First steps with R-DGSA

Ogy Grujic

October 20, 2016

This document serves as a demonstration of the capabilities of DGSA package. DGSA stands for "Distance Based Generalized Sensitivity Analysis", and it is a method for computing parameter sensitivities of computer experiments with complex intputs and complex outputs. Originally the method was developed by Fenwick et al 2012 (reference).

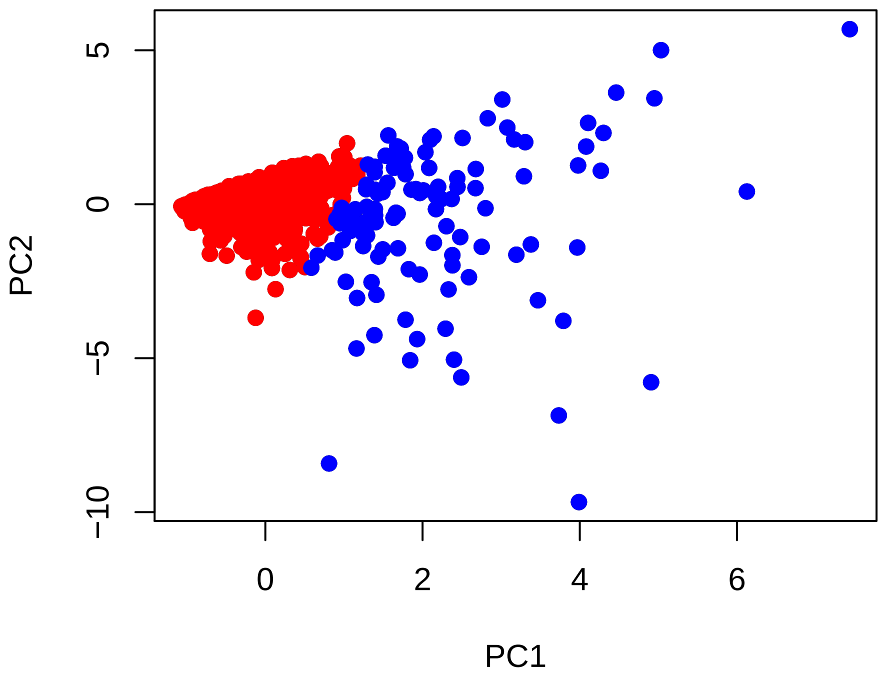
DGSA package implements the original method and expands the original visualization developments by employing advanced graphics provided by "corrplot" and "ggplot2" packages.

DGSA on Demo Dataset:

The first step in dgsa is to cluster the outputs in some way. Responses can be functional, vectors, scalars, or even mixed functions and scalars. The most adequate clustering method depends on the data, therefore it is left to the user to decide which clustering method to chose in data pre-processing stage. The only cluster related input necesarry for dgsa are the cluster codes associated with each data entry or design point.

In the example given below we used simple principal component analysis with kmeans clustering.

```r
library(DGSA)
INPUT         <- read.csv("../data/Input.csv")
OUTPUT.damage <- read.csv("../data/Output_damage.csv")
comps  <- prcomp(OUTPUT.damage)
scores <- t(t(comps$x[,1:2])/comps$sdev[1:2])
clustering = kmeans(scores, 2, 50)$cluster
plot(scores)
points(scores[clustering == 1,], col="red", pch = 19)
points(scores[clustering == 2,], col="blue", pch=19)
```



After clustering, user can either proceed straight to computations, or perform some exploratory data analysis of the cdf's in order to make an educated decision on how many bins to use for computation of sensitivities of interactions or perhaps even remove some parameters. The package provides convenient tools for such exploratory data analyses. The first function that we will introduce is called "plotCDFS", which as the name suggests plots parameter cdfs in the same form as they are used in the dgsa code. The code "all*" specifies that we are interested in cdfs of all available input parameters.

```
plotCDFS(clustering, INPUT, .code = "all*")
```

Similarly, by changing the ".code" parameter we can also plot cdfs of a single parameter, or even CDFs of interactions. The following two examples demonstrate such capabilities.

```
plotCDFS(clustering, INPUT, .code = "beta")
plotCDFS(clustering, INPUT, .code = "lambda")
```

```
plotCDFS(clustering, INPUT, .code = "beta|lambda", .nBins = 2)
```

```
plotCDFS(clustering, INPUT, .code = "beta|lambda", .nBins = 3)
```

Computing DGSA:

To compute parameter sensitivities with DGSA one should use the following function.

```
myDGSA <- dgsa(clustering, INPUT, .interactions = TRUE, .nBoot = 100, .nBins = 3, .alpha = 0.95, .paral
```

Note the controls that are available to users, such as the number of bootstrapped samples (.nBoot), number of bins for interactions (.nBins), significance test factor (.alpha), boolean operator (.interactions) specifying

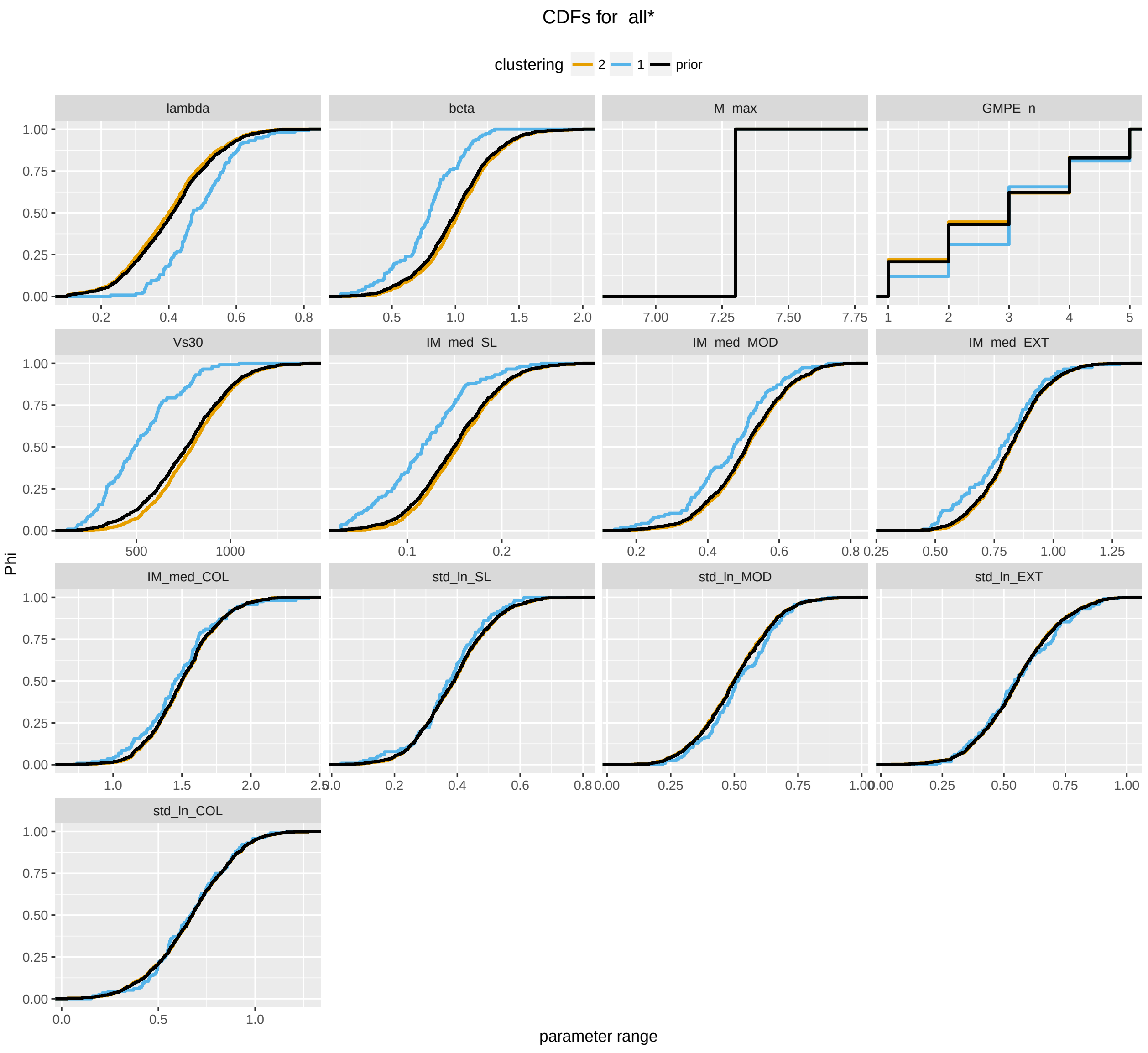
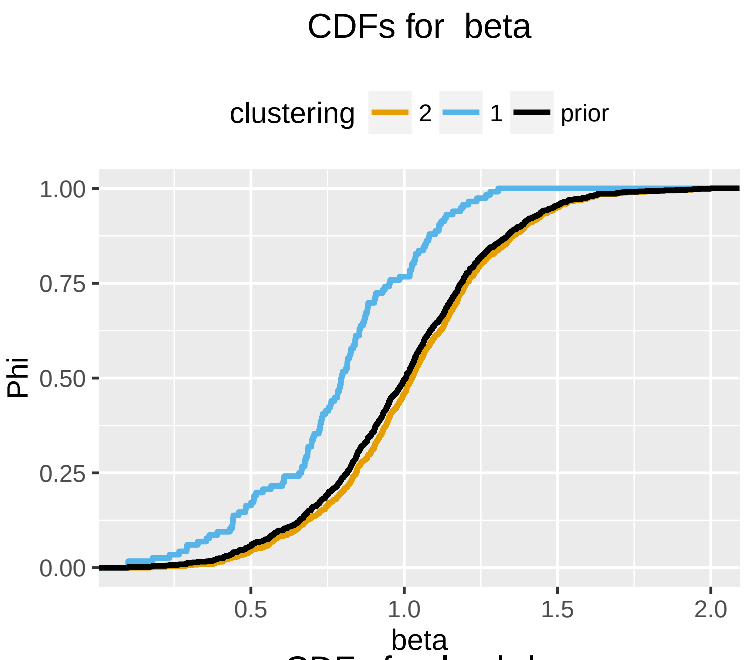


Figure 1: All CDFs



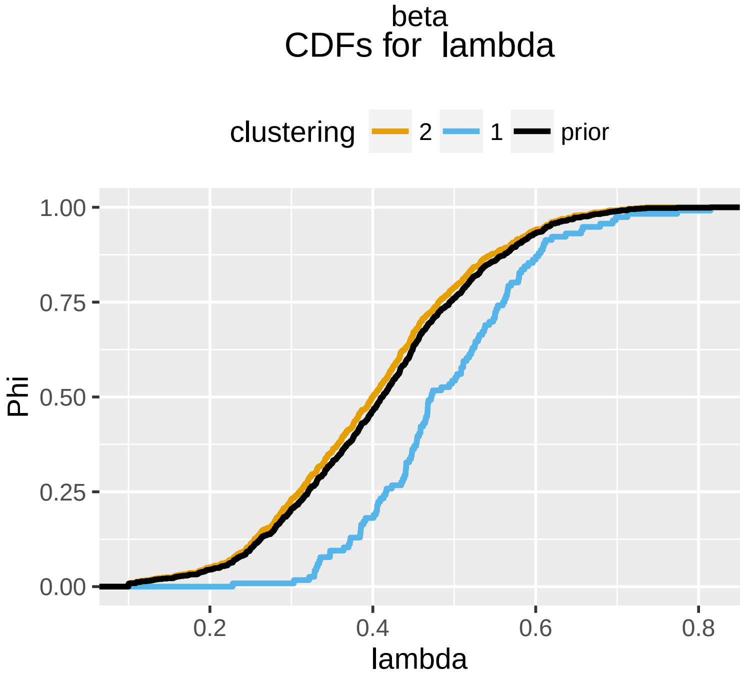


Figure 2: CDFs of single parameters

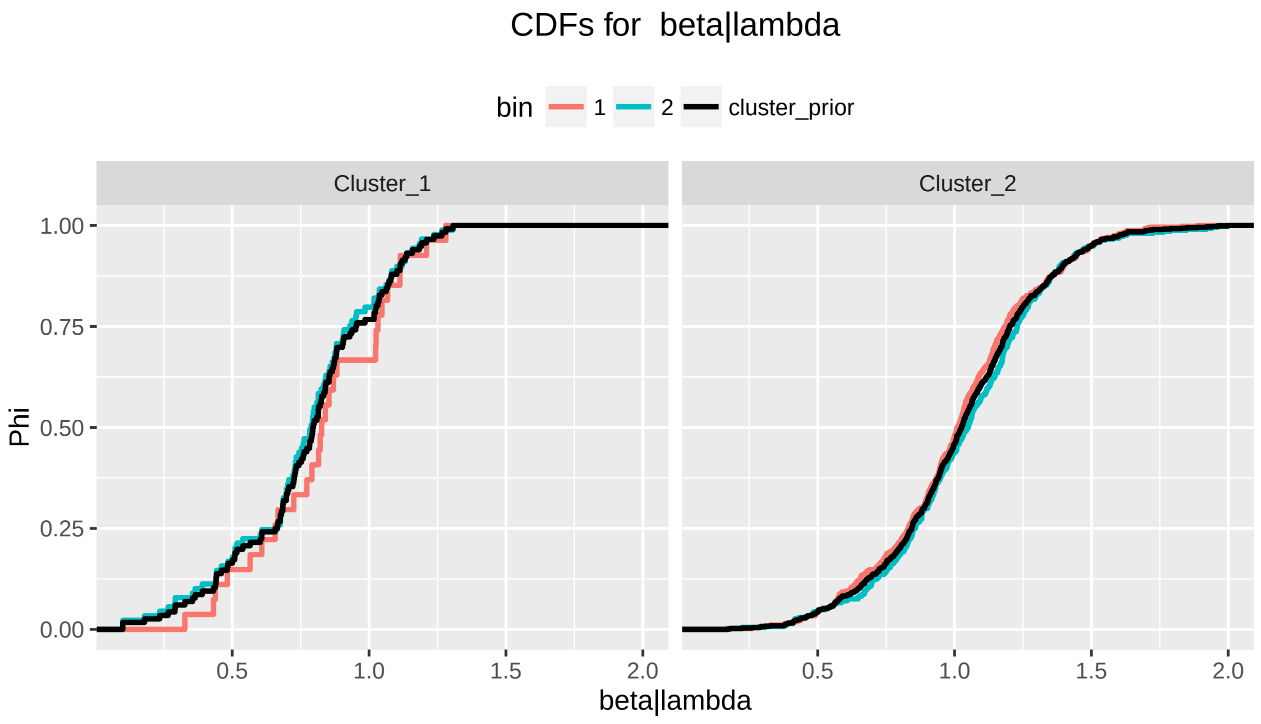


Figure 3: CDFs of interaction for 2 bins

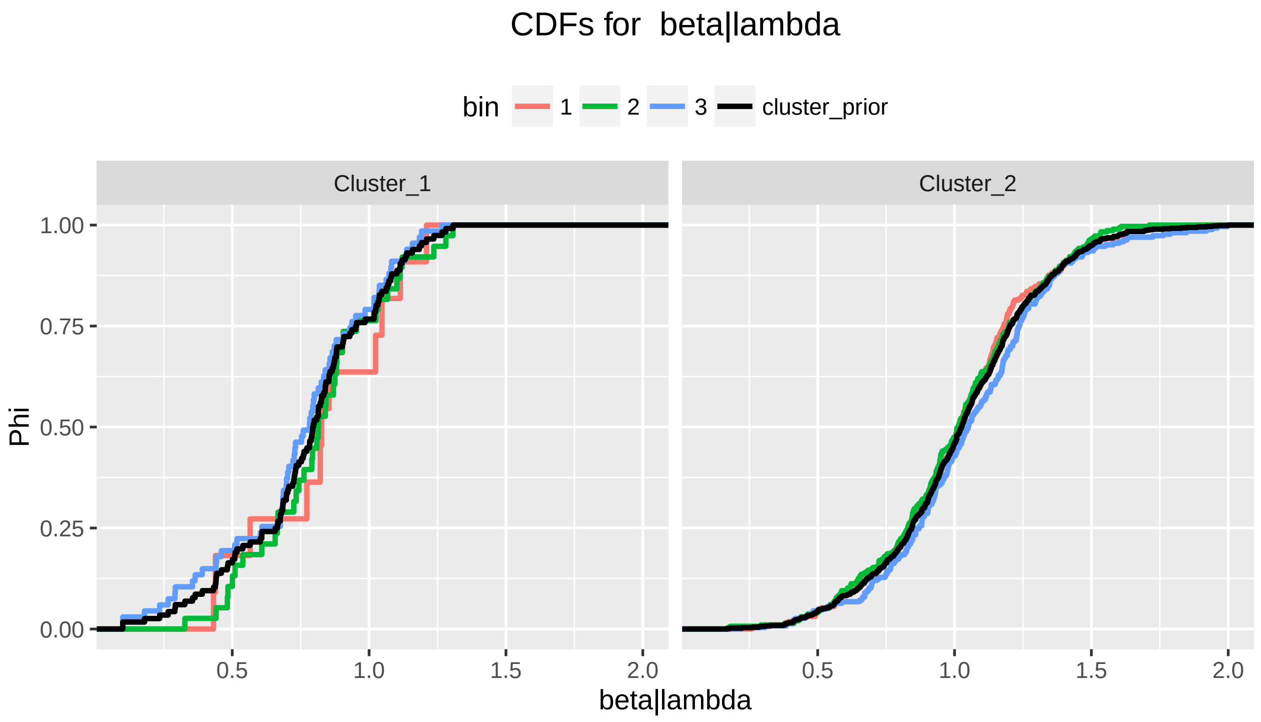


Figure 4: CDFs of interaction for 3 bins

whether to compute interactions or not. In the current version parameter (.parallel) is disabled but it is left as an option for future implementation to speed up bootstrapping through parallelization.

This function returns a list with two components. The first component is a 3D matrix of size (nbClusters X nbParameters X nbParameters). Diagnoal elements of this matrix are the main effects, while off-diagonal elements are sensitivities of interactions. All elements are automatically normalized/sandardized by .alpha value from the boostrapped distributions. If users want to change normalization factor the function given above has to be executed one more time, in other words the code does not save bootstrapped samples like the original MATLAB code on github.

To aid better visualization of the main effects and interactions we make use of the "corrplot" package which was originally developed for visualizations of covariance matrices. Without much effort we can gain significant insights into parameter interactions, perform parameter rankings through matrix rearrangement, and finally summarize the entire sensitivity analysis in one plot instead of using several paretto plots. The following wrapper function was developed for efficient communication with "corrplot" functions.

```r
plotMatrixDGSA(.dgsa, .hypothesis = TRUE, .method = "circle", ...)
```

Input parameter ".dgsa" is an object returned from the "dgsa" function (in our case "myDGSA"). The code will check if the object really came out of "myDGSA" function. The second parameter ".hypothesis" specifies whether to mark "non-sensitive" parameters on the correlation plot. User has a plethora of options that enable advanced markup of non-significant sensitivities (more on that later). The third parmeter is ".method" which specifies the plotting method passed to "corrplot" function. The choice of the method is limited by the "corrplot" capabilities. Finally, ". . . " parameters are all passed without change to "corrplot". Therefore, interested users should consult "corrplot" documentation (help(corrplot)) to learn more about advanced plotting capabilities.

```r
plotMatrixDGSA(myDGSA, .hypothesis = FALSE)
```

```r
plotMatrixDGSA(myDGSA, .hypothesis = TRUE)
```

One of the best features of corrplot package is matrix reordering. Many mehtods have been implemented, mainly for covariance matrix reordering, however the one that was found to work best in DGSA setting is "hieararchical clustering" with a "single" linkage. Results of this approach are given below. Notice that the parameters were ranked based on both their main effects and interactions. Particularly interesting part is the lower right corner which highlights that about 7 parameters pop up as the most important.

```r
plotMatrixDGSA(myDGSA, order = 'hclust', hclust.method='single', .hypothesis = FALSE)
```

If we turn ".hypothesis" parameter to TRUE the code will add "significance" marks to the matrix sensitivity plot to indicate which sensitivities were found to be insignificant in hypothesis testing. The following expample demonstrates such capability.

```r
plotMatrixDGSA(myDGSA, order = 'hclust', hclust.method='single', .hypothesis = TRUE)
```

A few more additional plots produced with advanced "corrplot" settings.

```r
plotMatrixDGSA(myDGSA, order = 'hclust', hclust.method='single', tl.srt = 65)
```

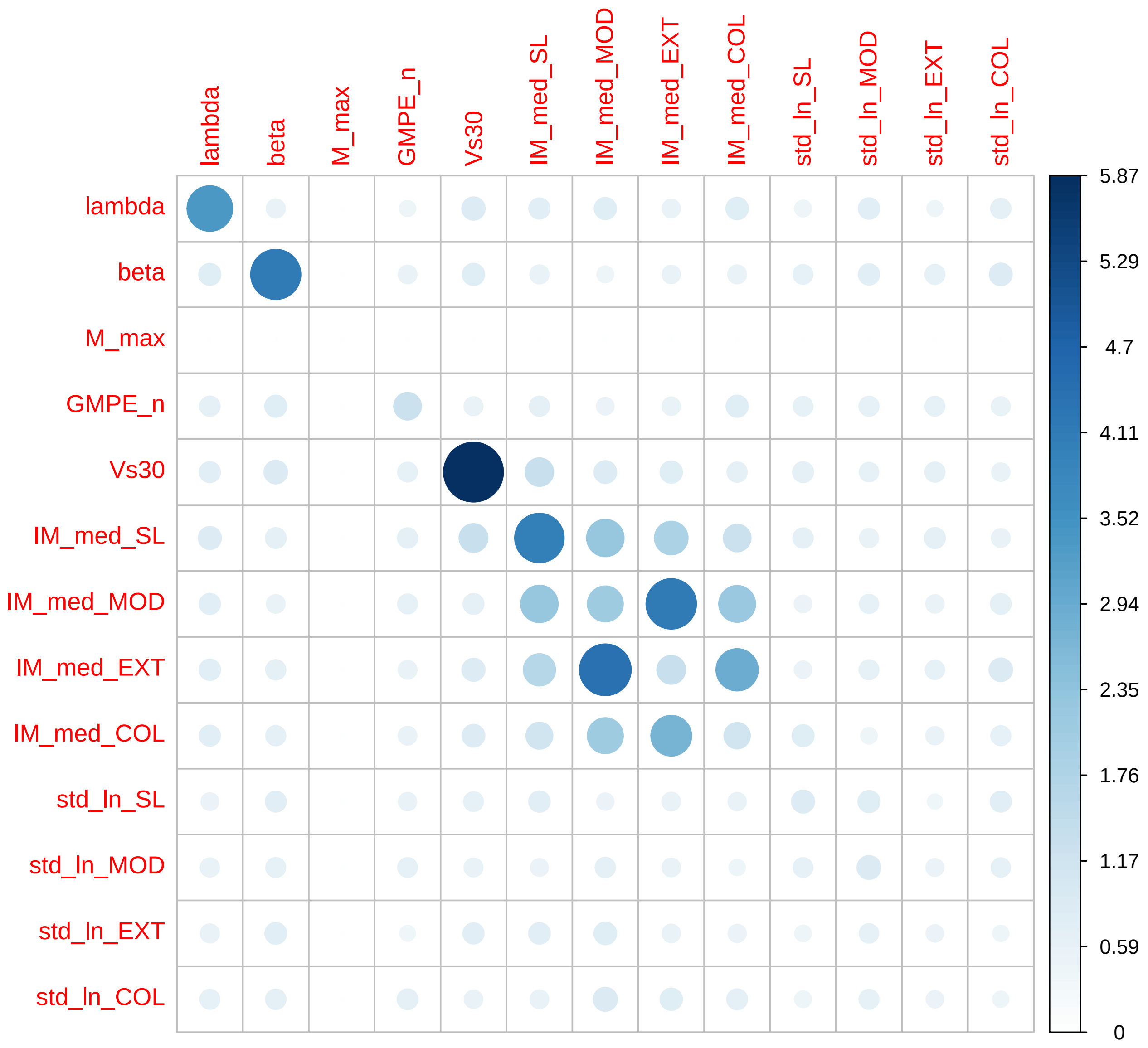


Figure 5: Main/Interaction plot without hypothesis testing

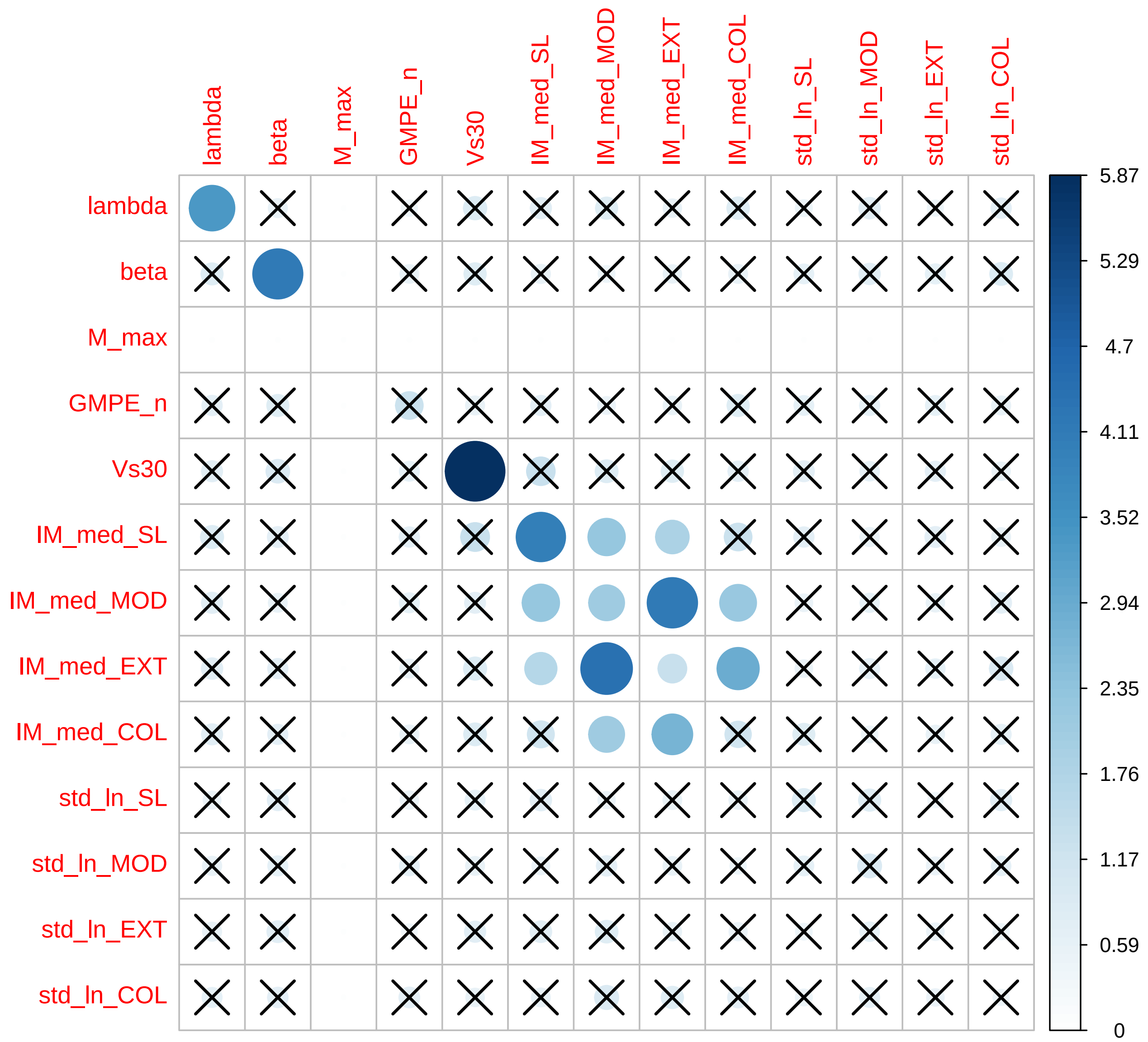


Figure 6: Main/Interaction plot with hypothesis testing

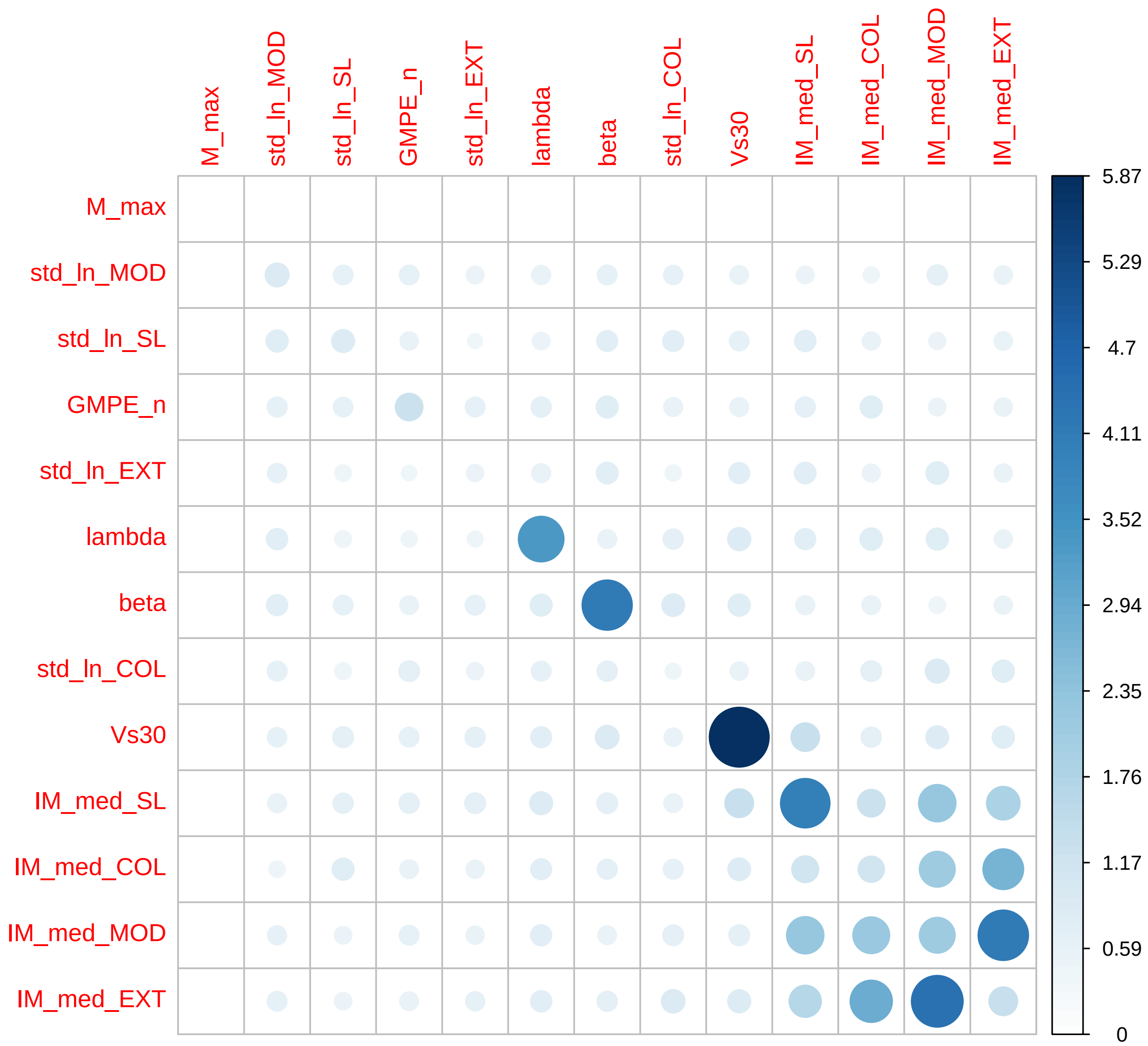


Figure 7: Main/Interaction plot with matrix reordering (hierarchical clustering method)

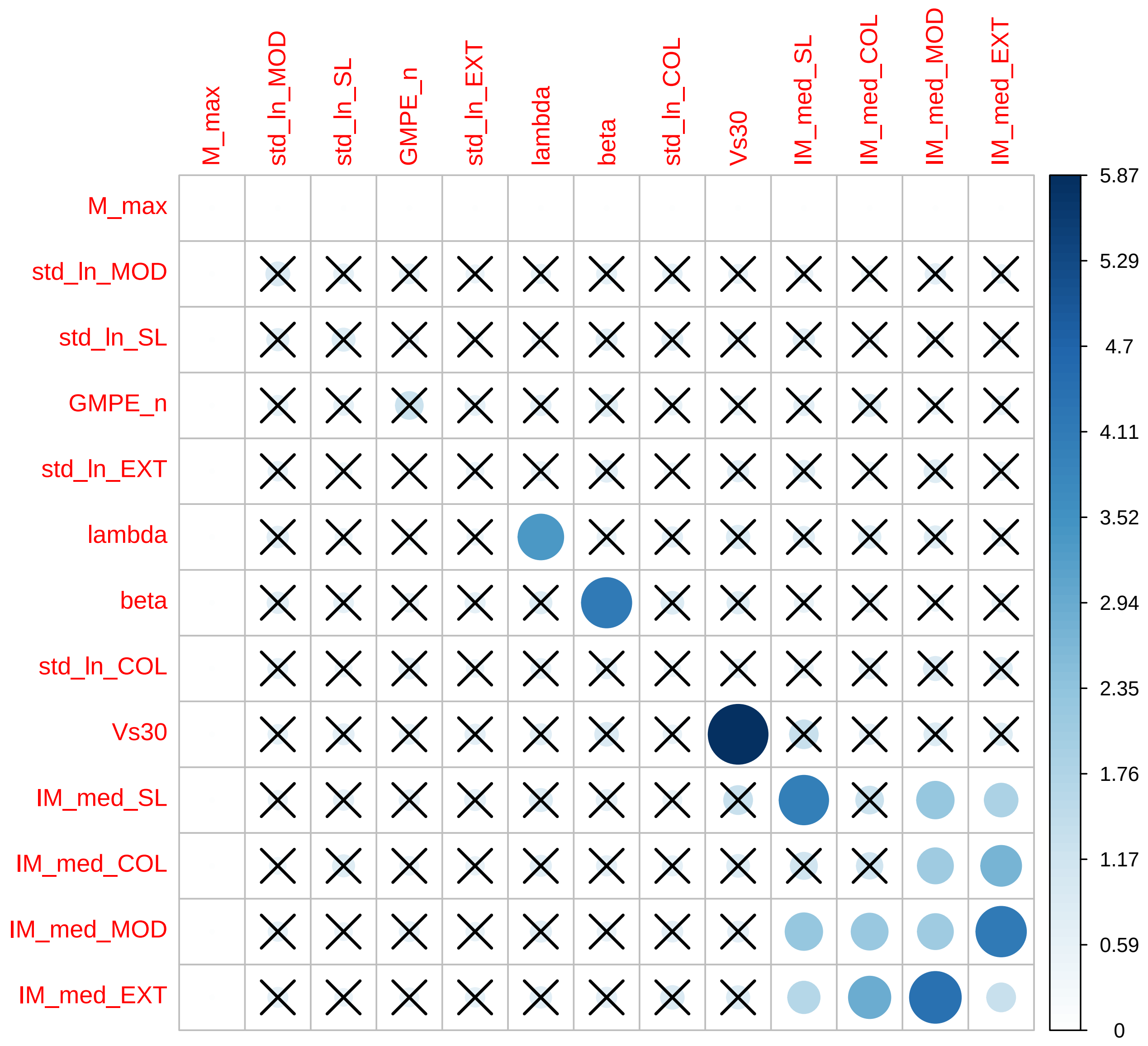


Figure 8: Main/Interaction plot with matrix reordering (hierarchical clustering method) + significance

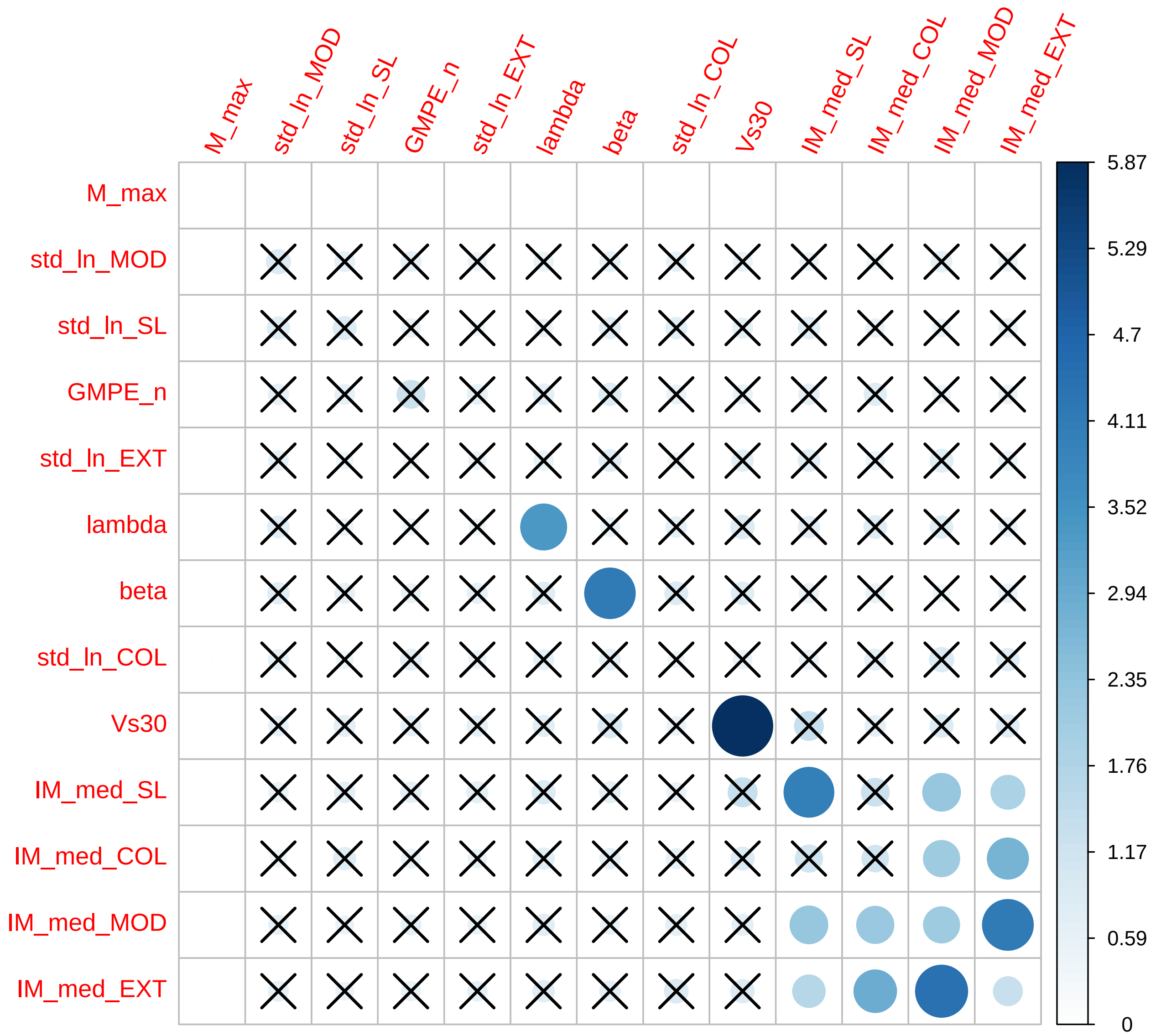
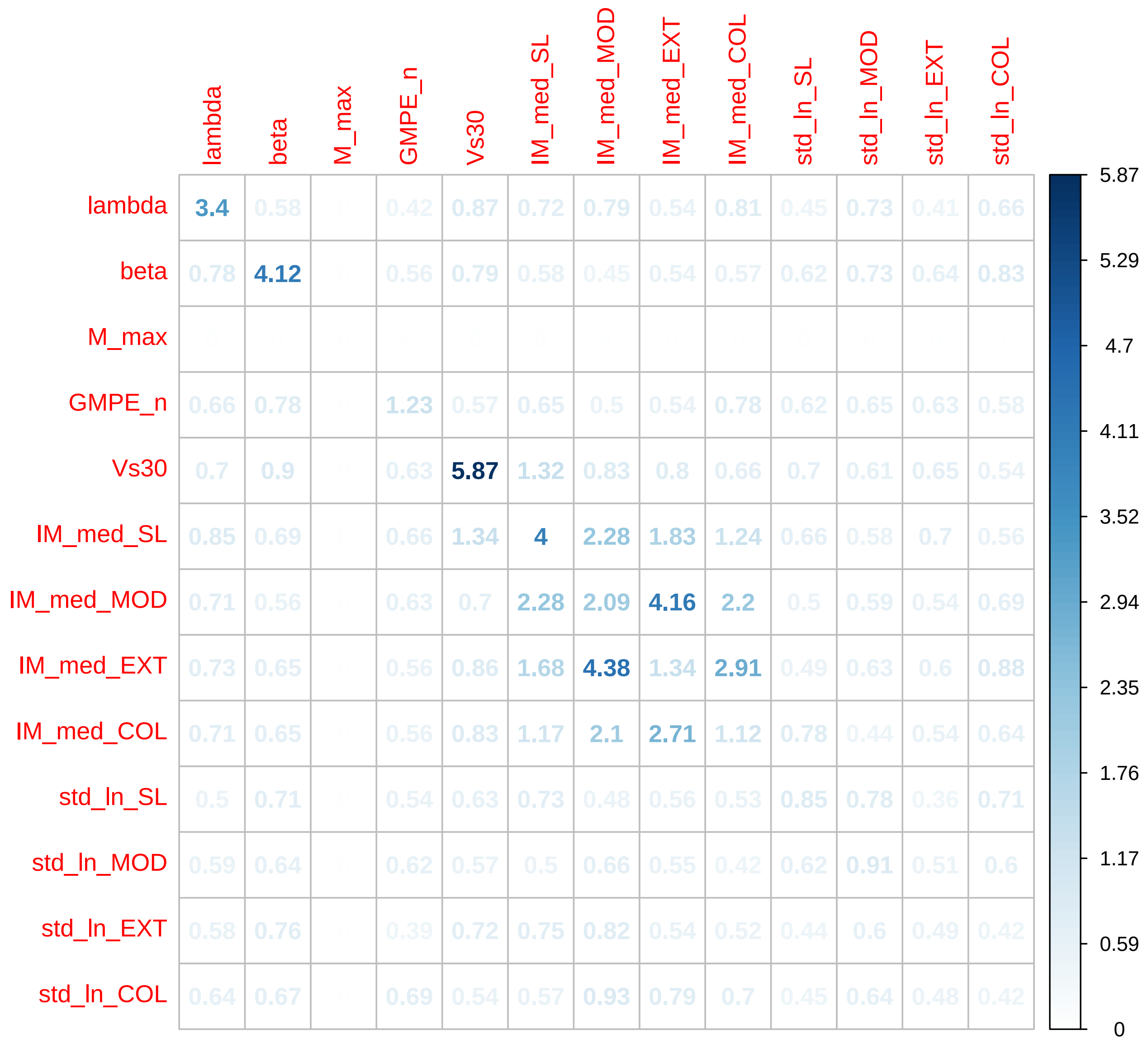


Figure 9: Main/Interaction plot with parameter reordering (hierarchical clustering method) with rotated text



	lambda	beta	M_max	GMPE_n	Vs30	IM_med_SL	IM_med_MOD	IM_med_EXT	IM_med_COL	std_ln_SL	std_ln_MOD	std_ln_EXT	std_ln_COL
lambda	3.4	0.58	0	0.42	0.87	0.72	0.79	0.54	0.81	0.45	0.73	0.41	0.66
beta	0.78	4.12	0	0.56	0.79	0.58	0.45	0.54	0.57	0.62	0.73	0.64	0.83
M_max	0	0	0	0	0	0	0	0	0	0	0	0	0
GMPE_n	0.66	0.78	0	1.23	0.57	0.65	0.5	0.54	0.78	0.62	0.65	0.63	0.58
Vs30	0.7	0.9	0	0.63	5.87	1.32	0.83	0.8	0.66	0.7	0.61	0.65	0.54
IM_med_SL	0.85	0.69	0	0.66	1.34	4	2.28	1.83	1.24	0.66	0.58	0.7	0.56
IM_med_MOD	0.71	0.56	0	0.63	0.7	2.28	2.09	4.16	2.2	0.5	0.59	0.54	0.69
IM_med_EXT	0.73	0.65	0	0.56	0.86	1.68	4.38	1.34	2.91	0.49	0.63	0.6	0.88
IM_med_COL	0.71	0.65	0	0.56	0.83	1.17	2.1	2.71	1.12	0.78	0.44	0.54	0.64
std_ln_SL	0.5	0.71	0	0.54	0.63	0.73	0.48	0.56	0.53	0.85	0.78	0.36	0.71
std_ln_MOD	0.59	0.64	0	0.62	0.57	0.5	0.66	0.55	0.42	0.62	0.91	0.51	0.6
std_ln_EXT	0.58	0.76	0	0.39	0.72	0.75	0.82	0.54	0.52	0.44	0.6	0.49	0.42
std_ln_COL	0.64	0.67	0	0.69	0.54	0.57	0.93	0.79	0.7	0.45	0.64	0.48	0.42

Figure 10: Main/Interaction plot "number"

```r
plotMatrixDGSA(myDGSA, .method = "number", .hypothesis = FALSE)
```

```r
plotMatrixDGSA(myDGSA, .method = "square", .hypothesis = FALSE)
```

```r
plotMatrixDGSA(myDGSA, .method = "shade", .hypothesis = FALSE)
```

```r
plotMatrixDGSA(myDGSA, .method = "pie", .hypothesis = FALSE)
```

```r
plotMatrixDGSA(myDGSA, .method = "pie", type = "lower", .hypothesis = FALSE)
```

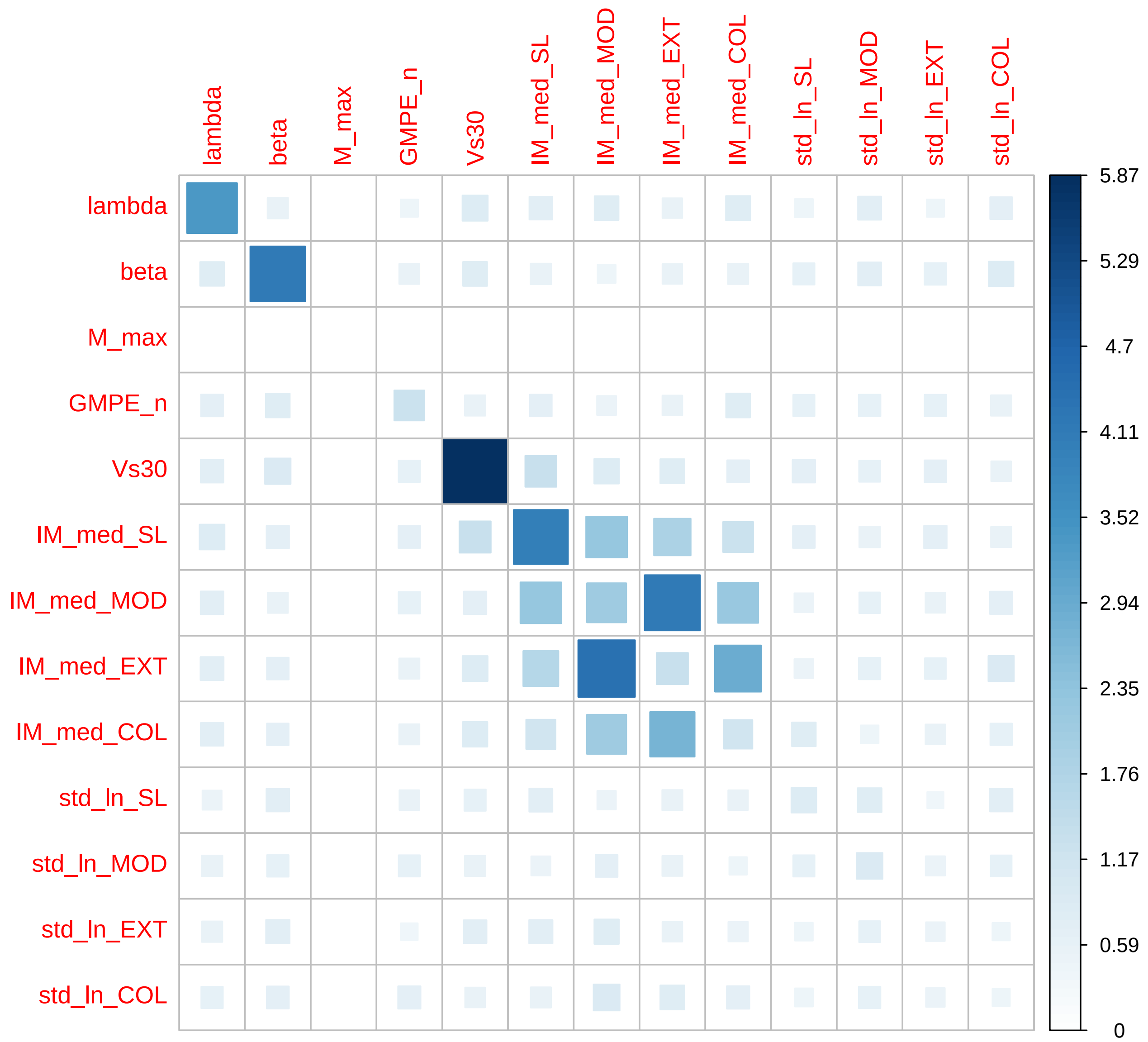


Figure 11: Main/Interaction plot "square"

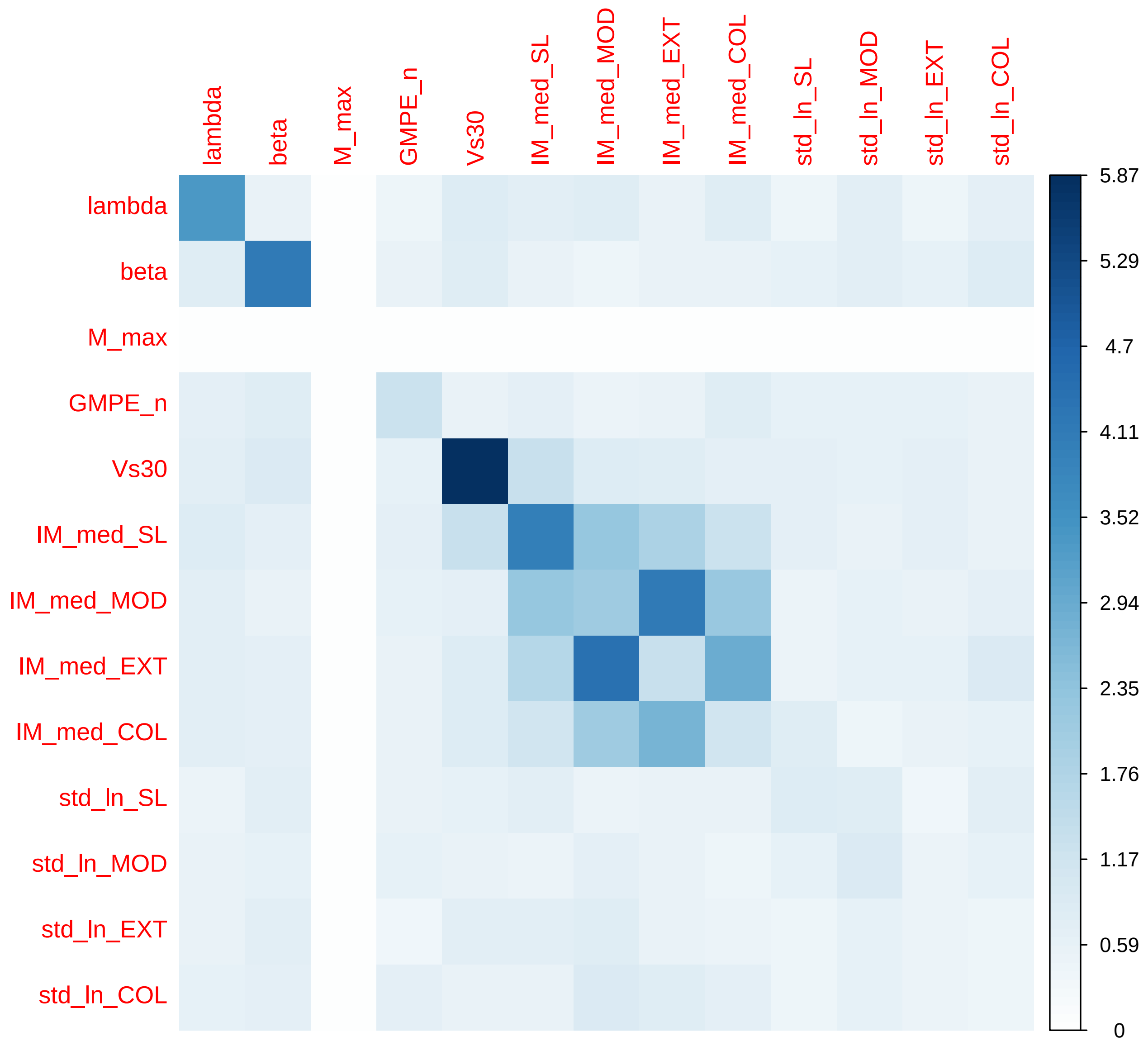


Figure 12: Main/Interaction plot "shade"

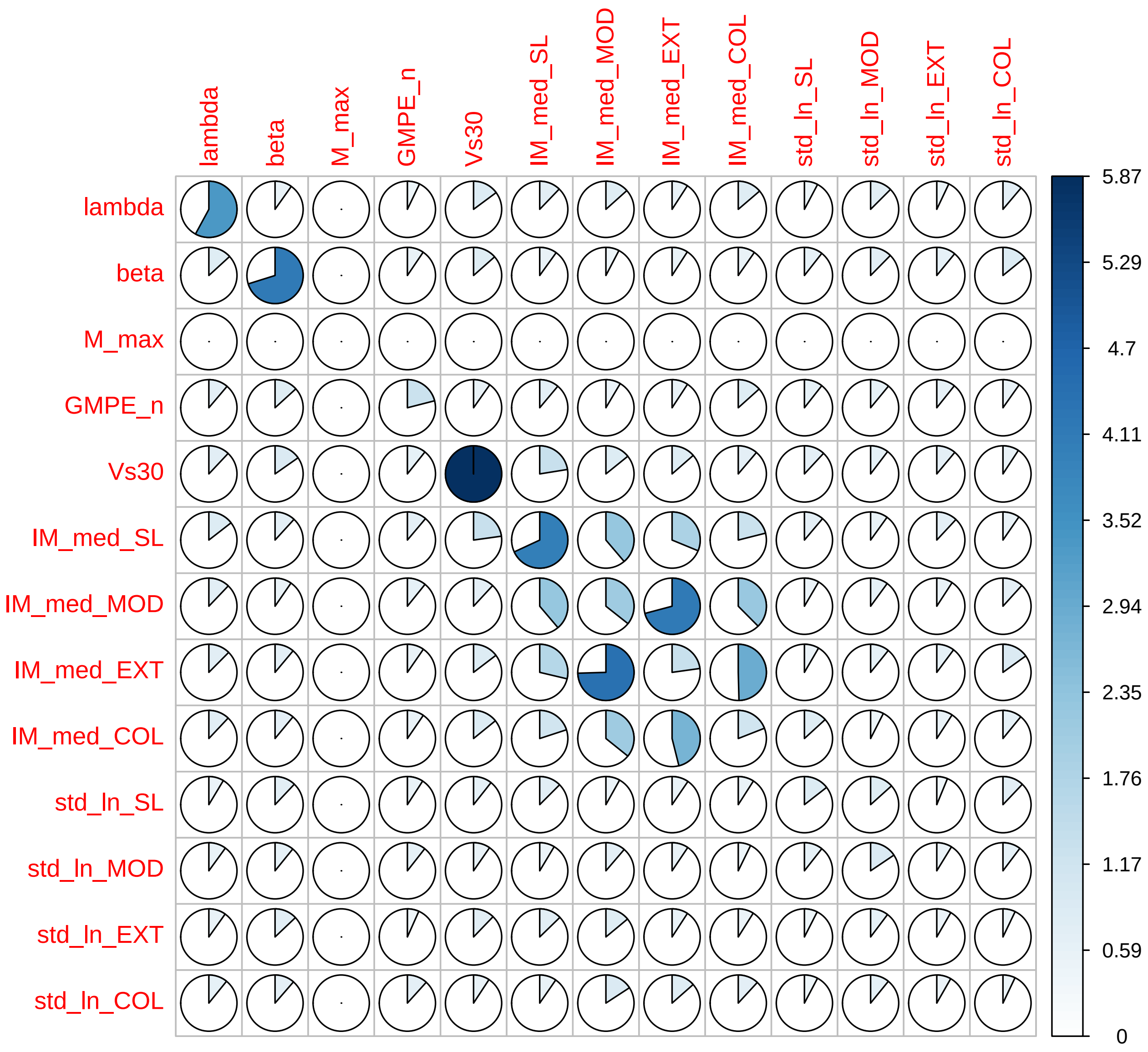


Figure 13: Main/Interaction plot "pie"

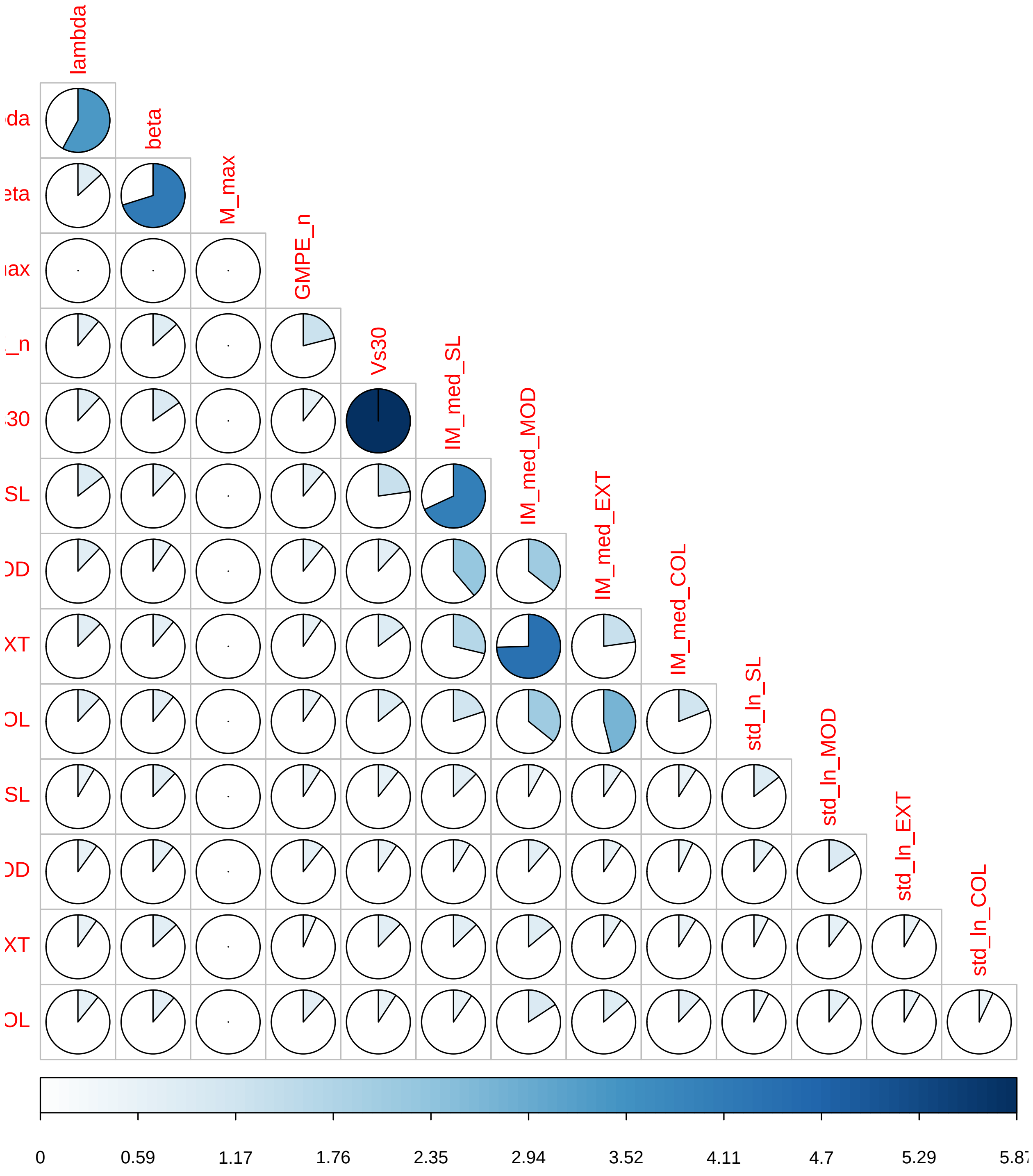


Figure 14: Main/Interaction plot "pie" just lower diagonal part

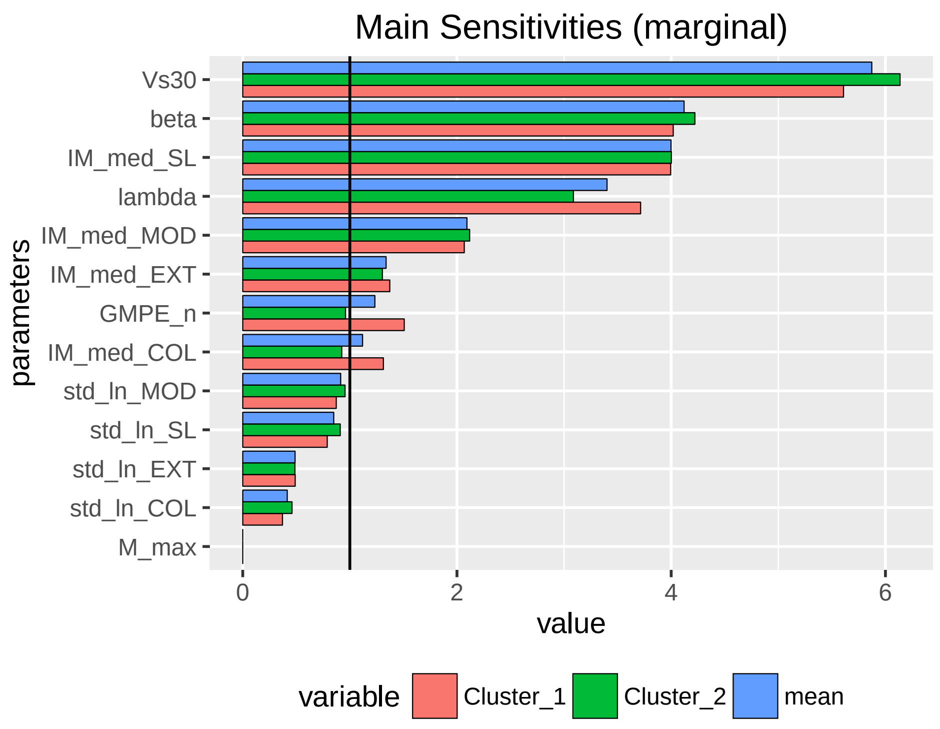


Figure 15: Pareto plot of main effects (ranked by mean)

```
plotMatrixDGSA(myDGSA, .method = "circle", diag=TRUE, .hypothesis = TRUE, tl.srt = 45,
               insig = "pch", pch = "+", pch.col = "black", pch.cex = 1.5)```
```

Pareto Plots:

DGSA package also has pareto plotting capabilities just like the original MATLAB code. For this feature we are using advanced graphics by "ggplot2" package. All wrapper functions also provide capability of returning an "ggplot" object for fine tunning of fonts/color via inline methods such as "g + ggtitle('mytitle')". etc. Users should consult ggplot2 documentation for more information.

```
plotParetoDGSA(myDGSA)
```

```
plotParetoDGSA(myDGSA, .interaction = "beta")
```

```
plotParetoDGSA(myDGSA, .interaction = "lambda")
```

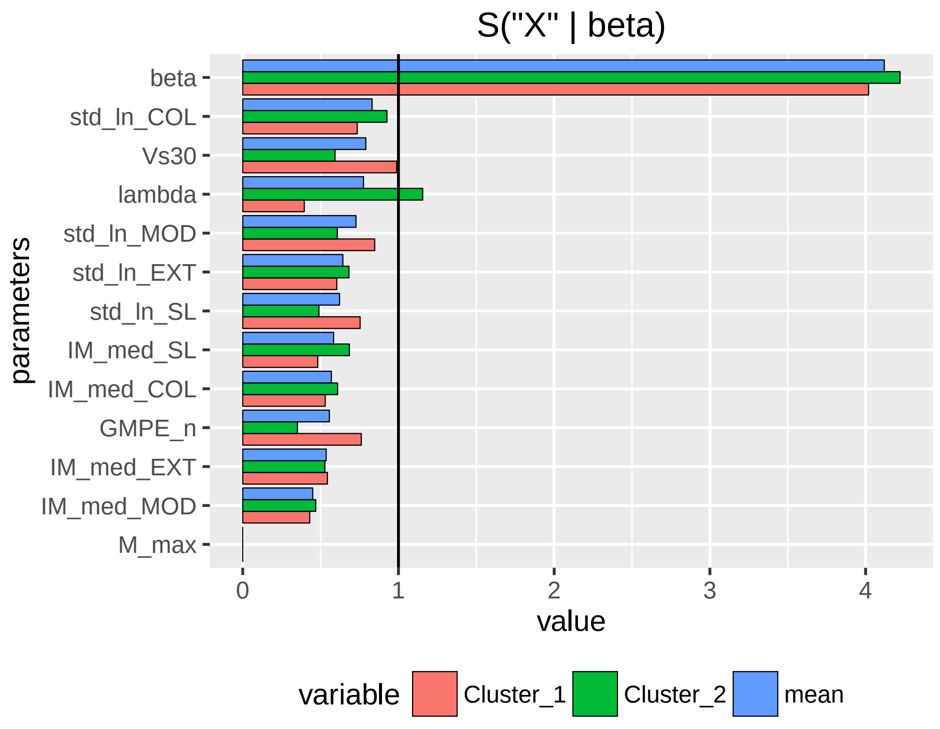


Figure 16: Pareto plot of parameter sensitivity given parameter beta (interactions). Note "beta|beta" is just its main effect

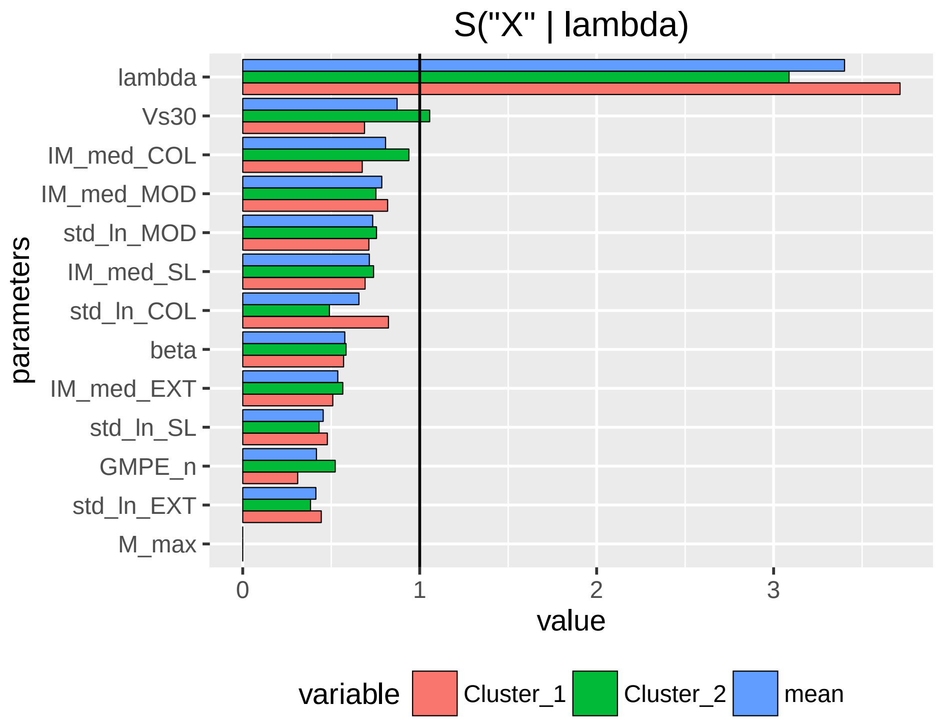


Figure 17: Pareto plot of parameter sensitivity given parameter lambda (interactions). Note "lambda|lambda" is just its main effect