# GET: Point patterns

**Mari Myllymäki**
Natural Resources Institute Finland (Luke)

**Abstract**

This vignette gives examples of the use of global envelopes for point pattern analysis, as implemented in the R package **GET**. When citing the vignette and package please cite Myllymäki and Mrkvička (2023) and references given by typing `citation("GET")` in R.

*Keywords*: global envelope test, goodness-of-fit, Monte Carlo test, R, spatial point pattern.

## 1. Introduction

This vignette gives examples of the use of global envelopes for the analysis of spatial point patterns. The examples utilize the R (R Core Team 2023) package **spatstat** (Baddeley, Rubak, and Turner 2015) in addition to the **GET** package (Myllymäki and Mrkvička 2023). The envelope plots are produced by the use of the **ggplot2** package (Wickham 2016), where we utilize the theme `theme_bw` for this document.

```
R> library("GET")
R> library("spatstat.model")
R> library("ggplot2")
R> theme_set(theme_bw(base_size = 9))
```

## 2. General workflow of the tests

### 2.1. Utilizing the spatstat package

In general, it is useful in the point pattern analysis utilize the **spatstat** package. The workflow utilizing spatstat with the **GET** package is typically the following: Say we have a point pattern, for which we would like to test a hypothesis, as a `ppp` object of spatstat. E.g.

```
R> X <- spruces
R> X

Marked planar point pattern: 134 points
marks are numeric, of storage type  'double'
window: rectangle = [0, 56] x [0, 38] metres
```

1. To test a simple hypothesis, e.g., complete spatial randomness (CSR):

- Use the function envelope of **spatstat** to create `nsim` simulations under CSR and to calculate the functions you want. Important: use the option `savefuns=TRUE` and specify the number of simulations `nsim`. See the help documentation in the **spatstat** package for possible test functions (if the argument `fun` is not given, the function `Kest()` is used, i.e. an estimator of the $K$-function).

  Making 999 simulations of CSR and estimating $K$-function for each of them and data (the argument `simulate` specifies how to perform simulations under CSR):

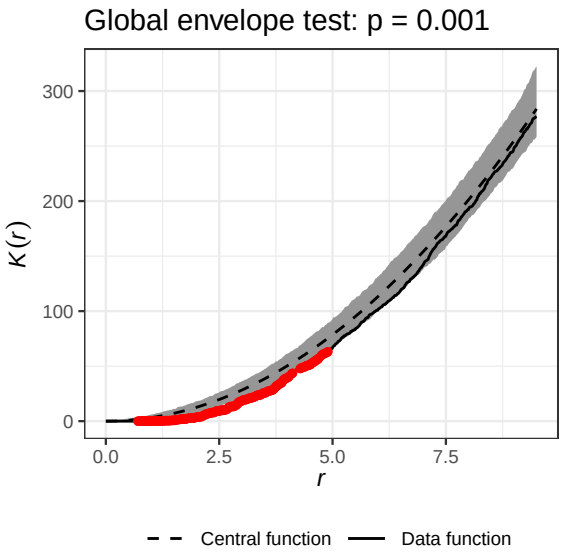
```
R> env <- envelope(X, nsim=1999, savefuns=TRUE,
+                  simulate=expression(runifpoint(ex=X)),
+                  verbose=FALSE)
```

- Perform the test

```
R> res <- global_envelope_test(env)
```

- Plot the result

```
R> plot(res)
```




2. To test a goodness-of-fit of a parametric model (composite hypothesis case):

- Fit the model to your data by means of the function `ppm()` or `kppm()` of **spatstat**. See the help documentation for possible models.

- Use the function `GET.composite()` to create `nsim` simulations from the fitted model, to calculate the functions you want, and to make an adjusted global envelope test. See the example below.

- Plot the result.

More detailed examples are given below.

### 2.2. The workflow when using your own programs for simulations

- (Fit the model and) Create $s$ simulations from the (fitted) null model.
- Calculate the functions $T_1(r), T_2(r), \ldots, T_{s+1}(r)$.
- Use `create_curve_set()` to create a `curve_set` object from the functions $T_i(r)$, $i = 1, \ldots, s+1$.
- Perform the test and plot the result

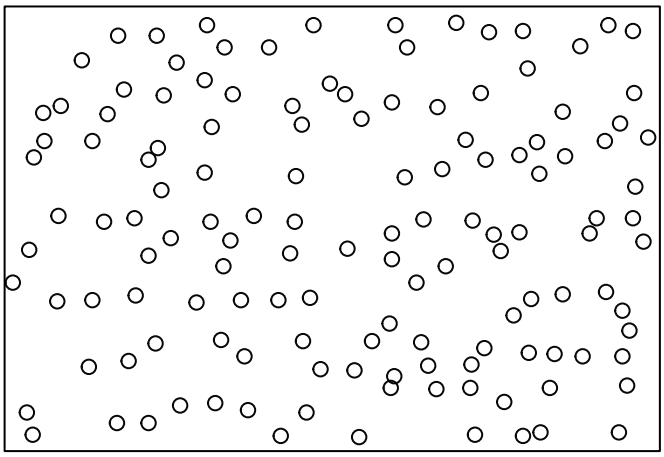
See example in the help file of the `global_envelope_test()` function.

## 3. Testing simple hypotheses

### 3.1. Testing complete spatial randomness (CSR)

Let us illustrate the CSR for the spruces data set from the R library **spatstat**.

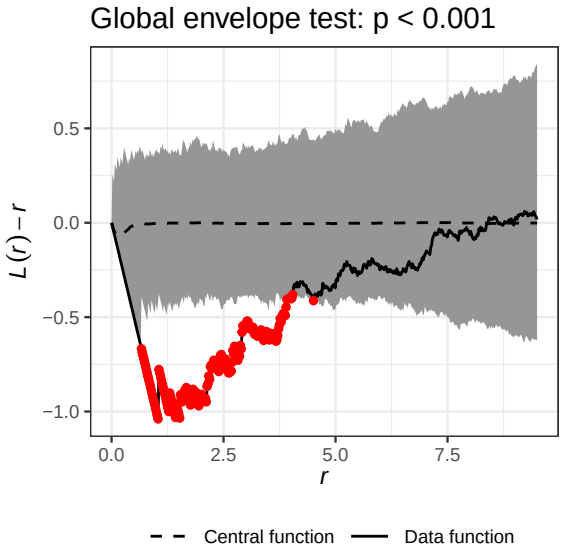
```
R> X <- unmark(spruces)
R> par(mfrow = c(1,1), mgp = c(0, 0, 0), mar = c(0, 0, 0, 0))
R> plot(X, main = "")
```



Below the function `envelope()` of the **spatstat** package is used to generate point patterns under CSR (specified in the argument `simulate`) and to calculate the centred $L$-functions (specified below by the arguments `fun`, `correction` and `transform`), which are used here as the test functions.

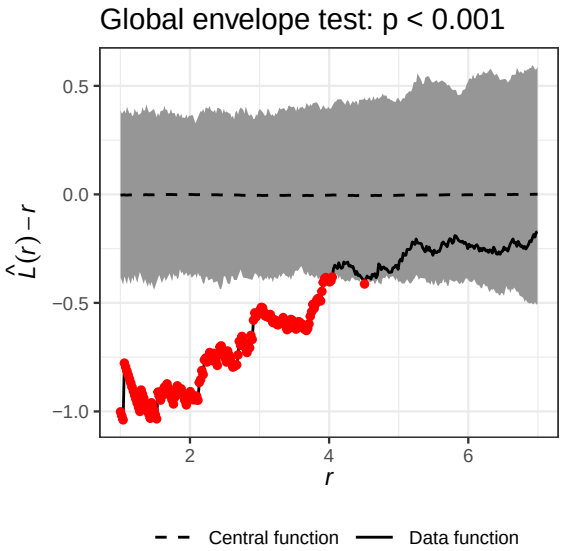
```
R> nsim <- 1999 # Number of simulations
R> env <- envelope(X, fun = "Lest", nsim = nsim,
+    savefuns = TRUE, # save the functions
```

```
+     correction = "translate", # edge correction for L
+     transform = expression(.-r), # centering
+     simulate = expression(runifpoint(ex = X)), # Simulate CSR
+     verbose = FALSE)
R> res <- global_envelope_test(env, type = "erl")
R> plot(res)
```




It is possible to cut the functions to an interval of distances $[r_{\min}, r_{\max}]$ (at the same time creating a `curve_set` from `env`) and perform the test on the functions on this interval only:

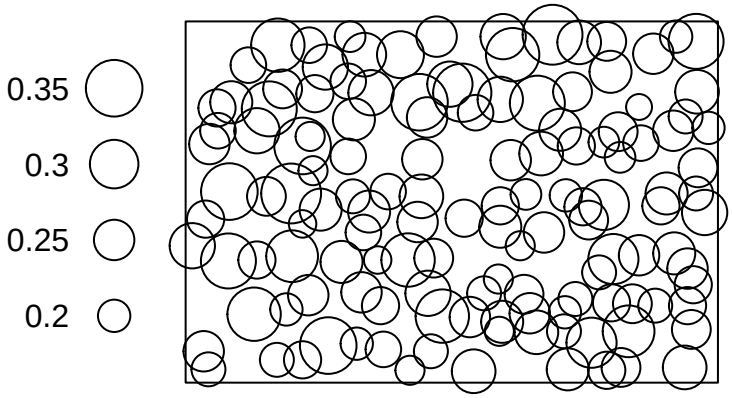
```
R> cset <- crop_curves(env, r_min = 1, r_max = 7)
R> # Do the rank envelope test (erl)
R> res <- global_envelope_test(cset, type = "erl")
R> plot(res) + ylab(expression(italic(hat(L)(r)-r)))
```



### 3.2. Testing random labeling of marks

Let now the studied marked point pattern be the spruces data with marks:

```
R> mpp <- spruces
R> par(mfrow=c(1,1), mgp=c(0, 0, 0), mar=c(0, 0, 0, 0))
R> plot(mpp, main = "")
```

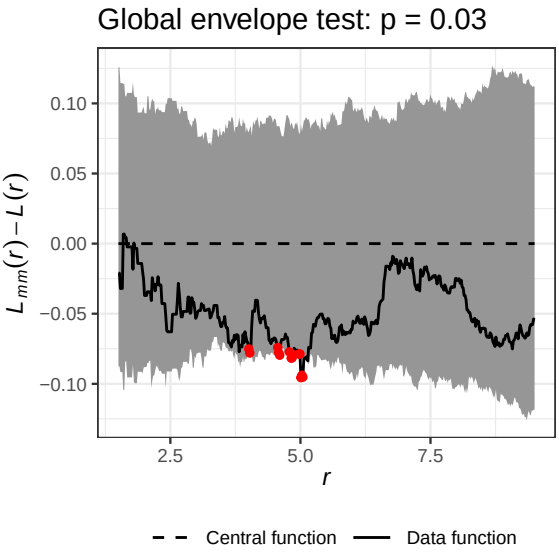



As in the CSR test, to perform the test of random labelling hypothesis, first the `envelope()` function of **spatstat** can be used to generate simulations and calculate the test function $T(r)$ for the data pattern (mpp) and each simulation. Below the estimator of the mark-weighted $L$-function, $L_{mm}(r)$, with translational edge correction is used as the test function. The argument `simulate` specifies the simulations under the random labeling, i.e., simple permutation of the marks.

```
R> nsim <- 1999 # Number of simulations
R> env <- envelope(mpp, fun = Kmark, nsim = nsim, f = function(m1, m2) { m1*m2 },
+    correction = "translate", returnL = TRUE,
+    simulate = expression(rlabel(mpp, permute = TRUE)), # Permute the marks
+    savefuns = TRUE, # Save the functions
+    verbose = FALSE)
```

Thereafter, the curves can be cropped to the desired $r$-interval and centered by the mean of the simulated functions for better visualization, before making the test.

```
R> # Crop curves to desired r-interval
R> curve_set <- crop_curves(env, r_min = 1.5, r_max = 9.5)
R> # Center the functions, i.e. take \hat{L}_mm(r)-the mean of simulated functions.
R> curve_set <- residual(curve_set)
R> # The global envelope test
R> res <- global_envelope_test(curve_set)
R> plot(res) + ylab(expression(italic(L[mm](r)-L(r))))
```




### 3.3. A combined global envelope test

Sometimes it may be desired to base the test on several test functions (Mrkvička, Myllymäki, and Hahn 2017). Below it is illustrated how the CSR can be tested simultaneously by means of $L$, $F$, $G$ and $J$ functions for the saplings data set available in the **GET** library. First some setup:

```
R> data(saplings)
R> X <- as.ppp(saplings, W = square(75))
R> nsim <- 499 # Number of simulations
R> # Specify distances for different test functions
R> n <- 500 # the number of r-values
R> rmin <- 0; rmax <- 20; rstep <- (rmax-rmin)/n
R> rminJ <- 0; rmaxJ <- 8; rstepJ <- (rmaxJ-rminJ)/n
R> r <- seq(0, rmax, by = rstep)    # r-distances for Lest
R> rJ <- seq(0, rmaxJ, by = rstepJ) # r-distances for Fest, Gest, Jest
```

Then perform simulations of CSR and calculate the $L$-functions saving the simulated patterns and functions:

```
R> env_L <- envelope(X, nsim = nsim,
+    simulate = expression(runifpoint(ex = X)),
+    fun = "Lest", correction = "translate",
+    transform = expression(.-r), # Take the L(r)-r function instead of L(r)
+    r = r,                       # Specify the distance vector
+    savefuns = TRUE,             # Save the estimated functions
+    savepatterns = TRUE,         # Save the simulated patterns
+    verbose = FALSE)
R> # The simulations can be obtained from the returned object:
R> simulations <- attr(env_L, "simpatterns")
```

And then the other test functions $F$, $G$, $J$ should be calculated for each simulated pattern:

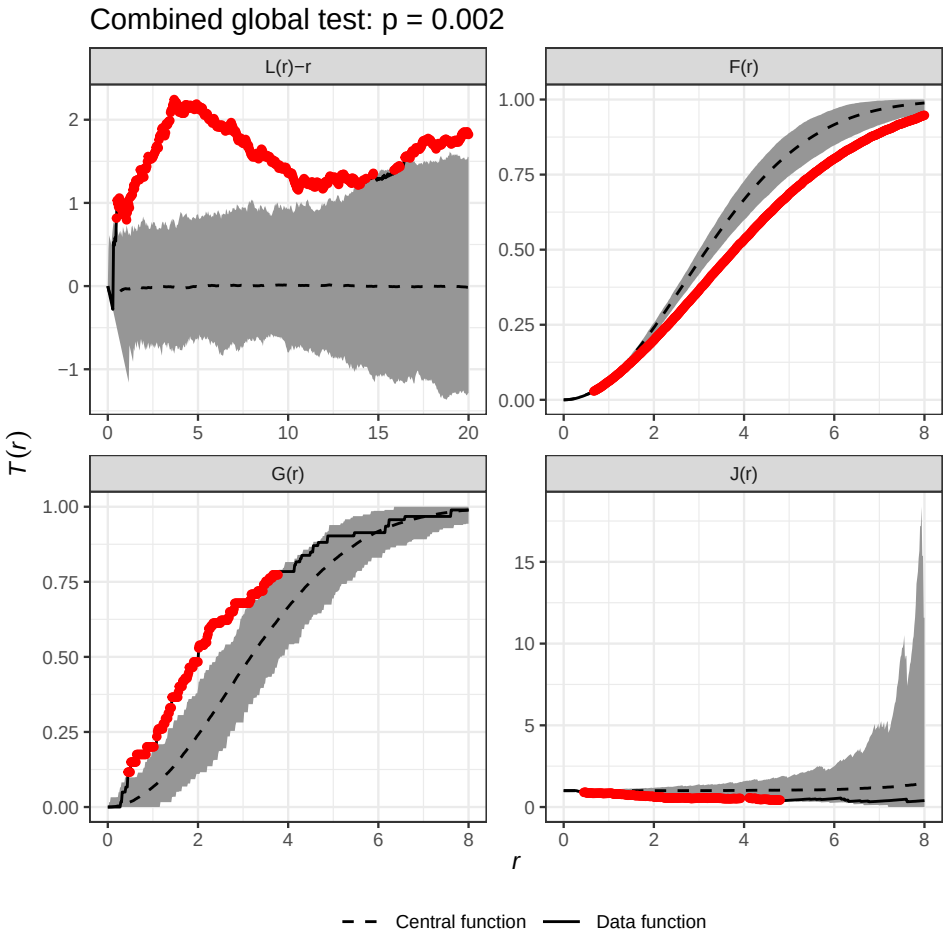
```
R> env_F <- envelope(X, nsim = nsim,
+    simulate = simulations,
+    fun = "Fest", correction = "Kaplan", r = rJ,
+    savefuns = TRUE, verbose = FALSE)
R> env_G <- envelope(X, nsim = nsim,
+    simulate = simulations,
+    fun = "Gest", correction = "km", r = rJ,
+    savefuns = TRUE, verbose = FALSE)
R> env_J <- envelope(X, nsim = nsim,
+    simulate = simulations,
+    fun = "Jest", correction = "none", r = rJ,
+    savefuns = TRUE, verbose = FALSE)
```

All the curves can then be cropped to the desired r-interval $I$, if needed,

```
R> curve_set_L <- crop_curves(env_L, r_min = rmin, r_max = rmax)
R> curve_set_F <- crop_curves(env_F, r_min = rminJ, r_max = rmaxJ)
R> curve_set_G <- crop_curves(env_G, r_min = rminJ, r_max = rmaxJ)
R> curve_set_J <- crop_curves(env_J, r_min = rminJ, r_max = rmaxJ)
```

and finally the combined global envelope calculated and plotted

```
R> res_combined <- global_envelope_test(curve_sets = list(curve_set_L, curve_set_F,
+                                          curve_set_G, curve_set_J))
R> plot(res_combined, labels = c("L(r)-r", "F(r)", "G(r)", "J(r)"))
```




## 4. The problem of NA values in the curve set

With some functions in spatial statistics, missing NA values often appear for some distances $r$, which typically are not of interest, e.g., for the $J$-function with too large distances and for the pair-correlation function at zero. There is currently no automatic way to compute the envelopes with NA values existing in the `curve_set` object: when such distances are present, they should be removed before computation of the envelope. This can be done using the function `crop_curves()`. The function allows 1) to crop the curves to a user-specified interval $[r_{\min}, r_{\max}]$ (arguments `r_min` and `r_max`), or 2) to crop away all $r$-distances with NA (or infinite) values.

Consider the example of adult trees from **GET** (Myllymäki and Mrkvička 2023, Section 3.2) replacing the $L$-function with the $J$-function. The `envelope()` function returns some NAs for the range of $r$-values specified below. Without cropping away these, the function `global_envelope_test()` returns an error. A working example is as follows (instead of `allfinite` one could specify cropping through `r_max`):

```
R> data("adult_trees")
R> X <- as.ppp(adult_trees, W = square(75))
R> env <- envelope(X, nsim = 999, fun = "Jest", correction = "km",
```

```
+    simulate = expression(runifpoint(ex = X)),
+    savefuns = TRUE, verbose = FALSE, r = seq(0, 10, length = 512))
R> cset <- crop_curves(env, allfinite=TRUE)
R> res <- global_envelope_test(cset)
```

## 5. A one-stage goodness-of-fit test (typically conservative!)

It is possible to perform a one-stage goodness-of-fit test for point process models as follows, accepting that the test may be conservative (or liberal). In literature, it has been recommended that the tests may be used if the test function is not closely related to the estimation procedure that was used to fit the model. However, `GET.composite()` can be used for adjusted tests, see the help file of this function and an example below.

```
R> X <- unmark(spruces)
R> # Minimum distance between points in the pattern
R> min(nndist(X))
```

```
[1] 1.044031
```

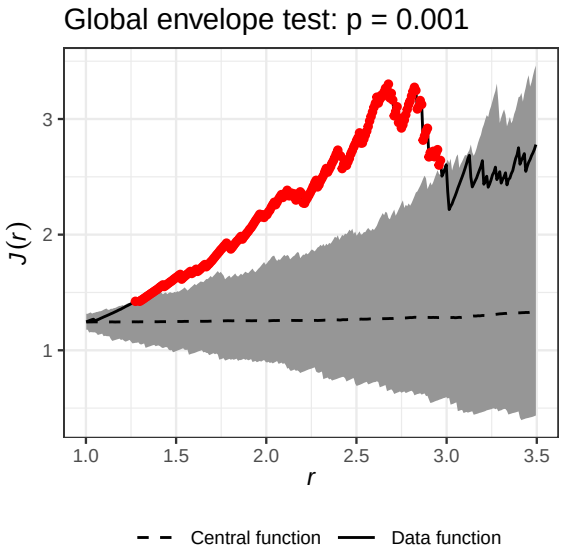
```
R> # Fit a model
R> fittedmodel <- ppm(X, interaction = Hardcore(hc = 1)) # Hardcore process
```

Simulating Gibbs process by `envelope()` is slow, because it uses an MCMC algorithm

```
R> #env <- envelope(fittedmodel, fun = "Jest", nsim = 999, savefuns = TRUE,
R> #                correction = "none", r = seq(0, 4, length = 500))
```

Using direct algorihm can be faster, because the perfect simulation is used here. Therefore, in the following we utilize the function `rHardcore()`:

```
R> simulations <- NULL
R> nsim <- 999 # Number of simulations
R> for(j in 1:nsim) {
+     simulations[[j]] <- rHardcore(beta = exp(fittedmodel$coef[1]),
+                           R = fittedmodel$interaction$par$hc,
+                           W = X$window)
+  }
R> env_HC <- envelope(X, simulate = simulations, fun = "Jest",
+    nsim = length(simulations),
+    savefuns = TRUE, correction = "none",
+    r = seq(0, 4, length = 500),
+    verbose = FALSE)
R> curve_set <- crop_curves(env_HC, r_min = 1, r_max = 3.5)
R> res_HC <- global_envelope_test(curve_set, type = "erl")
R> plot(res_HC) + ylab(expression(italic(J(r))))
```

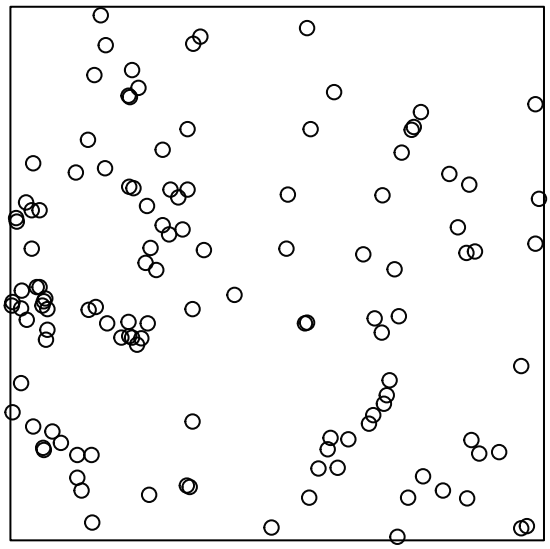



Note: Conditioning the Gibbs hard-core model on the number of points, the only parameter in the hard-core model is the hard-core distance. It is possible to fix the hard-core distance, e.g. to the minimum distance between two points in the data, and then the test of the hard-core model with fixed hard-core distance is simple (no parameters involved), and thus exact. This example is given in Myllymäki, Mrkvička, Grabarnik, Seijo, and Hahn (2017) and it is possible to prepare simulations for this case utilizing the function `rmh()` of the **spatstat** package. In the above example conditioning on the number of points was not employed.

## 6. Adjusted global envelope test for composite hypotheses

Let us test the fit of a Matern cluster process for the sapling data as an example of a composite hypothesis test. The adjusted test of the **GET** package is described in Section 2.3 of Myllymäki and Mrkvička (2023). The procedure was suggested by Baddeley, Hardegen, Lawrence, Milne, Nair, and Rakshit (2017) and extended for global envelopes in Myllymäki and Mrkvička (2023).

```
R> data(saplings)
R> saplings <- as.ppp(saplings, W = square(75))
R> par(mfrow = c(1,1), mgp = c(0, 0, 0), mar = c(0, 0, 0, 0))
R> plot(saplings, main = "")
```

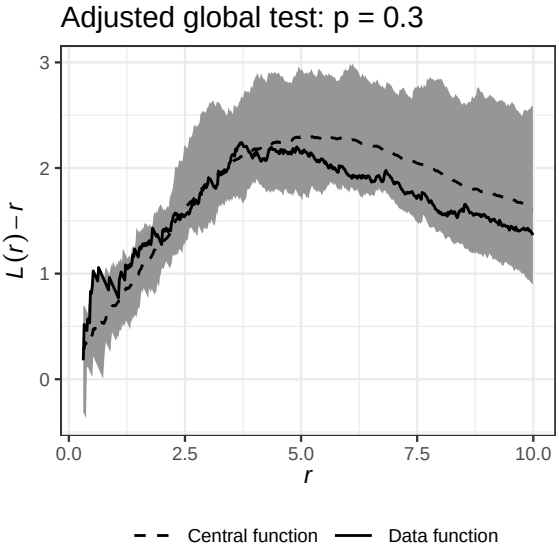


First define the $r$-distances and number of simulations. Here we set the number of simulations just to 19, for fast exploration of the code, but for serious analysis we recommend at least 499 simulations.

```
R> rmin <- 0.3; rmax <- 10; rstep <- (rmax-rmin)/500
R> r <- seq(0, rmax, by = rstep)
R> nsim <- 19 # Increase nsim for serious analysis!
```

The Matern cluster process can be fitted to the pattern using the `ppm()` function of **spatstat**. This utilizes minimum contrast estimation with the $K$-function. Then the adjusted global area rank envelope test can be performed using the function `GET.composite()`. Below we use the centred $L(r)$ function as the test function. The argument `type` specifies the global envelope test, see the help file of the `global_envelope_test()` function.

```
R> M1 <- kppm(saplings~1, clusters = "MatClust", statistic = "K")
R> adjenvL <- GET.composite(X = M1, nsim = nsim,
+    testfuns = list(L = list(fun="Lest", correction = "translate",
+         transform = expression(.-r), r = r)), # passed to envelope
+    type = "area", r_min = rmin, r_max = rmax, verbose = FALSE)
R> plot(adjenvL)
```




## 7. Testing global and local dependence on covariates

The function `GET.spatialF()` performs the one-stage global envelope tests based on spatial $F$- and $S$-statistics (Myllymäki, Kuronen, and Mrkvička 2020) to explore the effects of covariates in parametric point process models.

Let us look at a simple example of tropical rain forest trees.

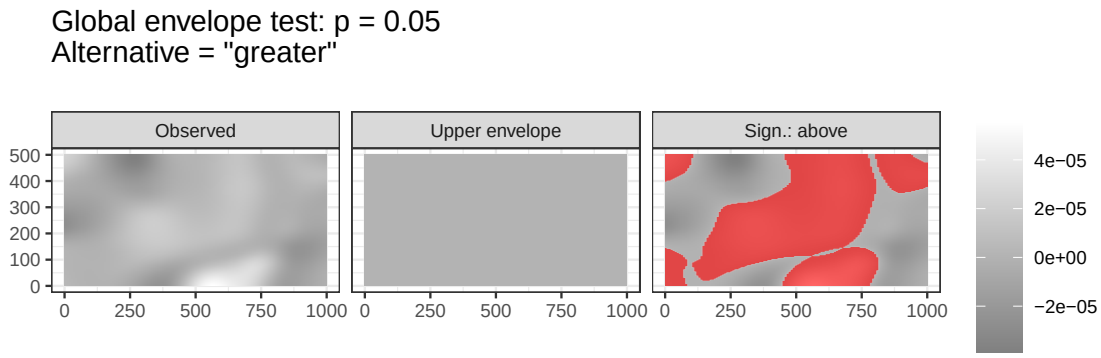
```
R> data(bei)
```

Let us study the effect of gradient on the intensity of the trees. We define the full model including this interesting covariate and the reduced model, which is otherwise the same as the full model, but the interesting covariate is excluded. Further the function `fitppm()` defines how the (full or reduced) model can be fitted to the point pattern.

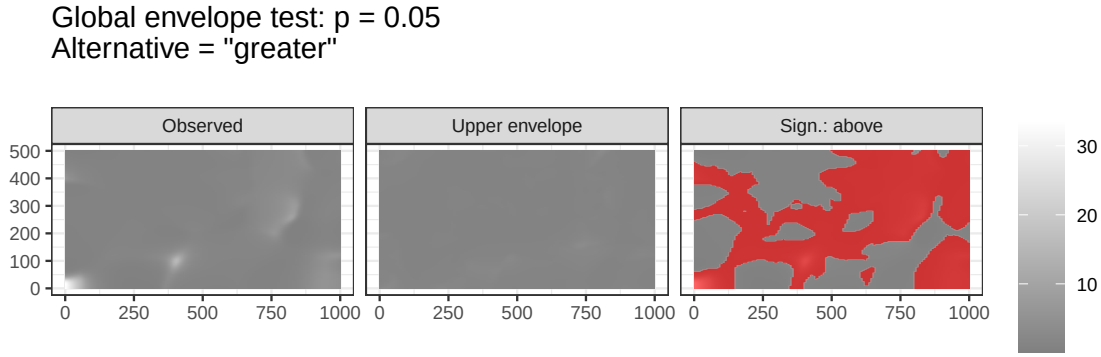
```
R> fullmodel <- ~ grad
R> reducedmodel <- ~ 1
R> fitppm <- function(X, model, covariates) {
+    ppm(X, model, covariates = covariates)
+  }
R> nsim <- 19 # Increase nsim for serious analysis!
R> res_sF <- GET.spatialF(bei, fullmodel, reducedmodel,
+                         fitppm, bei.extra, nsim)

Generating 19 simulated patterns ...1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
18,
19.

R> plot(res_sF$F, what = c("obs", "hi", "hi.sign"), sign.type = "col")
```




```
R> plot(res_sF$S, what = c("obs", "hi", "hi.sign"), sign.type = "col")
```



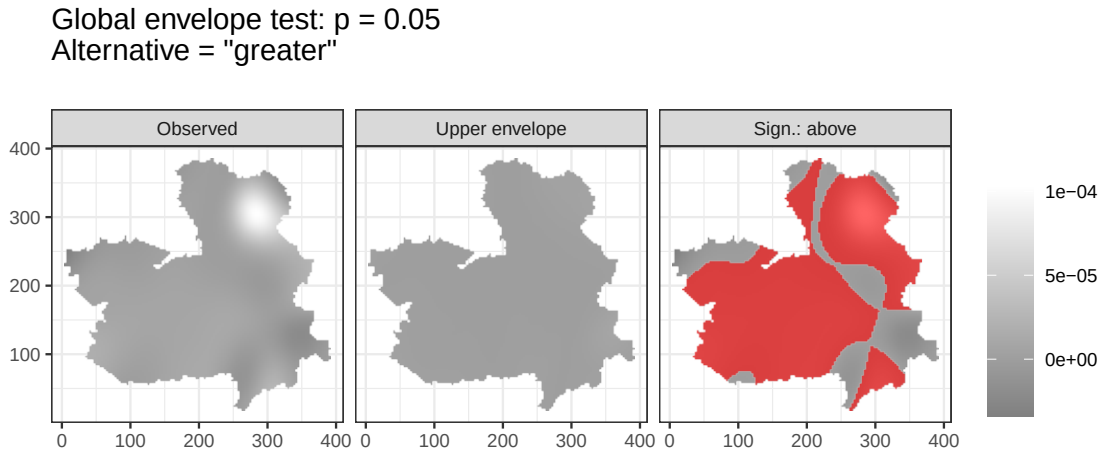

Another example is the test of the effect of elevation on the point pattern of lightnings.

```
R> # Example of forest fires
R> data("clmfires")
R> # Choose the locations of the lightnings in years 2004-2007:
R> pp.lightning <- unmark(subset(clmfires, cause == "lightning" &
+                   date >= "2004-01-01" & date < "2008-01-01"))
R> covariates <- clmfires.extra$clmcov100
R> covariates$forest <-
+    covariates$landuse == "conifer" | covariates$landuse == "denseforest" |
+    covariates$landuse == "mixedforest"
R> fullmodel <- ~ elevation + landuse
```

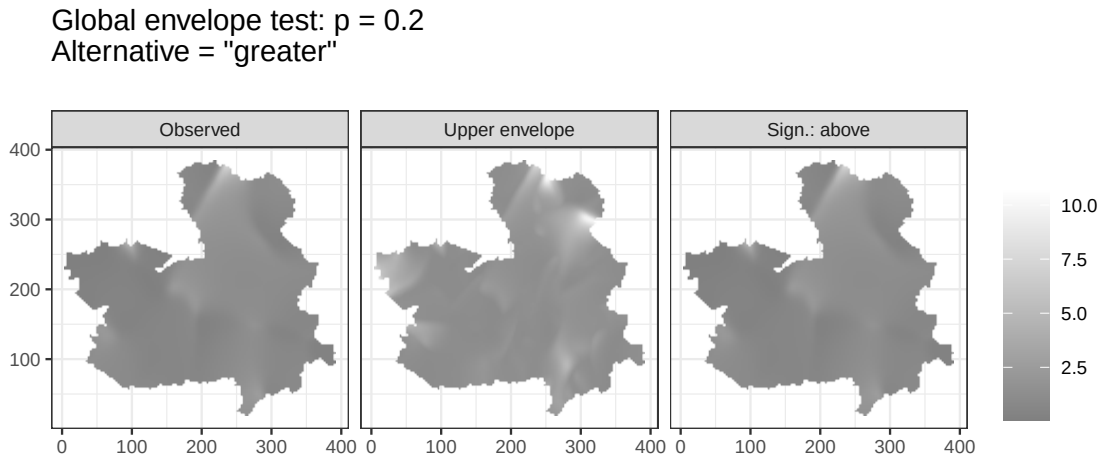
```
R> reducedmodel <- ~ landuse
R> nsim <- 19 # Increase nsim for serious analysis!
R> res_sF2 <- GET.spatialF(pp.lightning, fullmodel, reducedmodel,
+                          fitppm, covariates, nsim)

Generating 19 simulated patterns ...1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
18,
19.

R> plot(res_sF2$F, what = c("obs", "hi", "hi.sign"), sign.type = "col")
```




```
R> plot(res_sF2$S, what = c("obs", "hi", "hi.sign"), sign.type = "col")
```




Above only the inhomogeneous Poisson process was used as the model. Examples of the `fitfun()` functions for clustered and regular processes are:

```
R> # fitfun for the log Gaussian Cox Process with exponential covariance function
R> fitLGCPexp <- function(X, model, covariates) {
+    kppm(X, model, clusters = "LGCP", model = "exponential", covariates = covariates)
+  }
R> # fitfun for the hardcore process with hardcore radius 0.01
R> fitHardcore <- function(X, model, covariates) {
+    ppm(X, model, interaction = Hardcore(0.01), covariates = covariates)
+  }
```

## 8. An example analysis of the saplings data set

This is the example of Myllymäki *et al.* (2017, Supplement S4).

The saplings data set is available at the **GET** package.

```
R> data(saplings)
R> saplings <- as.ppp(saplings, W = square(75))
```

First choose the r-distances for $L(r)$ and $J(r)$ functions, respectively.

```
R> nr <- 500
R> rmin <- 0.3; rminJ <- 0.3
R> rmax <- 10; rmaxJ <- 6
R> rstep <- (rmax-rmin)/nr; rstepJ <- (rmaxJ-rminJ)/nr
R> r <- seq(0, rmax, by = rstep)
R> rJ <- seq(0, rmaxJ, by = rstepJ)
```

### 8.1. The CSR test based on the L(r)-r function

Note: CSR is simulated by fixing the number of points and generating `nsim` simulations from the binomial process, i.e. we deal with a simple hypothesis.

First, the envelope function of the spatstat package can be used to generate nsim simulations under CSR and to calculate the centred $L$-function for the data and each simulation.

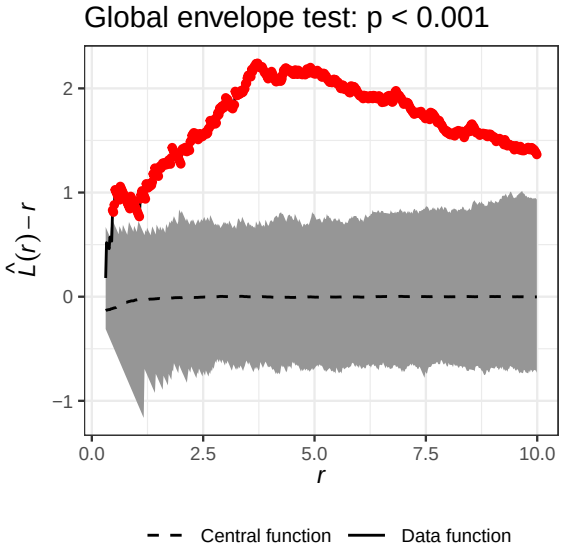
```
R> nsim <- 1999 # Number of simulations
R> env <- envelope(saplings, nsim = nsim,
+    simulate = expression(runifpoint(ex = saplings)), # Simulate CSR
+    fun = "Lest", correction = "translate", # estimator of L with transl. edge corr.
+    transform = expression(.-r), # Take the L(r)-r function instead of L(r)
+    r = r, # Specify the distance vector
+    savefuns = TRUE, # Save the estimated functions
+    verbose = FALSE)
```

Then the curves can be cropped to the desired interval of distances $[r_{\min}, r_{\max}]$

```
R> curve_set <- crop_curves(env, r_min = rmin, r_max = rmax)
```

And a global envelope test done by means of the `global_envelope_test()` function (`type="rank"` and larger `nsim` was used in Myllymäki *et al.* (2017, S4):

```
R> res_sapl <- global_envelope_test(curve_set, type = "erl")
R> plot(res_sapl) + ylab(expression(italic(hat(L)(r)-r)))
```




The CSR hypothesis is clearly rejected and the rank envelope indicates clear clustering of saplings. As a next step, we explore the Matern cluster process as a null model. This is a composite hypothesis.

## 8.2. Testing the fit of a Matern cluster process

First we fit the Matern cluster process to the pattern. Here we use the minimum contrast estimation with the $K$-funcion (the pair correction function can be chosen by setting `statistic = "pcf"`).

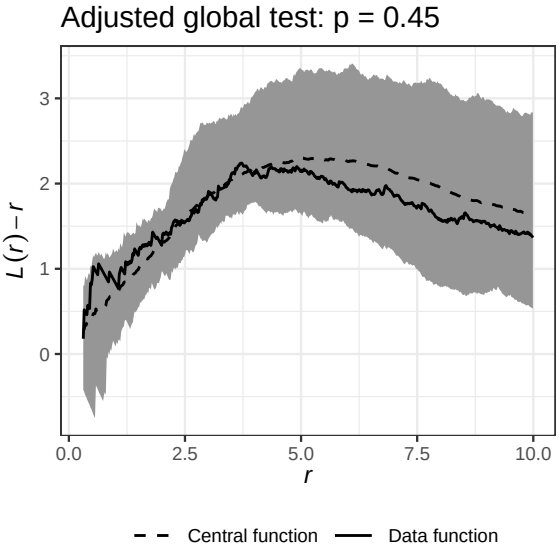
```
R> fitted_model <- kppm(saplings~1, clusters = "MatClust", statistic = "K")
```

Next step is to perform the adjusted directional quantile global envelope test using the centred $L$-function. (For the rank envelope test, choose `type = "rank"` instead and increase `nsim`.)

```
R> nsim <- 19 # 19 just for experimenting with the code!!
R> #nsim <- 499 # 499 is ok for type = 'qdir' (takes > 1 h)
R> adjenvL_sapl <- GET.composite(X = fitted_model,
+    fun = "Lest", correction = "translate",
+    transform = expression(.-r), r = r,
+    type = "qdir", nsim = nsim, nsimsub = nsim,
+    r_min = rmin, r_max = rmax, verbose = FALSE)
```

The result can then be plotted:

```
R> plot(adjenvL_sapl) + ylab(expression(italic(L(r)-r)))
```

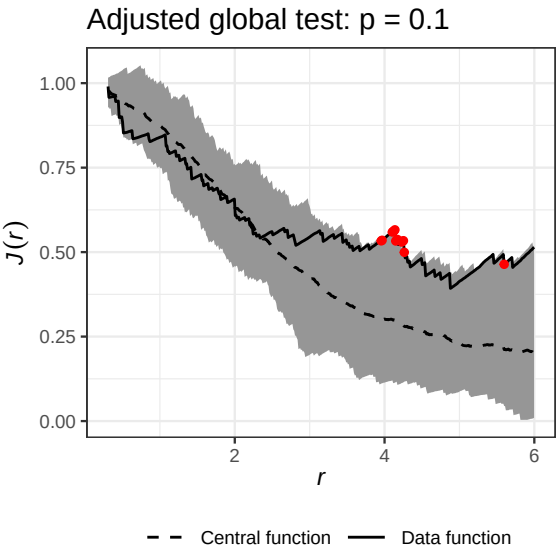



From the test with the centred $L$-function, it appears that the Matern cluster model would be a reasonable model for the saplings pattern. To further explore the goodness-of-fit of the Matern cluster process, test the model with the $J$-function: This takes quite some time if `nsim` is reasonably large.

```
R> adjenvJ_sapl <- GET.composite(X = fitted_model,
+    fun = "Jest", correction = "none", r = rJ,
+    type = "qdir", nsim = nsim, nsimsub = nsim,
+    r_min = rminJ, r_max = rmaxJ, verbose = FALSE)
```

And, plot the result

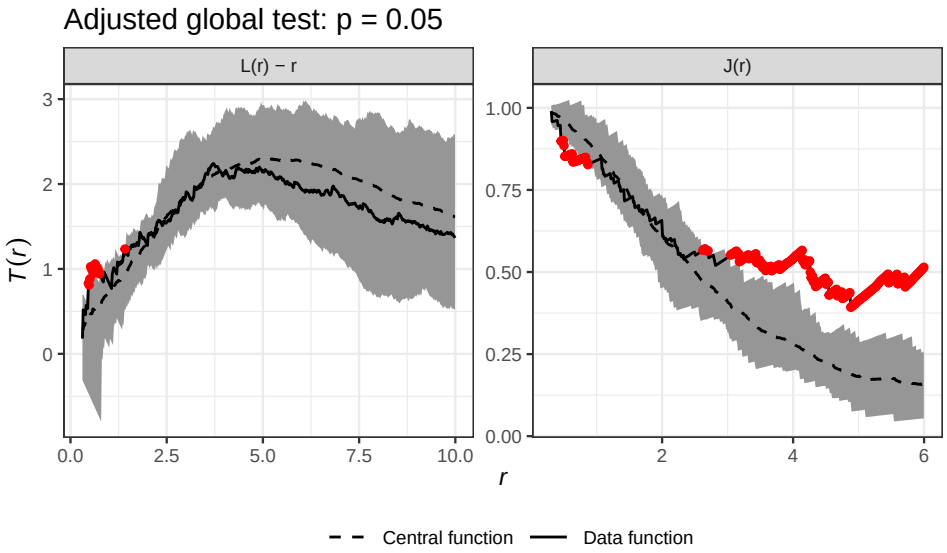
```
R> plot(adjenvJ_sapl) + ylab(expression(italic(J(r))))
```



Thus, it appears that the Matern cluster process is not adequate for the saplings data. The Matern cluster model might be interpreted as a regeneration process in circular gaps between large trees. However, it is possible that the gap openings in the forest were not exactly circular, thereby leading to the rejection of the model by the $J$-function.

It is also possible to test the fit of the Matern cluster process simultaneously by the two test functions:

```
R> adjenvLJ_sapl <- GET.composite(X = fitted_model,
+    testfuns = list(L = list(fun = "Lest", correction = "translate",
+                             transform = expression(.-r), r = r),
+                    J = list(fun = "Jest", correction = "none", r = rJ)),
+    type = "erl", nsim = nsim, nsimsub = nsim,
+    r_min = c(rmin, rminJ), r_max = c(rmax, rmaxJ),
+    save.cons.envelope = TRUE, verbose = FALSE)
R> plot(adjenvLJ_sapl)
```




## References

Baddeley A, Hardegen A, Lawrence T, Milne RK, Nair G, Rakshit S (2017). "On Two-Stage Monte Carlo Tests of Composite Hypotheses." *Computational Statistics & Data Analysis*, **114**, 75–87. doi:https://doi.org/10.1016/j.csda.2017.04.003.

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

Mrkvička T, Myllymäki M, Hahn U (2017). "Multiple Monte Carlo Testing, with Applications in Spatial Point Processes." *Statistics and Computing*, **27**(5), 1239–1255. doi:10.1007/s11222-016-9683-9.

Myllymäki M, Kuronen M, Mrkvička T (2020). "Testing Global and Local Dependence of Point Patterns on Covariates in Parametric Models." *Spatial Statistics*, **42**, 100436. ISSN 2211-6753. doi:10.1016/j.spasta.2020.100436.

Myllymäki M, Mrkvička T (2023). "**GET**: Global Envelopes in R." arXiv:1911.06583 [stat.ME]. doi:10.48550/arXiv.1911.06583.

Myllymäki M, Mrkvička T, Grabarnik P, Seijo H, Hahn U (2017). "Global Envelope Tests for Spatial Processes." *Journal of the Royal Statistical Society B*, **79**, 381–404. doi:10.1111/rssb.12172.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag, New York. ISBN 978-3-319-24277-4.

### Affiliation:

Mari Myllymäki
Natural Resources Institute Finland (Luke)
Latokartanonkaari 9
FI-00790 Helsinki, Finland
E-mail: mari.myllymaki@luke.fi
URL: https://www.luke.fi/en/experts/mari-myllymaki/