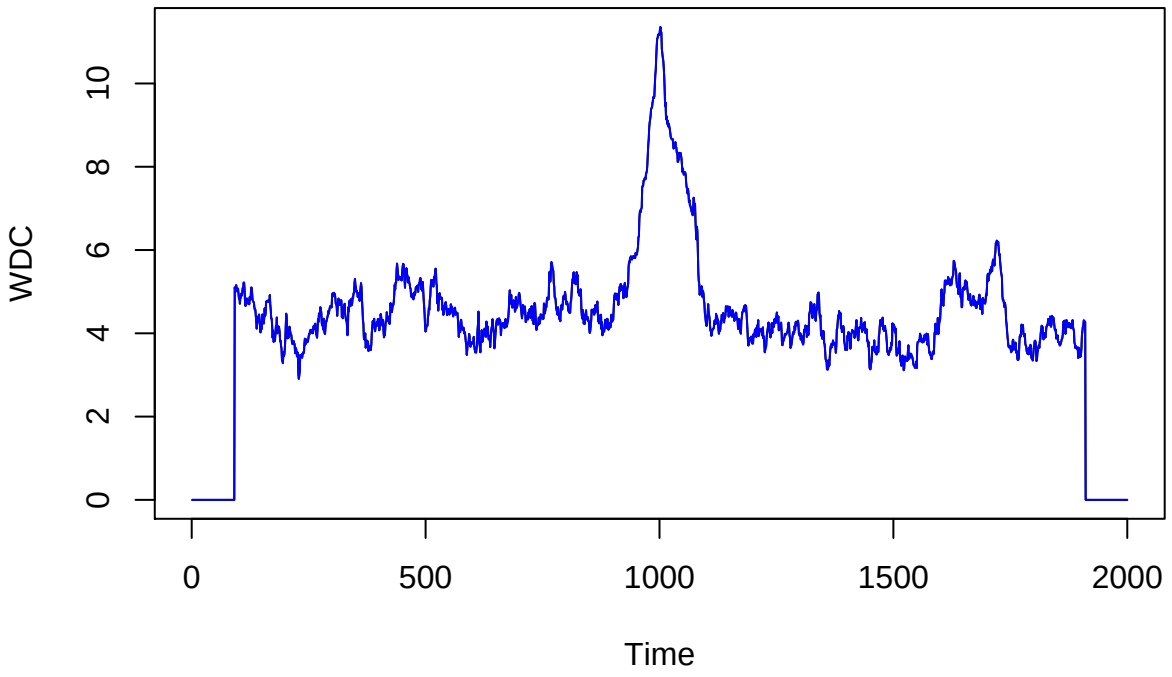
$$W_n(G)$$

(3.17)

```
WDC <- get_W(p2_change, p=2, G=90, estim = "DiagC")
WDC_C <- get_W_RCPP(p2_change, p=2, G=90, estim = "DiagC")
max(abs(WDC-WDC_C))
```

```
## [1] 1.865175e-14
```

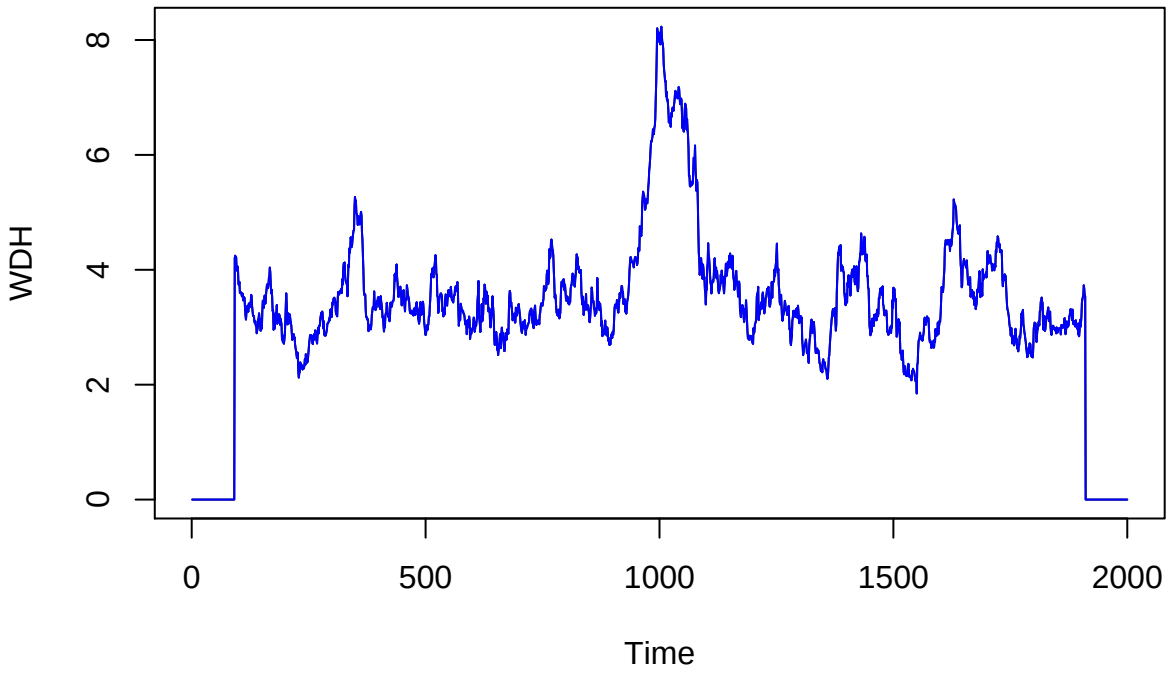
```
plot.ts(WDC); lines(WDC_C, col = "blue")
```




```
WDH <- get_W(p2_change, p=2, G=90, estim = "DiagH")
WDH_C <- get_W_RCPP(p2_change, p=2, G=90, estim = "DiagH")
max(abs(WDH-WDH_C))
```

```
## [1] 4.52971e-14
```

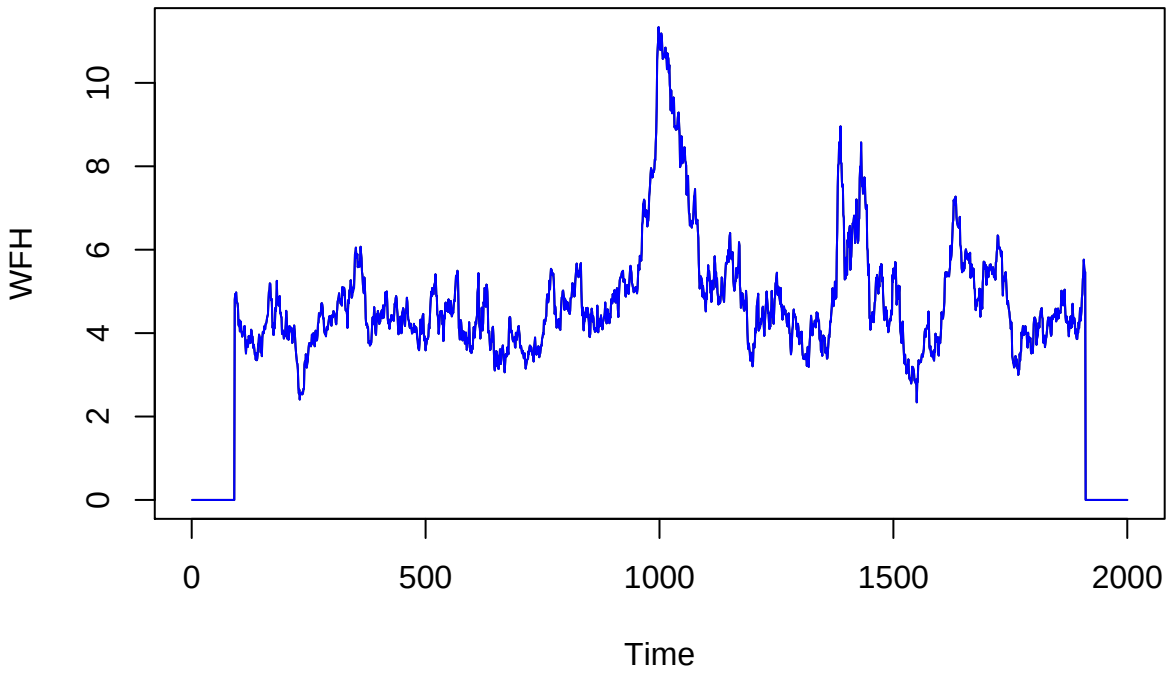
```
plot.ts(WDH); lines(WDH_C, col = "blue")
```




```
WFH <- get_W(p2_change, p=2, G=90, estim = "FullH")
WFH_C <- get_W_RCPP(p2_change, p=2, G=90, estim = "FullH")
max(abs(WFH-WFH_C))
```

```
## [1] 6.483702e-14
```

```
plot.ts(WFH); lines(WFH_C, col = "blue")
```



Here, the results are identical.

RcppParallel Simulations

While the computation of each individual test could be easily parallelised over the evaluation of $W_{k,n}$ for each time k, the procedure is already way faster than we should require it to be. We can, however, make some gains in monte carlo experiments. When simulating multiple replicates from a given process, as in chapter 5 of the report, each replicate can be sent to a different worker in parallel, allowing a greater number of simulations to be used. This should give us better approximations of properties such as asymptotic power, size, and estimation precision.

Runtime

The function `var_simulate_RCPP` generates, for each replicate, data from the stochastic process determined by the autoregression matrices `pars`, and returns a vector of zeros and ones corresponding to non-rejected nulls and rejected nulls.

```
pars <- list(a1,a2,a2,a1,a1+a2,a1%*%a2)
var_simulate_RCPP(pars,reps=10, ncores = 1)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

This is parallelised as follows

```
// [[Rcpp::export(var_simulate_RCPP)]]
NumericVector var_sim(List pars, int reps =100, int p=2, int G=200, double alpha =0.05, String estim =
  vec cp={500,1000,1500};
  NumericVector out(reps) ;
  RcppParallel::RVector<double> wo(out);
  //RcppParallel::RVector<double> wx(x);

  #if defined(_OPENMP)
  #pragma omp parallel for num_threads(ncores)
  #endif
   for(int repl = 0; repl < reps; repl++ ){
   List p1 = List::create(pars[0], pars[1]);List p2 = List::create(pars[2], pars[3]);List p3 = List::cr
   mat r1 = sim_data(p1, cp(0), 5);mat r2 = sim_data(p2, cp(1)-cp(0), 5);mat r3 = sim_data(p3, cp(2)-cp
   mat r = join_cols(r1,r2,r3); //full data
   List t = test_Wald(r, p, G, alpha,estim);
   wo[repl] = t[0];
   };
  //Output----------------------------------
  // List out = List::create(Named("Reject") = Reject, _["Wn"] = Wn, _["ChangePoints"] = cp, _["D_n"]=D
  return out ;
}
```

We compare how this performs for different numbers of cores

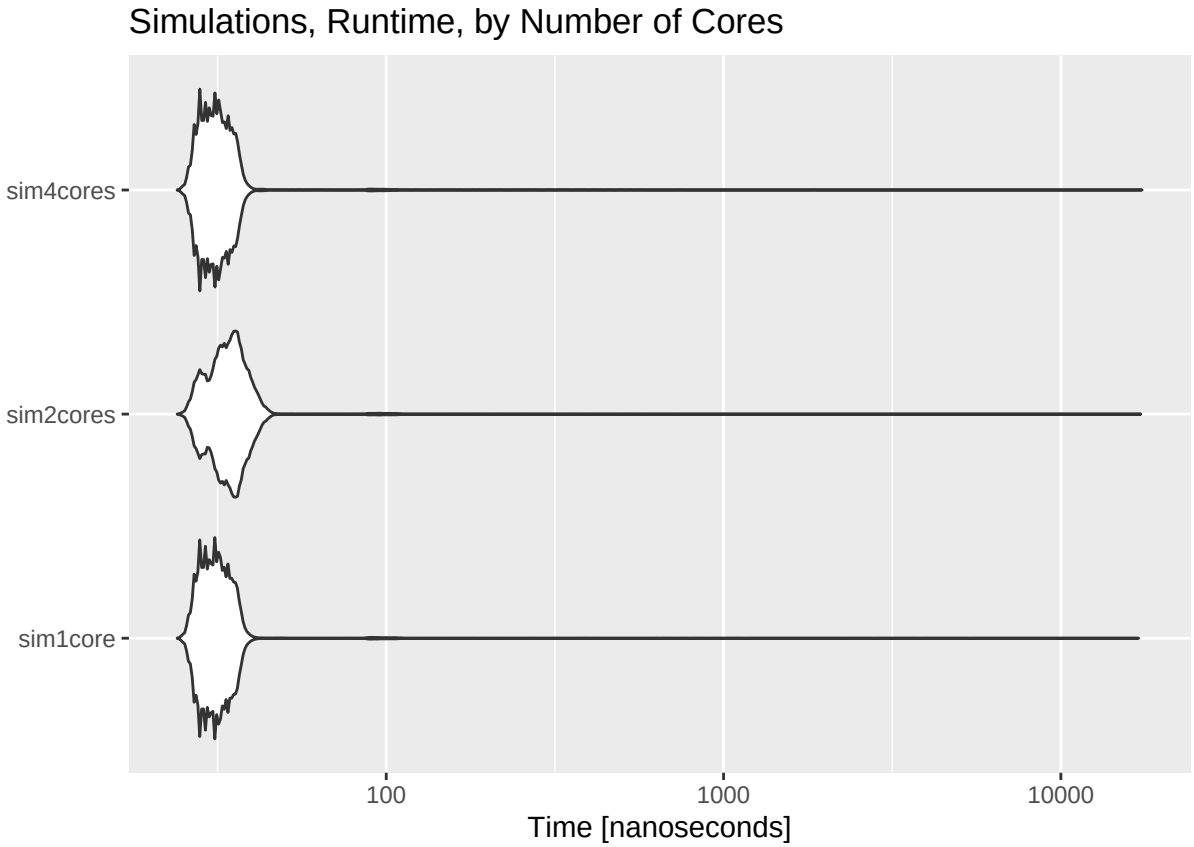
```
sim1core <- function()var_simulate_RCPP(pars,reps=10e10, ncores = 1)
sim2cores <- function()var_simulate_RCPP(pars,reps=10e10, ncores = 2)
sim4cores <- function()var_simulate_RCPP(pars,reps=10e10, ncores = 4)
simmb <- microbenchmark(sim1core, sim2cores, sim4cores, times = 100000)
print(simmb)
```

```
## Unit: nanoseconds
##       expr min lq     mean median uq   max neval
##   sim1core  24 29 32.70881     31 34 17001 1e+05
##  sim2cores  24 31 35.60684     34 37 17244 1e+05
```

```
##  sim4cores  24 29 33.13195     31 34 17420 1e+05
```

```
autoplot(simmb) + ggtitle("Simulations, Runtime, by Number of Cores")
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Running these simulations in parallel does not seem to speed them up, and indeed spreading the job across workers may actually slow the computation down; it is possible this is due to the relatively low dimensionality of the problem. For comparison, simulations with $N = 100$ replicates of the base-R implementation took hours, and necessitated use of HPC capability.