FunGraph-vignette-v1.0

Jinshuai Zhao


```
library(FunGraph)
```

About

FunGraph can analyze the roadmap of how each locus affects phenotypic variation through its own direct effect and a complete set of indirect effects due to regulation by other loci co-existing in large-scale networks.

1. Built-in data set

A brief introduction to the built-in datasets included in the FunGraph package

1.1 s

The result calculated by FunMap function

```
View(s)
```

1.2 remarker_data_par

Parameters of significant loci,Used in section 3

```
knitr::kable(head(remarker_data_par[, 1:8]))
```

8.337230	3.497184	0.2957548	8.607717	2.998936	0.2851245	0.6987277	0.4435762
8.592442	3.103905	0.2933308	8.317543	3.403687	0.2846634	0.6967723	0.4432327
8.553887	3.029294	0.2889793	8.460743	3.268071	0.2897523	0.7070803	0.4468609
8.432053	3.097884	0.2937111	8.506934	3.268015	0.2876961	0.7081221	0.4471769
8.358974	3.381529	0.2920958	8.809231	2.843473	0.2847465	0.6919556	0.4406727
8.425480	3.214206	0.2893257	8.723735	3.195815	0.2903528	0.7061321	0.4465551

1.3 remarker_effect

The net genetic effects of significant loci

```
knitr::kable(head(remarker_effect[, 1:11]))
```

	0.5	1	1.5	2	4	6	8	10	12	16	20
165	0.1111488	0.1163742	0.1209287	0.1246919	0.1304946	0.1229884	0.1099969	0.0992102	0.0932589	0.0911523	0.0928606
621	0.0796838	0.0887020	0.0979011	0.1071026	0.1399249	0.1583058	0.1591641	0.1485382	0.1344875	0.1125605	0.1019486
1308	0.0449891	0.0480284	0.0508803	0.0534711	0.0599899	0.0592036	0.0530533	0.0454013	0.0389173	0.0315371	0.0288997
1924	0.0244523	0.0275280	0.0305829	0.0335220	0.0420878	0.0419111	0.0327656	0.0191641	0.0058885	0.0120869	0.0200960
1930	0.1287813	0.1366015	0.1439170	0.1505740	0.1683445	0.1708693	0.1635977	0.1543176	0.1473441	0.1414748	0.1406649
2063	0.0248274	0.0276860	0.0307054	0.0338596	0.0471123	0.0594547	0.0689491	0.0751736	0.0788070	0.0817847	0.0825346

1.4 fourth_effect_cluster

According to BIC, the optimal cluster number is 14

```
View(fourth_effect_cluster)
```

1.5 fourteen_cluster_mean_effect

Mean curves of 14 types of modules

```
View(fourteen_cluster_mean_effect)
```

2. Input data

Before running FunGraph, user need to provide genotypic and phenotypic datasets, and they should be cleaned and merged to exactly the same format of the example data.

2.1 Phenotypic data

Phenotypic dataset contains control group and the treatment group, each with same sample id as row names and same column number to represent the times phenotypic data were measured.S_mo has been added to the built-in data set to run the code directly.

```
View(S_mo)
```

X0.5	X1	X1.5	X2	X4	X6	X8	X10	X12	X16	X20
1.612616	1.5863064	1.952254	2.117031	2.932787	6.097333	6.902934	7.722446	8.288681	8.190092	7.867866
2.973901	2.8866764	3.436882	4.375284	5.573351	6.979879	8.341971	9.145597	9.779405	8.712159	8.546202
1.392786	1.4631533	1.947402	1.908619	1.537354	4.567139	5.308634	7.261710	7.461244	8.177293	6.500922
1.227642	0.8788837	2.025432	2.104262	4.509321	5.910901	6.996811	8.468599	8.354112	8.306056	8.441991
2.449149	2.2877501	2.410949	2.366622	2.427286	4.312015	5.691001	7.258493	7.543431	7.564375	7.481482
1.722264	1.4287763	1.171185	1.710733	2.389138	2.769667	5.033761	6.472184	6.906527	7.734474	6.019460

2.2 Genotypic data

Genotypic data have sample id as columns and SNP id as row names, and may include additional information such as the SNP position in linkage for manhattan plot.S_SNP has been added to the built-in data set to run the code directly.

```
View(S_SNP)
```

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
165	0	1	0	0	0	0	0	1	1	1

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
621	0	0	1	1	1	1	1	0	0	0
1308	0	0	1	1	1	1	1	1	0	1
1924	0	1	1	1	1	1	1	1	1	0
1930	0	1	0	0	0	0	0	1	1	0
2063	0	1	0	0	0	0	0	1	1	0

3. FunMap

Functional mapping (biFunMap) is crucial to this model, for it excavates how specific QTLs determines the complex trait. The mean vector and covariance structure should be modeled according to the design of the experiment.

3.1 FunMap calculation

The output results are of a list type, including a, b, r parameters and likelihood values of H0, a, b, r parameters and likelihood values of H1 of each SNP, and LR values of each SNP. s has been added to the built-in data set to run the code directly.

```
s <- Funmap_function(funmap_data = S_mo, funmap_data_snp = S_SNP, times = c(0.5,
    1, 1.5, 2, 4, 6, 8, 10, 12, 16, 20))
```

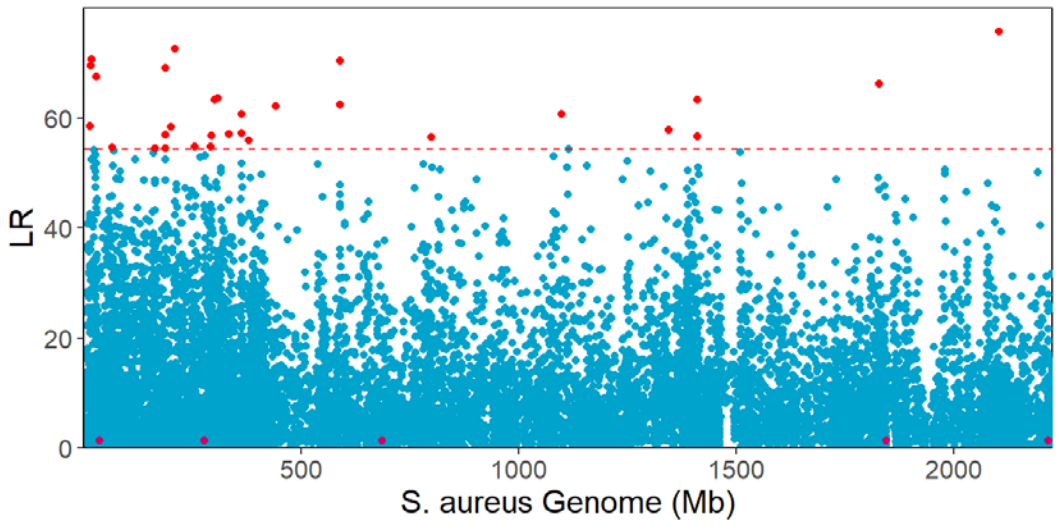
3.2 1000 permutation tests determine the threshold

```
permutaion(funmap_data = S_mo, funmap_data_snp = S_SNP, times = c(0.5, 1, 1.5, 2,
    4, 6, 8, 10, 12, 16, 20), permutaion_times = 1000, rownumber = 1000)
```

3.3 FunMap results visulation

Manhattan plot using calculate LR values

```
plot_Manhattan_fn(funmap_data_snp = S_SNP, LR = s$LR, threshold_value = 54.33574)
```



4. Genetic effect curve

4.1 Calculation of genetic curve values

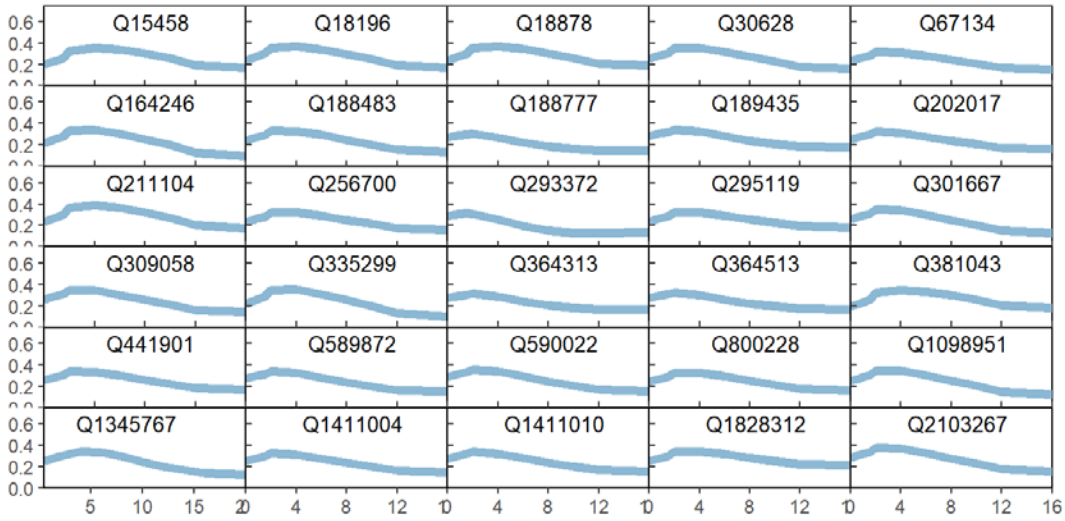
The net genetic effect of each gene locus (SNP) was calculated. remarker_effect and remarker_data_par has been added to the built-in data set to run the code directly.

```
remarker_data_par <- t(sapply(1:length(s$LR), function(c) s$All_H1_result[c][[1]]$H1_par))
remarker_effect <- Get_effect(S_SNP, remarker_data_par, times = c(0.5, 1, 1.5, 2,
    4, 6, 8, 10, 12, 16, 20))
```

4.2 Display diagram of significant QTLs genetic curve

sig_SNP:Significant QTLS were screened by threshold sig_data_par:Extracting significant QTL parameters sig_effect:The genetic effect values of significant QTLs were calculated

```
library(ggplot2)
library(patchwork)
sig_SNP <- S_SNP[which(s$LR > 54.33574), ]
sig_data_par <- t(sapply(c(which(s$LR > 54.33574)), function(c) s$All_H1_result[c][[1]]$H1_par))
sig_effect <- Get_effect(sig_SNP, sig_data_par, times = c(0.5, 1, 1.5, 2, 4, 6, 8,
    10, 12, 16, 20))
effect_plot_fn <- function(effect_data, colnn, times) {
    tmp <- as.data.frame(t(rbind(effect_data, times = times)))
    p1 <- ggplot(tmp) + geom_line(data = tmp, aes(x = times, y = tmp[, colnn]), size = 2,
        color = "#8CB8D4") + theme_classic() + labs(y = NULL, x = NULL) + theme(panel.background = element_rect(color =
"black",
        fill = "transparent"), plot.margin = unit(c(0, 0, 0, 0), "lines")) + scale_x_continuous(expand = c(0,
        0)) + annotate("text", label = paste0("Q", rownames(effect_data)[colnn]),
        x = 8.5, y = 0.6, size = 3.5) + scale_y_continuous(expand = c(0, 0), limits = c(0,
        0.75))

    return(p1)
}
effect_plot_left_fn <- function(i) {
    effect_plot_fn(effect_data = sig_effect, colnn = i, times = c(0, 0.5, 1, 1.5,
        2, 4, 6, 8, 10, 12, 16, 20)) + theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
        axis.ticks.length.x = unit(-0.1, "cm"))
}
effect_plot_middle_fn <- function(i) {
    effect_plot_fn(effect_data = sig_effect, colnn = i, times = c(0, 0.5, 1, 1.5,
        2, 4, 6, 8, 10, 12, 16, 20)) + theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
        axis.ticks.length.x = unit(-0.1, "cm"), axis.title.y = element_blank(), axis.text.y = element_blank(),
        axis.ticks.length.y = unit(-0.1, "cm"))
}
effect_plot_down_fn <- function(i) {
    effect_plot_fn(effect_data = sig_effect, colnn = i, times = c(0, 0.5, 1, 1.5,
        2, 4, 6, 8, 10, 12, 16, 20)) + theme(axis.title.y = element_blank(), axis.text.y = element_blank(),
        axis.ticks.length.y = unit(-0.1, "cm"))
}

sig_effect_plot <- list()

for (i in c(1, 6, 11, 16, 21)) {
    sig_effect_plot[[i]] <- effect_plot_left_fn(i)
}

for (i in c(2, 3, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15, 17, 18, 19, 20, 22, 23, 24,
    25)) {
    sig_effect_plot[[i]] <- effect_plot_middle_fn(i)
}
for (i in c(27, 28, 29, 30)) {
    sig_effect_plot[[i]] <- effect_plot_down_fn(i)
}
sig_effect_plot[[26]] <- effect_plot_fn(effect_data = sig_effect, colnn = 26, times = c(0.5,
    1, 1.5, 2, 4, 6, 8, 10, 12, 16, 20))

sig_effect_plot_all <- sig_effect_plot[[1]]
for (i in c(2:30)) {
    sig_effect_plot_all <- sig_effect_plot_all + sig_effect_plot[[i]]
}
sig_effect_plot_all <- sig_effect_plot_all + plot_layout(ncol = 5)
sig_effect_plot_all
```



5. Functional cluster

FunClu characterize whether and how a specific locus determines the developmental trajectories of the complex trait expressed over environments.

5.1 Calculation of genetic curve values

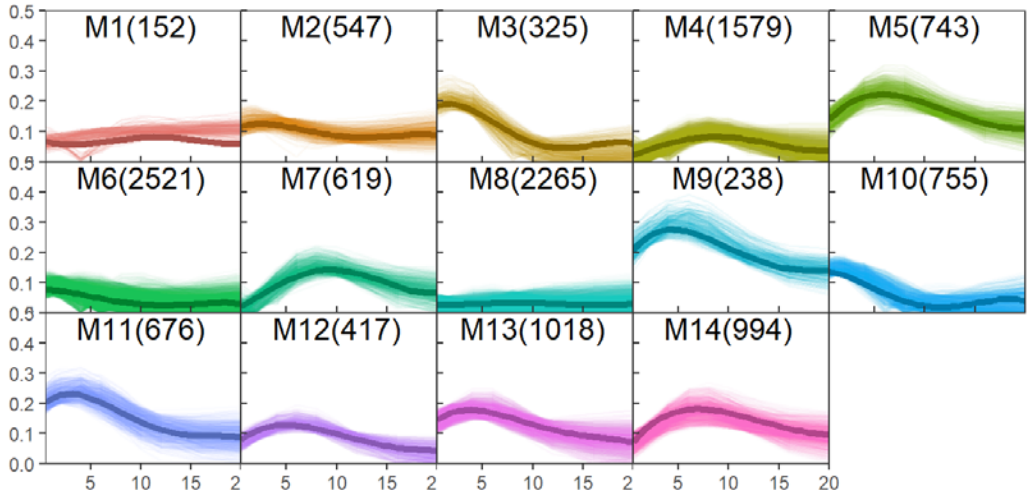
fourth_effect_cluster has been added to the built-in data set to run the code directly.

```
bic_list <- list()
bic <- list()
for (k in 1:25) {
    bic <- Funcluster(remarker_effect, times = c(0.5, 1, 1.5, 2, 4, 6, 8, 10, 12,
        16, 20), rep = 1, k, legendre_order = 4)
    bic_list[[i]] <- bic
}
fourth_effect_cluster <- bic_list[[i]]
```

5.2 Display the best clustering graph

```
library(reshape2)
library(orthopolynom)
## 载入需要的程辑包：polynom
clustered_data <- fourth_effect_cluster
par1 <- clustered_data[[2]][[1]]$curve_par
times <- c(0.5, 1, 1.5, 2, 4, 6, 8, 10, 12, 16, 20)
clustered_df <- data.frame(cbind(row.names(clustered_data[[2]][[1]][["clustered_data"]]),
    clustered_data[[2]][[1]][["clustered_data"]]))
colnames(clustered_df) <- c("ID", times, "cluster")
long_df <- melt(clustered_df, id.vars = c("ID", "cluster"))
long_df[, 4] <- as.numeric(long_df[, 4])
colnames(long_df) <- c("SNP", "cluster", "time", "effect")
normalization <- function(x) {
    ((x - min(x))/(max(x) - min(x)) * 0.075) + 0.02
}

alpha <- 1/table(clustered_df$cluster)
alpha <- normalization(alpha)
get_mean_df <- function(data, times, legendre_order) {
    legendre_fit <- function(par) {
        x <- seq(min(times), max(times), length = 30)
        fit <- sapply(1:length(par), function(c) par[c] * legendre.polynomials(n = legendre_order,
            normalized = F)[[c]])
        legendre_fit <- as.matrix(as.data.frame(polynomial.values(polynomials = fit,
            x = scaleX(x, u = -1, v = 1))))
        x_interpolation <- rowSums(legendre_fit)
        return(x_interpolation)
    }
    mean_df <- sapply(1:nrow(data), function(c) legendre_fit(as.numeric(data[c, ])))
    colnames(mean_df) <- c(1:nrow(data))
    mean_df <- melt(mean_df)
    colnames(mean_df) <- c("time", "cluster", "effect")
    mean_df$time <- seq(min(times), max(times), length = 30)
    return(mean_df)
}
mean_curve <- get_mean_df(par1, times = c(0.5, 1, 1.5, 2, 4, 6, 8, 10, 12, 16, 20),
    4)


# 其中h是色相，范围越大，相邻颜色之间差异越大；c是饱和度，值越大色彩越浓艳饱满；l是亮度，大亮小暗。
library(RColorBrewer)
library(scales)

cols1 <- hue_pal(h = c(0, 720) + 15, c = 90, l = 60)(30)
darken <- function(color, factor = 1.3) {
    col <- col2rgb(color)
    col <- col/factor
    col <- rgb(t(col), maxColorValue = 255)
    col
}
cluster_polt_fn <- function(i) {
    cluster <- long_df[which(long_df$cluster == i), ]
    cluster$time <- as.numeric(as.character(cluster$time))
    cluster_mean <- mean_curve[which(mean_curve$cluster == i), ]
    p <- ggplot() + geom_line(cluster, mapping = aes(time, effect, group = SNP),
        color = cols1[i], alpha = alpha[i]) + geom_line(cluster_mean, mapping = aes(time,
        effect), color = darken(cols1[i]), size = 1.5) + theme_bw() + theme(panel.grid = element_blank()) +
        xlab("") + ylab("") + scale_x_continuous(expand = c(0, 0)) + scale_y_continuous(expand = c(0,
        0), limits = c(0, 0.5)) + annotate("text", label = paste0("M", unique(cluster$cluster),
        "(", dim(cluster)[1]/11, ")"), x = 10, y = 0.45, size = 5) + theme(plot.margin = unit(c(0,
        0, 0, 0), "lines"))
    return(p)
}
cluster_polt <- list()
cluster_polt_left_fn <- function(i) {
    p <- cluster_polt_fn(i) + theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
        axis.ticks.length.x = unit(-0.1, "cm"))
    return(p)
}
```

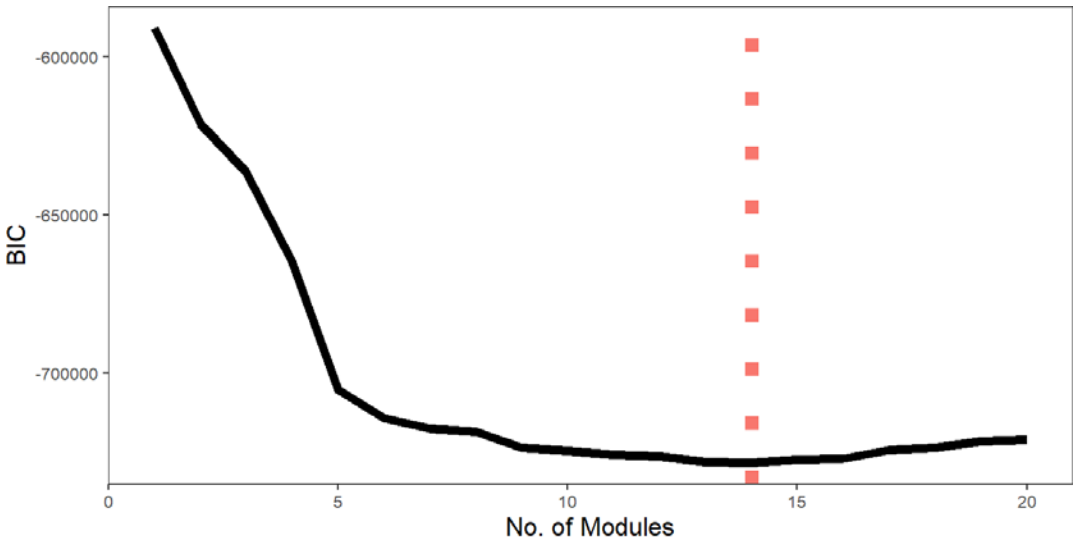
```
cluster_polt_middle_fn <- function(i) {
    p <- cluster_polt_fn(i) + theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
        axis.ticks.length.x = unit(-0.1, "cm"), axis.title.y = element_blank(), axis.text.y = element_blank(),
        axis.ticks.length.y = unit(-0.1, "cm"))
    return(p)
}

cluster_polt_down_fn <- function(i) {
    p <- cluster_polt_fn(i) + theme(axis.title.y = element_blank(), axis.text.y = element_blank(),
        axis.ticks.length.y = unit(-0.1, "cm"))
    return(p)
}
cluster_polt <- list()
for (i in c(1, 6)) {
    cluster_polt[[i]] <- cluster_polt_left_fn(i)
}

for (i in c(2, 3, 4, 5, 7, 8, 9, 10)) {
    cluster_polt[[i]] <- cluster_polt_middle_fn(i)
}

for (i in c(12, 13, 14)) {
    cluster_polt[[i]] <- cluster_polt_down_fn(i)
}

cluster_polt[[11]] <- cluster_polt_fn(11)
cluster_polt_all <- cluster_polt[[1]]
for (i in c(2:clustered_data[[1]][1])) {
    cluster_polt_all <- cluster_polt_all + cluster_polt[[i]]
}

cluster_polt_all <- cluster_polt_all + plot_layout(ncol = 5)
cluster_polt_all
```



5.3 Display the BIC graph

The value of BIC was calculated from the Funcluster function.

```
BIC <- c(-591045.6, -621432.9, -636578.2, -664730.9, -705232.3, -714223.3, -717334.2,
    -718324.2, -723442.1, -724443.2, -725674.3, -726223.2, -727881.3, -728135.5,
    -727281, -726923.3, -724106, -723333.7, -721561.9, -720992.1)
BIC_data <- data.frame(bic = BIC, m = c(1:20))
BIC_plot <- ggplot() + geom_line(data = BIC_data, aes(x = as.numeric(BIC_data$m),
    y = as.numeric(BIC_data$bic)), size = 2) + theme_bw() + labs(x = "No. of Modules",
    y = "BIC") + geom_vline(xintercept = 14, size = 3, color = "#F8766D", linetype = "dotted") +
    scale_x_continuous(expand = c(0.05, 0.05)) + scale_y_continuous(expand = c(0.05,
    0.05)) + theme(axis.text = element_text(size = 8), axis.title = element_text(size = 12),
    plot.margin = unit(c(0, 0, 0, 0), "lines")) + theme(legend.position = "none") +
    theme(panel.grid = element_blank())
BIC_plot
```



6. Decomposition_curve

The net genetic effects of SNPs were divided into independent genetic effects (the effects generated or emitted by the SNP itself) and dependent genetic effects (the effects from other SNPs outside).Here we show the curve decomposition of the top layer network, and the same for other layers.The result contains elements for drawing curves and networks.

```
times <- c(0.5, 1, 1.5, 2, 4, 6, 8, 10, 12, 16, 20)
clustered_data <- fourth_effect_cluster
par1 <- clustered_data[[2]][[1]]$curve_par
get_mean_effect <- function(data, times, legendre_order) {
    legendre_fit <- function(par) {
        x <- seq(min(times), max(times), length = 30)
        fit <- sapply(1:length(par), function(c) par[c] * legendre.polynomials(n = legendre_order,
            normalized = F)[[c]])
        legendre_fit <- as.matrix(as.data.frame(polynomial.values(polynomials = fit,
            x = scaleX(x, u = -1, v = 1))))
        x_interpolation <- rowSums(legendre_fit)
        return(x_interpolation)
    }
    mean_df <- sapply(1:nrow(data), function(c) legendre_fit(as.numeric(data[c, ])))
    colnames(mean_df) <- c(1:nrow(data))
    return(mean_df)
}
mean_effect <- t(get_mean_effect(par1, times, 4))
fourteen_cluster_mean_effect <- Decomposition_curve(exp_index = seq(min(times), max(times),
    length = 30), legendre_order = 4, data = mean_effect, alpha = 1e-06)
fourteen_cluster_mean_effect
```

7. QTLs interactive network

Here we use the results of Section 5(fourteen_cluster_mean_effect$after) to map the top-level network,fourteen_cluster_mean_effect has been added to the built-in data set to run the code directly.

```
fourteen_cluster_mean_effect$after
##    from to   dep_effect color
## 1     2  1 0.0152691861     -
## 2     4  1 0.0072015048     -
## 3     6  1 0.0141639262     +
## 4     1  2 0.0131214552     -
## 5     6  2 0.0205609150     -
## 6     8  2 0.0001805243     -
## 7    12  2 0.0141992917     +
## 8     2  3 0.0044219523     +
## 9     4  3 0.0043202321     -
## 10    6  3 0.0162453174     -
## 11    1  4 0.0168967510     -
## 12    7  4 0.0039358485     +
## 13    8  4 0.0015239157     -
## 14   14  4 0.0406757621     +
## 15    4  5 0.0012032588     -
## 16    8  5 0.0167668455     -
## 17   12  5 0.0269170554     +
## 18   14  5 0.0773274964     +
## 19    1  6 0.0018402147     -
## 20    2  6 0.0156774890     -
## 21    8  6 0.0007396216     -
## 22   10  6 0.0125265301     -
## 23    1  7 0.0189169001     -
## 24    4  7 0.0190001715     +
## 25    6  7 0.0359872841     +
## 26    1  8 0.0007975217     -
## 27    4  8 0.0002773634     -
## 28    9  8 0.0007647858     +
## 29   10  8 0.0021641058     +
## 30   13  8 0.0020298244     +
## 31   14  8 0.0018728672     -
## 32    2  9 0.1139923593     -
## 33    8  9 0.0087627414     -
## 34   12  9 0.0760234303     +
## 35    4 10 0.0199706938     -
## 36    6 10 0.0128989113     -
## 37    8 10 0.0003523787     -
## 38    2 11 0.0002658893     -
## 39    6 11 0.0375100216     -
## 40    8 11 0.0038631144     -
## 41   12 11 0.0371529412     +
## 42    2 12 0.0141271060     -
## 43    8 12 0.0065692333     -
## 44   13 12 0.0325444657     -
## 45   14 12 0.0312780747     +
## 46    1 13 0.0294509510     -
## 47    6 13 0.0358489921     -
## 48    8 13 0.0005547924     +
## 49   12 13 0.0387844078     +
## 50    1 14 0.0514245053     -
## 51    4 14 0.0052119496     +
## 52    8 14 0.0031545800     -
## 53   12 14 0.0077550678     +
```

Call write_xlsx to save fourteen_cluster_mean_effect$after as an XLSX file and import it into Cytoscape parameters,as follow: