manuscript No.
(will be inserted by the editor)

Supplementary material for the paper "Multiple Monte Carlo Testing, with Applications in Spatial Point Processes", with code examples

Tomáš Mrkvička · Mari Myllymäki · Ute Hahn

Abstract This document contains R codes for the examples in [Mrkvička, T., Myllymäki, M., Hahn, U.: Multiple Monte Carlo Testing, with Applications in Spatial Point processes. Statistics and Computing (2016)], using R libraries spptest and spatstat.

R codes

All the examples below require the R (R Core Team, 2014) libraries `spptest` (`https://github.com/myllym/spptest`) and `spatstat` (Baddeley et al., 2015). The versions 0.4 of `spptest` and 1.41-1 of `spatstat` were used for this paper.

The code for the particles example in Section 6

First define the number of simulations and r-distances:

```
> nsim <- 9999 # the number of simulations for the tests
> n <- 500 # the number of r-values
> rmin <- 6.1; rmax <- 125; rstep <- (rmax-rmin)/n
> rminJ <- 6.1; rmaxJ <- 30; rstepJ <- (rmaxJ-rminJ)/n
> r <- seq(0, rmax, by=rstep)    # r-distances for Lest
> rJ <- seq(0, rmaxJ, by=rstepJ) # r-distances for Fest, Gest, Jest
```

Tomáš Mrkvička
Department of Applied Mathematics and Informatics, Faculty of Economics, University of South Bohemia, Studentská 13, 370 01, České Budějovice, Czech Republic
E-mail: mrkvicka.toma@gmail.com

Mari Myllymäki
Natural Resources Institute Finland (Luke), P.O. Box 18, FI-01301 Vantaa, Finland
E-mail: mari.myllymaki@luke.fi

Ute Hahn
Centre for Stochastic Geometry and Advanced Bioimaging, Department of Mathematics, University of Aarhus, 8000 Aarhus C, Denmark
E-mail: ute@imf.au.dk

Assume `X` contains the point pattern of particles as an `ppp` object (see the help page in spatstat),

```
> X
planar point pattern: 337 points
window: rectangle = [0, 512] x [0, 512] units
```

The minimum interpoint distance between the particles can be calculated utilizing `spatstat`'s function `minnndist` and the hard core distance obtained:

```
> minnndist(X)
[1] 5.884157
> HD <- minnndist(X) * X$n / (X$n+1)
> HD
[1] 5.866748
```

Then the hard core process can be fitted to the data using spatstat's function `ppm`:

```
> model <- ppm(X, interaction=Hardcore(HD))
```

In order to deal with a simple hypothesis, we want to fix the number of points in the pattern. The function `rmh` in `spatstat` allows to generate simulations of the hard core process with the fixed number of simulations:

```
> # Initialization
> simulations <- NULL
> # Specify the model for 'rmh'
> model2 <- list(cif="hardcore",
+                par=list(beta=exp(model$coef[1]), hc=HD),
+                w=X$window)
> # Specify an initial pattern for 'rmh'
> init_x <- runifpoint(X$n, win=X$window)
> # Make nsim simulations
> for(s in 1:nsim)
>   simulations[[s]] <- rmh(model=model2, start=list(x.start=init_x),
+                           control=list(p=1, nrep=1e5, nverb=5000))
```

In the `rmh` call, `p = 1` sets that only moves are proposed, which consequently keeps the number of points fixed. The object `simulations` must then be passed to `spatstat`'s function `envelope`, which can be utilized in calculation of the test functions. In the `envelope` calls, `savefuns` must be set to `TRUE` so that the test functions are saved. The arguments `fun`, `correction`, `transform` and `r` specify the test functions and distances to be used.

```
> # L(r)-r function
> envHC_L <- envelope(X, nsim=nsim,
+                     simulate=simulations, # The generated simulations
+                     fun="Lest", correction="translate",
+                     transform = expression(.-r), # L(r)-r instead of L(r)
+                     r=r, # Specify the distance vector
+                     savefuns=TRUE)
> # F(r) function
> envHC_F <- envelope(X, nsim=nsim,
```

```
+                     simulate=simulations,
+                     fun="Fest", correction="Kaplan", r=rJ,
+                     savefuns=TRUE)
> # G(r) function
> envHC_G <- envelope(X, nsim=nsim,
+                     simulate=simulations,
+                     fun="Gest", correction="km", r=rJ,
+                     savefuns=TRUE)
> # J(r) function
> envHC_J <- envelope(X, nsim=nsim,
+                     simulate=simulations,
+                     fun="Jest", correction="none", r=rJ,
+                     savefuns=TRUE)
```

Thereafter, the curves are cropped to the desired r-interval I and combined together

```
> cset_L <- crop_curves(envHC_L, r_min=rmin, r_max=rmax)
> cset_F <- crop_curves(envHC_F, r_min=rminJ, r_max=rmaxJ)
> cset_G <- crop_curves(envHC_G, r_min=rminJ, r_max=rmaxJ)
> cset_J <- crop_curves(envHC_J, r_min=rminJ, r_max=rmaxJ)
> # Combine the curve sets
> cset_LFGJ <- combine_curve_sets(list(cset_L, cset_F, cset_G, cset_J))
```

Then the rank envelope test itself is done and the result (in the style of Figure 4) plotted as follows:

```
> res <- rank_envelope(cset_LFGJ)
> plot(res, use_ggplot2=TRUE,
+      labels=c("L(r)-r", "F(r)", "G(r)", "J(r)"),
+      separate_yaxes=TRUE)
```

The combined scaled MAD envelope tests can be performed by means of the function `combined_scaled_MAD_envelope`

```
> res <- combined_scaled_MAD_envelope(curve_sets=list(cset_L, cset_F,
+                                                     cset_G, cset_J),
+                                     test = "qdir")
```

where the argument `test` specifies either the directional quantile or the studentized MAD test. The result can be plotted

```
> plot(res, use_ggplot2=TRUE,
+      labels=c("L(r)-r", "F(r)", "G(r)", "J(r)"),
+      separate_yaxes=TRUE)
```

In order to test the Strauss hard-core process (a composite hypothesis) fixing the number of points in the pattern, we first define a function for fitting this null model and a function for generating realizations from the fitted null model. These two functions are

```
> # A function for fitting the Strauss hard-core process
> StraussHard.fit <- function(X) {
+   fittedmodel <- ppm(X, interaction=StraussHard(r=15, hc=HD),
```

```
+                      method="logi")
+   list(X=X, model=fittedmodel)
+ }
> # A function for simulating from a given Strauss hard-core process
> StraussHard.simfun <- function(par) {
+   # par should be a list containing 'X' () and
+   # 'model' (fitted model object)
+   # Specify the model for 'rmh'
+   mod <- list(cif="straush",
+               par=list(beta=exp(par$model$coef[1]),
+                        gamma=exp(par$model$coef[2]),
+                        r=par$model$interaction$par$r,
+                        hc=par$model$interaction$par$hc),
+               w=par$X$window)
+   # Generate an initial pattern
+   init_x <- runifpoint(par$X$n, win=par$X$window)
+   # Simulate (p=1 sets that only moves are proposed -> the number of points is fixed)
+   rmh(model=mod, start=list(x.start=init_x),
+       control=list(p=1, nrep=1e5, nverb=5000))
+ }
```

The `.simfun` function needs to accept as the argument the object that is returned by the `.fit` function. Then we define a list of lists of arguments needed to be passed to the function `envelope` in order to calculate the test functions $L(r) - r$, $F(r)$, $G(r)$ and $J(r)$ with our choices of edge corrections and distances.

```
> testfuns <- list(L = list(fun="Lest", correction="translate",
+                           transform = expression(.-r), r=r),
+                  F = list(fun="Fest", correction="Kaplan", r=rJ),
+                  G = list(fun="Gest", correction="km", r=rJ),
+                  J = list(fun="Jest", correction="none", r=rJ))
```

These objects and additional arguments are passed to `dg.combined_global_envelope` to perform the adjusted combined global directional quantile MAD envelope test with $s = 499$ simulations (and additional 499 simulations for each of these simulations) as follows:

```
> adjenv <- dg.combined_global_envelope(X = X, nsim=499, nsimsub = 499,
+                                 simfun=StraussHard.simfun,
+                                 fitfun=StraussHard.fit,
+                                 testfuns=testfuns,
+                                 test = "qdir",
+                                 r_min = c(rmin, rminJ, rminJ, rminJ),
+                                 r_max = c(rmax, rmaxJ, rmaxJ, rmaxJ),
+                                 mc.cores=10L)
```

Above `r_min` and `r_min` give the minimum and maximum distances for each of the test functions. The argument `mc.cores` can be used to define how many cores to use (default 1) for simulations needed to be done in order to calculate the adjusted level of the test. The test result (in the style of Figure 6) can be plotted

```
> plot(adjenv, plot_type="MAD",
+      use_ggplot2=TRUE, separate_yaxes=TRUE,
+      labels=c("L(r)", "F(r)", "G(r)", "J(r)"))
```

The code for the ENF example in Section 7

Assume the objects `X1`, `X2`, `X3` and `X4` are R objects of type `ppp` containing the entry (or end) point patterns of ENFs. Define the number of simulations to be performed and the distances on which the L-functions are to be evaluated:

```
> s <- 9999 # The number of simulations
> r <- seq(0, 80, length=513) # The distances for each pattern
```

For each point pattern, make then s simulations of CSR and calculate the test functions for the data and each simulated pattern. This can be done using spatstat's function `envelope` as follows (here code example for the first point pattern `X1`):

```
> env1 <- envelope(X1, nsim=s,
+   simulate = expression(runifpoint(X1$n, win=X1$window)),
+   fun=Lest, transform=expression(.-r), correction="translate",
+   r=r, savefuns = TRUE)
```

In the argument `simulate` we specify the simulation of CSR, i.e. we simulate the binomial process with the number of points equal to the number of points in the point pattern.

Assume now that four `envelope` objects `env1`, ..., `env4` have been created. They can be combined together to one `curve_set` object

```
> curve_set_full <- combine_curve_sets(list(env1, env2, env3, env4))
```

and the rank envelope test performed simply by the command

```
> res <- rank_envelope(curve_set_full)
```

A plot similar to those in Figures 8 and 9 can be produced as follows:

```
> plot(res, use_ggplot2=TRUE,
+   ylab=expression(italic(hat(L)(r)-r)), # label for y-axis
+   separate_yaxes = TRUE,
+   labels=paste("Subject", c(171, 224, 230, 256), sep=" ")) # labels
```

The code for the rain forest example in Section 8

Assume `X` is a multitype marked point pattern of class `ppp`

```
> X
marked planar point pattern: 2402 points
Multitype, with levels = DES2PA, FARAOC, HYBAPR
window: rectangle = [650, 750] x [250, 350] units
```

First define the number of simulations, the distances (zero needs to be included for spatstat's `Lcross` function and removed later) and take the mark levels into a vector:

```
> nsim <- 2499 # number of simulations
> r <- c(0, seq(1, 25, length=513)) # distances
> mark_levels <- levels(X$marks)
> mark_levels
[1] "DES2PA" "FARAOC" "HYBAPR"
```

Then for each pair of mark levels, i.e. for each pair of sub-point patterns, perform `nsim` simulations under the null hypothesis (random shifts defined below in the argument `simulate`), calculate the L_{ij}-functions and save results. This can be utilizing `envelope` from spatstat, remembering to set `savefuns=TRUE`:

```
> for(i in 1:(length(mark_levels)-1)) {
+  for(j in (i+1):length(mark_levels)) {
+     assign(paste("env", i, j, sep=""),
+         envelope(X, nsim = nsim,
+                  fun=Lcross, correction="translate",
+                  transform = expression(.-r),
+                  i=mark_levels[i], j=mark_levels[j],
+                  simulate = expression(rshift(X,
+                                  which=mark_levels[j],
+                                  edge="torus")),
+                  savefuns = TRUE, r=r))
+  }
+ }
```

The saved functions can be cropped to $I = [1, 25]$ using

```
> env12 <- crop_curves(env12, r_min=1, r_max=25)
```

for each of the three `envelope` objects `env12`, `env13` and `env23`, which then can be combined together

```
> curve_set_full <- combine_curve_sets(list(env12, env13, env23))
```

And, the created object can be passed to `rank_envelope`

```
> res <- rank_envelope(curve_set_full, lexo=TRUE)
> res
Rank envelope test
 p-value of the test: 0.8792 (ties method: lexical)
 p-interval         : (0.8792, 0.88)
```

and the result plotted

```
> plot(res, use_ggplot2=TRUE,
+ ylab=expression(paste("centred ", italic(L[ij](r)))),
+ separate_yaxes=TRUE, max_ncols_of_plots=3,
+ labels=c(paste(mark_levels[1], " - ", mark_levels[2], sep=""),
+           paste(mark_levels[1], " - ", mark_levels[3], sep=""),
+           paste(mark_levels[2], " - ", mark_levels[3], sep="")))
```

where the named arguments specify a figure similar to Figure 12.

References

Baddeley, A., E. Rubak, and R. Turner (2015). *Spatial Point Patterns: Methodology and Applications with R.* London: Chapman and Hall/CRC Press.

R Core Team (2014). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing.