MultiSEp: Predicting Gene Dependency Relationships (GDRs) from Multiomics Data

Adeline McKie, Mark Wappett & Ian Overton

July 2026

Contents

1. Introduction

The MultiSEp R package integrates 'omics data to predict Gene Dependency Relationships (GDRs) such as synthetic lethality or induced dependency. Functionality is also available to analyse and visualise the results.

MultiSEp (Multimodal Subsets of Expression) takes a matrix of continuous data in a gene by sample format (normally gene expression but possibly other gene-linked 'omics data such as DNA methylation), determines the optimal number of clusters for each gene, and produces cluster assignments for each sample. These clusters are integrated with different 'omics data types, such as gene effect score data from RNAi or CRISPR screens, or mutation calls from genome sequencing data to generate candidate GDRs. The MultiSEp package builds upon the web server SynLeGG (https://www.overton-lab.uk/synlegg/, Wappett *et al.* 2021), with significant additional functionality including new analytics for drug discovery applications with patient data.

This vignette demonstrates how MultiSEp may be applied to predict, analyse and visualise GDRs. We will follow the flow of the package structure outlined in Figure 1, below, which summarises key functions. Wrapper functions are shown with dashed lines in Figure 1 and will be considered in the 'Quickstart' section, below.

Some of the example data used in this package was provided by the Cancer Dependency Map 2024Q2 (https://doi.org/10.25452/figshare.plus.25880521.v1, Arafeh *et al.* 2025).

Citing MultiSEp

If you use MultiSEp in your work, please cite: AS McKie, M Wappett, H Vandierendonck, IM Overton (2026). The MultiSEp R package. https://cran.r-project.org/package=MultiSEp Please note: a preprint article will replace the above citation in the near future.

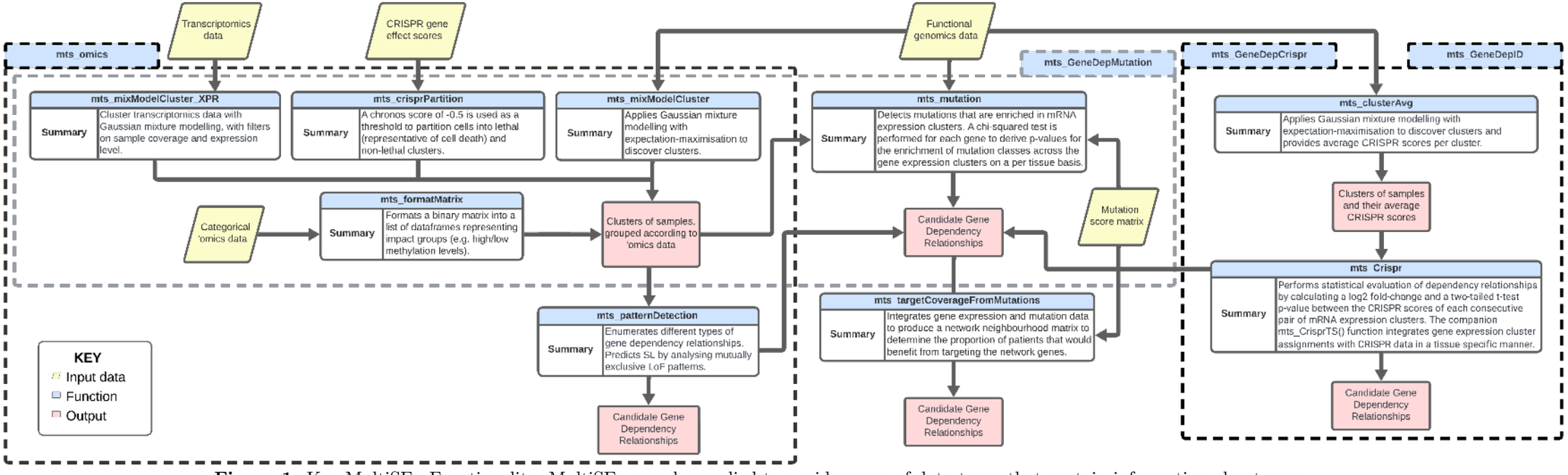

Figure 1: Key MultiSEp Functionality. MultiSEp may be applied to a wide range of data types that contain information about gene functional status in order to predict gene dependency relationships (GDRs), including Synthetic Lethality (SL). Input data (yellow) may be analysed through several different workflows. Dashed boxes outline the wrapper functions, which provide end-to-end pipelines for convenience. A central step is grouping samples to represent different gene function states (clusters); achieved by Gaussian mixture modelling, threshold-based partitioning or from data categories such as mutational status. Analysis of samples' distribution across the functional states for gene pairs reveals patterns that evidence GDRs. For example, SL gene pairs canonically have a mutually exclusive loss of function (LoF) pattern, corresponding to depletion of samples when both genes have LoF. The "Functional genomics data" input (top) could be many different data types (for example, transcriptomics, CRISPR, proteomics, methylation, mutations). Similarly, the "Categorical 'omics data" input (left-middle) could be multiple data types such as mutations, histone marks, methylation. Common use cases analyse transcriptomics (XPR), CRISPR with XPR, or mutation with XPR and functions are provided for these, such as *mts_mixModelCluster XPR()*.

2. Quickstart

Wrapper functions are available for key MultiSEp workflows, summarised below. Broadly, two analysis strategies are available for Gene Dependency Relationship (GDR) discovery. Both depend upon Gaussian mixture modelling (GMM). One approach is suitable for analysis of various samples (tissue, organoids or cell lines *etc.*) and deploys the binomial test to investigate GDRs in multiple data types (left-hand side, Figure 1). For example, by analysis of the GMM-derived gene expression clusters for two genes; analysing gene expression clusters with a categorical data type such as mutations or methylation status; and comparing categorical data types against themselves or each other (for example mutation vs mutation). A further approach was primarily developed for analysis of *in vitro* data such as from cell lines or organoids (SynLeGG methods) and a key application evaluates differences in CRISPR scores for one gene across GMM-derived expression clusters for a second gene; with statistical significance estimated using the t-test (right-hand side, Figure 1).

Predicting Synthetic Lethality (SL) and other gene dependencies with `mts_omics()`

mts_omics() can be applied to various 'omics data types to predict pairwise SL and other Gene Dependency Relationships (GDRs), including when only gene expression data is available (*e.g.* RNA-seq). Therefore, mts_omics() may be used for analysis of data from many different kinds of samples, such as tumour biopsies. Gaussian mixture modelling (GMM) defines clusters of samples where gene function is diminished, or lost (LoF); for example corresponding to the lowest expression values for a gene. Depletion of samples in the LoF clusters combined for two genes is characteristic of SL for most data types. An exception is that enrichment in the LoF clusters is expected for SL with CRISPR data, for example correlating CRISPR and gene expression (XPR). Additionally, mts_omics() can be used to discover arbitrary GDR patterns by setting *SyntheticLethalityPrediction=FALSE*. Examples of mts_omics() are given below for analysis of XPR data, as well as XPR with CRISPR data. When the same data-type is used, GDRs are symmetrical. However, GDRs arising from two data types have two orientations. For example, the analysis of gene A's XPR against gene B's CRISPR is a different orientation to analysing gene B's XPR against gene A's CRISPR. Both orientations are evaluated by default, if the data are available.

```
data("depMapXPR_subset")
SL_XPRvsXPR <- mts_omics(
  dataMatrix=depMapXPR_subset[1:5,],
  qVal=1, # 0.05 is recommended (default)
  cores=1) # increase if possible

SL_XPRvsCRISPR = mts_omics(
  dataMatrix = depMapCRISPRscores_subset[1:5,],
  dataMatrix2 = depMapXPR_subset[1:5,],
  directionality = "enrichment",
  effectsize = FALSE, # TRUE is recommended for CRISPR/XPR
  qVal = 1, # 0.05 is recommended (default)
  cores = 1 # increase if possible
)
```

Analysis with mts_omics() using categorical data, such as mutations and (continuous) gene expression, requires binary values for the categorical data type:

```
data("depMapMUT_small")
data("depMapXPR_small")

mutImpact <- as.data.frame( # convert to binary values
    ifelse(as.matrix(depMapMUT_small) == "WT", 2, 1),
    stringsAsFactors = FALSE, check.names = FALSE)
  rownames(mutImpact) <- rownames(depMapMUT_small)
```

```
SL_mut_vs_XPR <- mts_omics(
    dataMatrix   = mutImpact,
    dataMatrix2  = depMapXPR_small,
    categorical1 = TRUE, # mutation data is categorical
    directionality = "depletion",
    qVal         = 0.05,
    effectsize   = FALSE, # recommended for categorical data
    cores        = 1)
```

SL prediction with only categorical data is also possible, for example with mutations:

```
mutClusters <- mts_formatMatrix(matrix = mutImpact, cores = 1)

SL_mut_only = mts_omics(
  mixModelClusters1 = mutClusters,
  directionality    = "depletion",
  qVal              = 1,
  effectsize        = FALSE, # recommended for categorical data
  cores             = 1
  )
```

Predicting SL with CRISPR and gene expression (SynLeGG)

mts_GeneDepCrispr() runs SL discovery with Gaussian mixture modelling (GMM) and t-test; expected inputs are a gene expression matrix and a CRISPR (or RNAi) gene effect score matrix in gene by sample format. An earlier version of this function generated the 'CRISPR' results for the SynLeGG resource (https://www.overton-lab.uk/synlegg/, Wappett *et al.* 2021). Data should be continuous and in log2 format. We recommend running in Linux with multiple cores for timely production of results. The standard analysis with mts_GeneDepCrispr() predicts SL pairs. An example is given below:

```
data("depMapXPR_subset")
data("depMapCRISPRscores_subset")
mtsGeneDepResults <- mts_GeneDepCrispr(exprsMatrix=depMapXPR_subset[1:5,],
                                       crisprMatrix=depMapCRISPRscores_subset,
                                       cores=1, fcVal=-0.1, pVal=0.1)
```

Induced dependency prediction with CRISPR and gene expression (SynLeGG)

Predicted induced dependency pairs (synthetic dosage lethal relationships) are output from mts_GeneDepID(), which runs the induced dependency t-test pipeline using a gene expression matrix and a CRISPR gene effect score matrix in gene by sample format. An example is given below:

```
data("depMapXPR_subset")
data("depMapCRISPRscores_subset")
mtsGeneDepIDResults <- mts_GeneDepID(exprsMatrix=depMapXPR_subset[1:5,],
                                     crisprMatrix=depMapCRISPRscores_subset,
                                     cores=1, fcVal=0.1, pVal=0.1)
```

Predicting SL with mutation and gene expression (SynLeGG)

mts_GeneDepMutation() runs the chi-squared test for SL discovery using a gene expression matrix and a mutation summary matrix in gene by sample format. A previous version of this function generated results

in the 'Mutation' section of the SynLeGG resource (https://www.overton-lab.uk/synlegg/, Wappett *et al.* 2021). The output from mts_GeneDepMutation() is predicted SL gene pairs. Please see the example below:

```
data("depMapMUT_subset")
data("depMapTissue_subset")
mtsGeneDepMutResults <- mts_GeneDepMutation(exprsMatrix=depMapXPR_subset[1:5,],
                        tissueMatrix = depMapTissue_subset,
                        mutMatrix = depMapMUT_subset, pVal=0.1)
```

3. Prediction of Gene Dependency Relationships (GDRs) with Functional Genomics Data

The availability of large-scale 'omics data enables analysis of the consequences of gene loss of function (LoF) across many different genetic, transcriptional and disease backgrounds. These context-specific differences in the impact of gene LoF upon cell viability can reveal Gene Dependency Relationships (GDRs), including Synthetic Lethality (SL). Data from CRISPR and RNAi screens coupled with gene expression and other 'omics data across many different cell lines can sample the co-occurrence of gene pair LoF and so may be leveraged to discover depletion or enrichment patterns that are characteristic of SL or other GDRs. Tumours have endogenous perturbations, for example arising from mutations, which influence gene functional status. The variability in endogenous perturbations across different tumours may also be analysed to discover SL vulnerabilities by investigating mutually exclusive LoF patterns for pairs of genes.

Data partitioning to map gene function states

In order to discover mutually exclusive LoF patterns, MultiSEp requires a map of gene function states. This is relatively straightforward for categorical data, for example high impact mutations are expected to disrupt gene function. One of many possible mechanisms is where a mutation creates a premature STOP codon that leads to nonsense-mediated decay or a truncated protein, resulting in LoF due to the absence of the protein (or part of the protein) in the cell. On the other hand, more sophisticated methods are required to partition continuous data such as gene expression. The MultiSEp functionality for assigning gene function states using categorical and continuous data is outlined in the sections below.

Unsupervised clustering to reveal gene states for continuous data

MultiSEp runs Gaussian mixture modelling (GMM) with Expectation-Maximisation to discover clusters (sometimes called modes), that represent different gene function states. Cardinality (#clusters) is determined by Bayesian Information Criterion regularisation (Lubbock *et al.* 2013). A typical use case derives clusters of samples from gene expression (XPR) data.

The following example demonstrates GMM for gene expression data with mts_mixModelCluster_XPR() to derive cluster assignments for genes based on their expression patterns across the samples. The output of mts_mixModelCluster_XPR() is a list of data frames which correspond to a gene (row name) from the input gene expression object. The 'cores' argument may be increased to enable multithreading, resulting in faster execution time (not available in MS Windows at present). In addition to GMM, mts_mixModelCluster_XPR() applies filters that ensure the genes taken forwards for analysis have sufficient data to produce meaningful results. In particular, these filters ensure that the gene is expressed above background levels in a sufficient number of samples:

```
data("depMapXPR_subset")
ExpressionClusters <- mts_mixModelCluster_XPR(dataMatrix = depMapXPR_subset[1:5,],
                          GeneXPRthresh = 3.321928, # equal to log2(10) (default)
                          NumSampleThresh = 20)  # (default)
```

The ExpressionClusters object (above) is a list of dataframes where each gene corresponds to a separate

dataframe containing the samples, sample values (expression) and cluster assignment:

```
knitr::kable(head(ExpressionClusters[[1]], 10))
```

Sample	Values	Cluster_Assignment
NIHOVCAR3	0.0000000	1
HL60	0.1634987	2
CACO2	0.0703893	1
HEL	6.9182670	4
HEL9217	7.1325768	4
MONOMAC6	0.0000000	1
LS513	4.9307373	4
A101D	0.0976108	2
C2BBE1	0.0426443	1
NCIH2077	0.0000000	1

```
names(ExpressionClusters)
#> [1] "EIF1AY"  "RPP25"   "DNAJC15" "NMT2"    "STX2"
```

A data-agnostic function is available, mts_mixModelCluster(); which performs GMM without the above filters. Quality metrics vary between data types and should be considered before running this function. A value of at least 20 samples with values above the background value is recommended. The input is a data matrix and therefore many different data types may be analysed; such as drug sensitivity, transcriptomics, proteomics, CRISPR, methylation etc. The output of mts_mixModelCluster() is a list of dataframes, as shown above.

Threshold-based partitioning of continuous data

Data may also be partitioned using a fixed threshold value. The mts_crisprPartition() function is designed to separate scores (e.g. from Chronos) into two classes that approximate to (1) 'cell death' or (2) 'little or no cell death'. This function could also be applied to partition other data types into classes representing different gene functional states or responses, assuming that a reasoned threshold value can be chosen. This approach is motivated by the observation that unsupervised clustering may sometimes define cluster boundaries that do not align well with functional states of interest - because cluster assignment is driven by the distribution of data points. For example, CRISPR disruption of some genes may kill most cell lines (or leave most cell lines unaffected). In these cases, GMM might define cluster boundaries that do not distinguish between samples that undergo elevated cell death and those that are relatively unaffected by the gene LoF. Accordingly, setting a cluster boundary based upon prior knowledge can be beneficial. The t-test (SynLeGG) methods such as mts_GeneDepCrispr() do not require boundaries to be set in the CRISPR data, which could have advantages especially when there are small sample sizes. The output of mts_crisprPartition() is a list of dataframes, as shown in the section above.

Partitioning categorical data

Categorical data, for example mutational status, does not typically require clustering. The mts_formatMatrix() function may be used for preprocessing if the input is a binary matrix. For example a matrix representing mutation or methylation data may encode gene functional status with numerical values. A value of 1 within the input binary matrix is assigned by the mts_formatMatrix() to 'HIGH' impact (loss of function) and a value of 2 is 'LOW' impact. The output follows a similar format to that produced by mts_mixModelCluster_XPR() where each row name (e.g. gene) corresponds to a separate dataframe that contains the impact groups for the samples evaluated. The following example predicts pairwise genetic dependencies using a mutational matrix formatted by mts_formatMatrix():

```
data("MUT_impactMatrix")
Formatted_MUTimpactMatrix <- mts_formatMatrix(matrix = MUT_impactMatrix, cores = 1)
mutation_vs_XPR_SL <- mts_patternDetection(
                                  mixModelClusters1 = Formatted_MUTimpactMatrix,
                                  mixModelClusters2 = ExpressionClusters,
                                  effectsize = TRUE,
                                  effectsize_threshold = 0, # not recommended, just for
                                  ↪  this example
                                  directionality = "depletion",
                                  qVal = 1) # 0.05 is recommended and the default
```

Analysing canonical dependency patterns for gene pairs

The mts_patternDetection() function can identify many different GDR patterns by evaluation of sample depletion (or enrichment) in pairwise combinations of gene clusters - which form cells in a contingency table. The mts_omics() function is a wrapper for mts_patternDetection() and therefore may provide convenient access to the same functionality. Statistical significance is estimated with the binomial test. Depletion of samples in the bottom-left table cell is evaluated by default; where both genes are assumed to have loss of, or low, function. However, all of the contingency table cells may be evaluated for depletion or enrichment patterns. The example below predicts synthetic lethality (SL) for gene expression data by evaluating the canonical mutually exclusive LoF pattern:

```
data(mixModelClusters_vignette) # precomputed for speed
mixturemodelClusters = mixModelClusters_vignette
XPRvsXPR_SL <- mts_patternDetection(mixModelClusters1=mixturemodelClusters,
                                                    directionality = "depletion",
                                                    p_adjustMethod = "BY",
                                                    qVal = 0.01, effectsize = TRUE,
                                                    SyntheticLethalityPrediction = TRUE)
```

The output from mts_patternDetection() characterises the table cell evaluated ('Cluster_Combination'); including the actual and expected count of samples, as well as the statistical significance of the difference in the counts (q-value). The total number of cells in the contingency table is also given in the 'TotalCluster_Combinations' column:

```
XPRvsXPR_SL %>%
  arrange(q_value) %>%
  arrange(Actual_Count) %>%
  kable(row.names = FALSE, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results" = 12))
```

Results											
Gene1	Gene2	Gene1_ClusterData	Gene2_ClusterData	Actual_Count	Expected_Count	TotalCluster_Combinations	Cluster_Combination	Sample_Count	p_value	Effect_Size	q_value
PSMB8	TTC7B	1	1	0	9.241564	9	1_x_1	1304	9.38e-05	0.9988526	0.0035237
PSMB8	TUBA1A	1	1	19	42.708589	6	1_x_1	1304	3.09e-05	0.9893465	0.0023247
DNAJC15	TUBA1A	1	1	49	79.941718	6	1_x_1	1304	8.93e-05	0.9762825	0.0035237

The pattern of samples for a gene pair can be visualised with mts_plotClusterDistribution(). The resulting 'MultiSEp plot' shows boundaries between the expression clusters; the clusters for gene1 and gene2 are respectively shown on the x-axis and y-axis. Figure 2 shows depletion of samples in the bottom-left contingency table cell for PSMB8 and TTC7B, consistent with an SL relationship:

```
mts_plotClusterDistribution(mixModelClusters = mixturemodelClusters,
                                gene1 = "PSMB8", gene2 = "TTC7B")
```

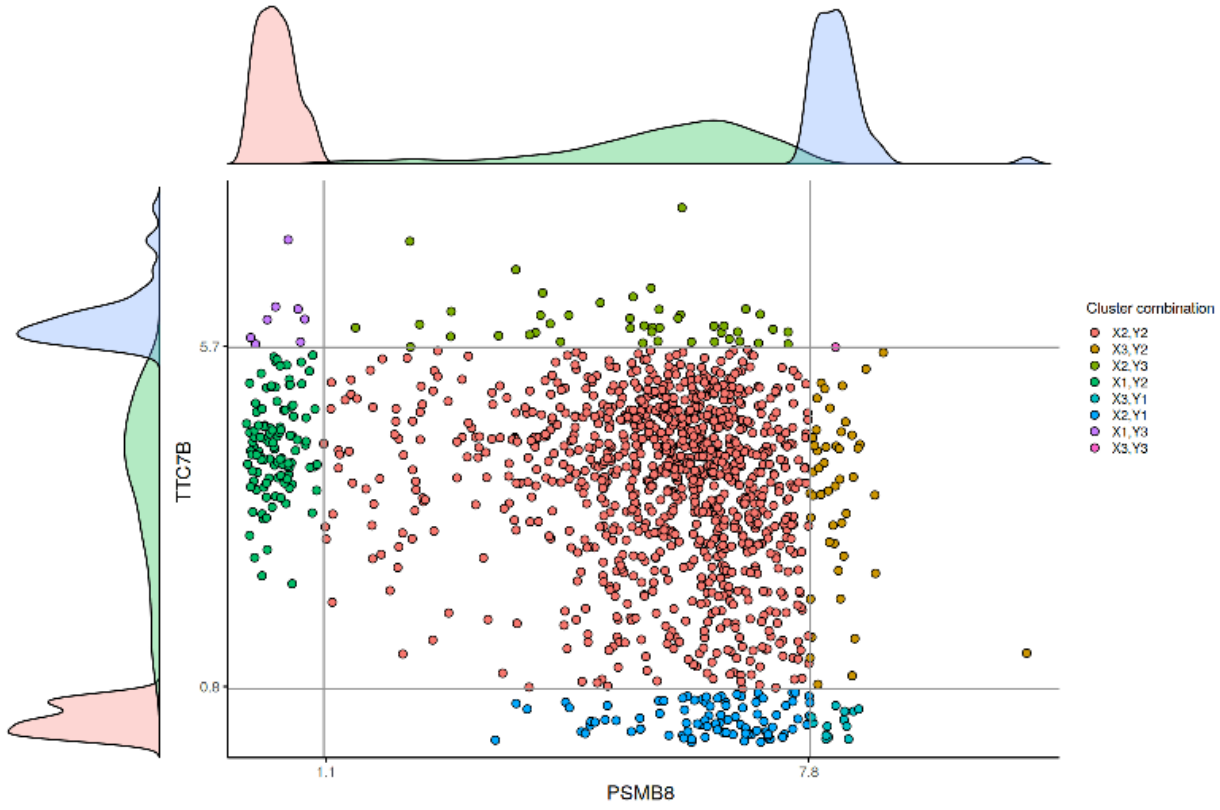

Figure 2: MultiSEp plot of PSMB8 and TTC7B expression. Clusters from Gaussian mixture modelling of the gene expression data are shown (top for PSMB8, left for TTC7B) and the cluster boundaries form a contingency table. Each circle in the table represents a sample, coloured by the pairwise cluster combination. The bottom-left table cell corresponds to low expression of both genes, and may be assumed to represent low or loss of function. The absence of samples in the bottom-left cell is consistent with an SL relationship between TTC7B and PSMB8.

The mts_patternDetection() function may also be applied to evaluate all of the cluster combinations in the contingency table (i.e. across all table cells), by altering the SyntheticLethalityPrediction flag:

```
GDRdepletionPattern <- mts_patternDetection(mixModelClusters1=mixturemodelClusters,
                                    directionality = "depletion",
                                    qVal=0.01,
                                    SyntheticLethalityPrediction = FALSE)
```

Results are shown on the next page:

```
GDRdepletionPattern %>%
  arrange(Actual_Count / Expected_Count) %>%
  kable(row.names = FALSE, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results" = 12))
```

Results

Gene1	Gene2	Gene1_ClusterData	Gene2_ClusterData	Actual_Count	Expected_Count	TotalCluster_Combinations	Cluster_Combination	Sample_Count	p_value	Effect_Size	q_value
SNAP25	TUBA1A	4	2	4	42.34356	8	4_x_1	1304	0.00e+00	0.9964971	0.0000000
SNAP25	TTC7B	4	3	6	31.12117	12	3_x_1	1304	0.00e+00	0.9954694	0.0000095
CCDC88A	TUBA1A	3	2	14	48.54908	6	3_x_1	1304	0.00e+00	0.9916333	0.0000010
CCDC88A	TUBA1A	3	2	19	46.35276	6	1_x_2	1304	3.40e-06	0.9893465	0.0006546
ATP1B1	MYC	2	2	49	111.26457	4	2_x_2	1304	0.00e+00	0.9762825	0.0000000
PSMB8	TUBA1A	3	2	19	42.70859	6	1_x_1	1304	3.09e-05	0.9893465	0.0053405
SNAP25	TUBA1A	4	2	72	143.82209	8	3_x_1	1304	0.00e+00	0.9666143	0.0000000

For example, SNAP25 and TUBA1A have depletion of samples in the bottom right table cell (4x1 Cluster_Combination), shown in Figure 3:

```
mts_plotClusterDistribution(mixModelClusters = mixturemodelClusters,
                            gene1 = "SNAP25", gene2 = "TUBA1A")
```

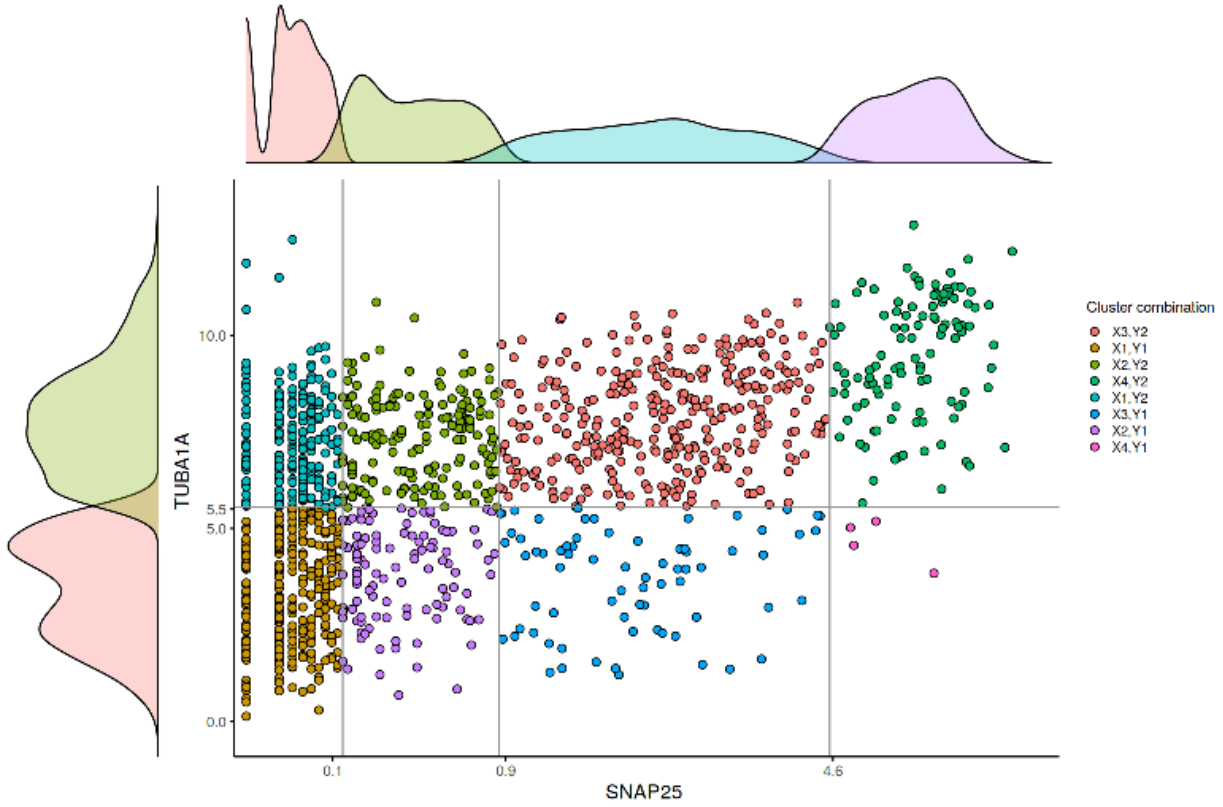

Figure 3: MultiSEp plot of SNAP25 and TUBA1A expression. Depletion of samples in the bottom-right cell was identified by *mts_patternDetection()*. Clusters from Gaussian mixture modelling of the gene expression data are shown (top for SNAP25, left for TUBA1A) and the cluster boundaries form a contingency table. Each circle in the table represents a sample, coloured by the pairwise cluster combination.

Mapping GDRs with sample enrichment: CRISPR data

We expect synthetic lethal (SL) relationships to involve enrichment of samples in the bottom left contingency table cell if we are analysing CRISPR data, as long as the second data type (e.g. gene expression) codes loss of function (LoF) with low values. As before, the patterns formed by the samples may also be explored across all cluster combinations in the contingency table by setting the SyntheticLethalityPrediction argument to FALSE. The code on the next page analyses CRISPR and gene expression data:

```
 CRISPR_clusters = mts_crisprPartition(dataMatrix =
↪  depMapCRISPRscores_subset[c(3,9,17),])
CRISPR_XPR_enrichment_SL <- mts_patternDetection(
                                    mixModelClusters1=mixturemodelClusters,
                                    mixModelClusters2 = CRISPR_clusters,
                                    p_adjustMethod = "BH",
                                    qVal = 0.01, effectsize = FALSE, #  effectsize =
                                    ↪  TRUE is generally recommended, is set to
                                    ↪  FALSE just for this example
                                    directionality = "enrichment",
                                    SyntheticLethalityPrediction = TRUE)

CRISPR_XPR_enrichment_GDRs <- mts_patternDetection(
                                   mixModelClusters1=mixturemodelClusters,
                                   mixModelClusters2 = CRISPR_clusters,
                                   p_adjustMethod = "BH",
                                   qVal = 0.01, effectsize = FALSE,  #  effectsize =
                                   ↪  TRUE is generally recommended, is set to FALSE
                                   ↪  just for this example
                                   directionality = "enrichment",
                                   SyntheticLethalityPrediction = FALSE)
```

Results are shown below:

```
CRISPR_XPR_enrichment_SL %>%
  arrange(q_value) %>%
  kable(row.names = FALSE, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results" = 11))
```

Results										
Gene1	Gene2	Gene1_ClusterData	Gene2_ClusterData	Actual_Count	Expected_Count	TotalCluster_Combinations	Cluster_Combination	Sample_Count	p_value	q_value
ATP1B1	ATP1B3	1	1	109	28.74777	4	1_x_1	896	0	0
DNAJC15	DNAJC19	1	1	109	34.14621	6	1_x_1	896	0	0

For example, there is a significant enrichment of samples in the bottom-left contingency table cell for DNAJC19 CRISPR scores and DNAJC15 gene expression; this pattern is consistent with a synthetic lethal relationship (Figure 4).

```
mts_plotClusterDistribution(mixModelClusters = mixturemodelClusters,
                            mixModelClusters2 = CRISPR_clusters,
                            gene1 = "DNAJC15", gene2 = "DNAJC19",
                            gene1_datatype="log2 gene expression",
                            gene2_datatype = "CRISPR gene effect score")
```

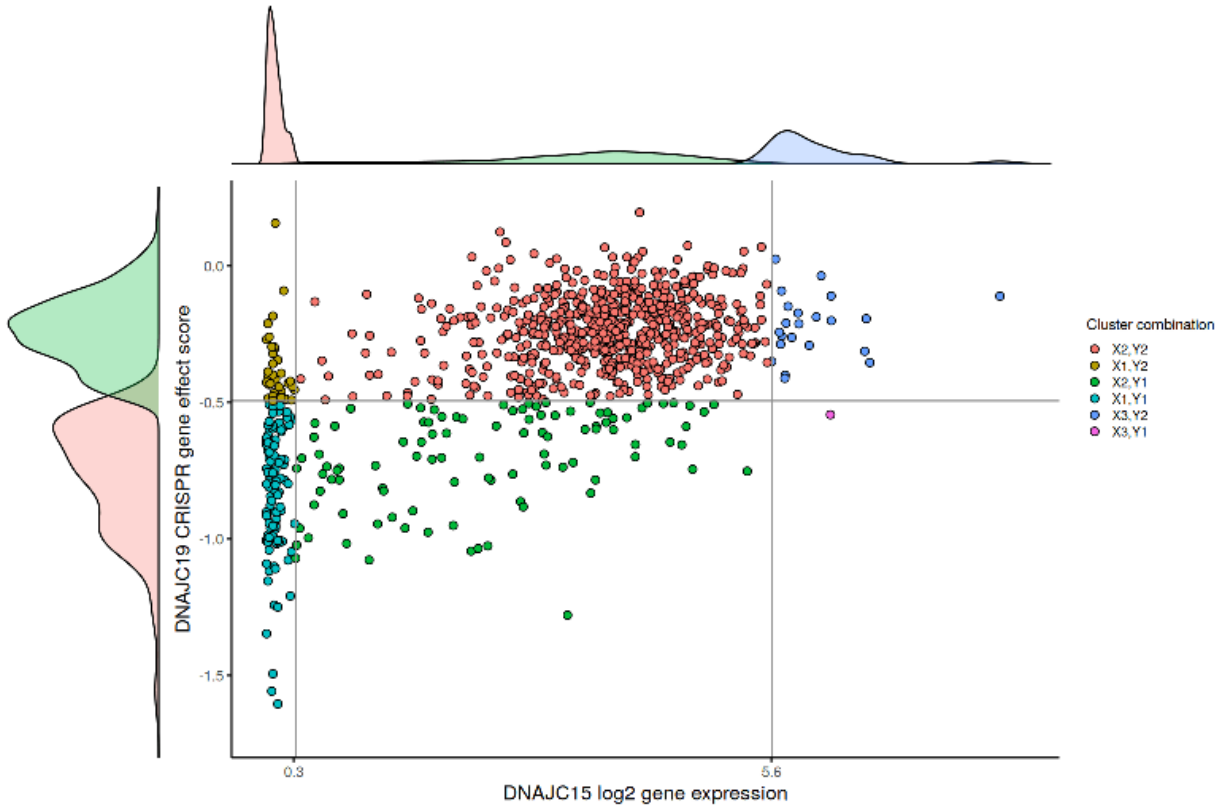

Figure 4: MultiSEp plot of DNAJC19 CRISPR scores and DNAJC15 gene expression. Clusters from Gaussian mixture modelling of the gene expression data are shown at the top for DNAJC15, and the results of partitioning the DNAJC19 CRISPR scores at a value of -0.5 are shown in the density on the left-hand side of the Figure. The cluster boundaries form a contingency table and each circle in the table represents a sample, coloured by the pairwise cluster combination. Enrichment of samples in the bottom-left contingency table cell is consistent with an SL relationship.

A table of GDRs for gene expression with CRISPR effect scores is shown below:

```
CRISPR_XPR_enrichment_GDRs %>%
  arrange(q_value) %>%
  kable(row.names = FALSE, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results" = 11))
```

Results										
Gene1	Gene2	Gene1_ClusterData	Gene2_ClusterData	Actual_Count	Expected_Count	TotalCluster_Combinations	Cluster_Combination	Sample_Count	p_value	q_value
ATP1B1	ATP1B3	2	2	109	28.74777	4	1_x_1	896	0.0e+00	0.00e+00
DNAJC15	DNAJC19	3	2	109	34.14621	6	1_x_1	896	0.0e+00	0.00e+00
ATP1B1	ATP1B3	2	2	684	603.74777	4	2_x_2	896	0.0e+00	0.00e+00
DNAJC15	DNAJC19	3	2	627	556.56250	6	2_x_2	896	5.0e-07	6.50e-06
ATP1B1	PSMB5	2	2	31	11.88951	4	1_x_2	896	2.3e-06	2.39e-05

For example ATP1B1 and ATP1B3 have significant enrichment in the 1_x_1 table cell (death occurs in cells with ATP1B3 loss and low ATP1B1 gene expression, consistent with an SL relationship) and in the 2_x_2 table cell (cells with high ATP1B1 gene expression are viable when ATP1B3 is lost), shown in Figure 5:

```
mts_plotClusterDistribution(mixModelClusters = mixturemodelClusters,
                            mixModelClusters2 = CRISPR_clusters,
                            gene1 = "ATP1B1", gene2 = "ATP1B3",
                            gene1_datatype="log2 gene expression",
                            gene2_datatype = "CRISPR gene effect score")
```

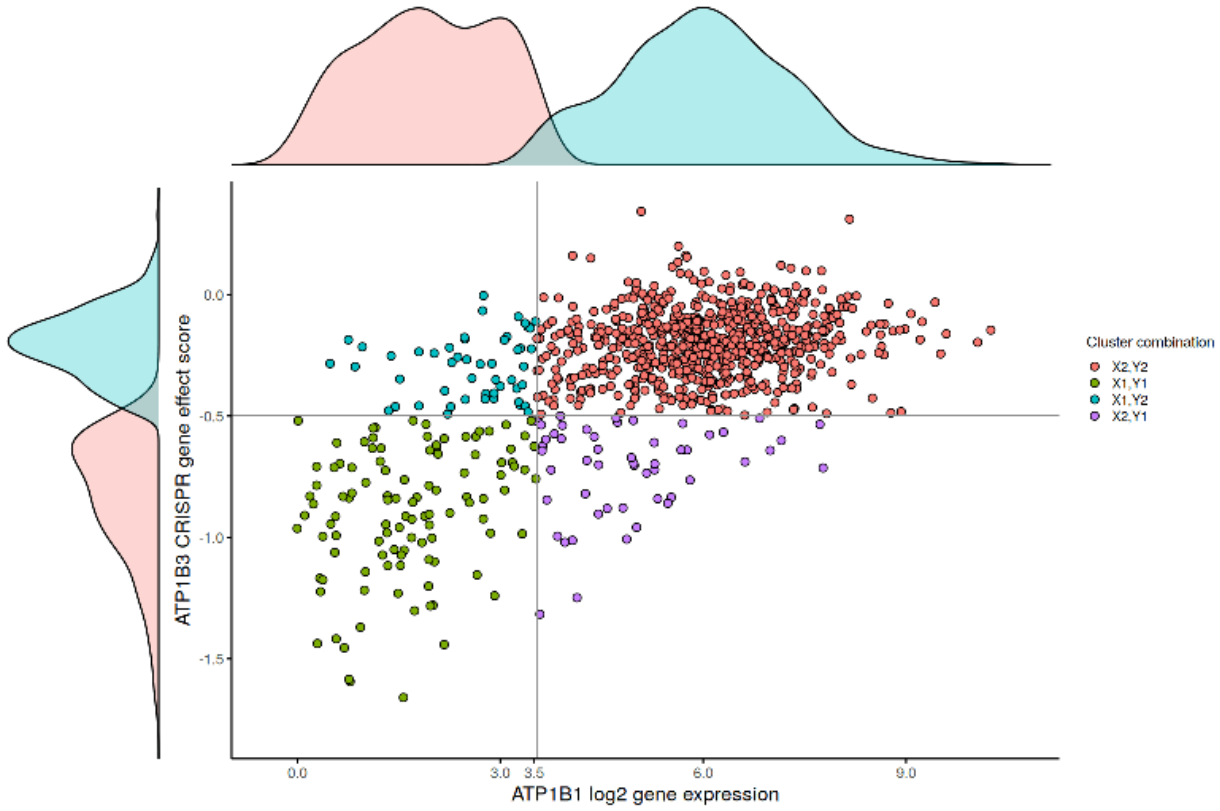

Figure 5: MultiSEp plot of ATP1B3 CRISPR scores and ATP1B1 gene expression. Clusters from Gaussian mixture modelling of the gene expression data are shown at the top for ATP1B1, and the results of partitioning the ATP1B3 CRISPR scores at a value of -0.5 are shown by the distributions at the left-hand side. Each circle in the plot represents a sample, coloured by the pairwise cluster combination. This example highlights how enrichment of samples may be evaluated across the whole contingency table with *mts_patternDetection()*; the bottom-left (1,1) and top-right (2,2) table cells have statistically significant enrichment.

Tissue-specific GDR detection

Context-specific gene dependencies are enriched within particular tissues or lineages (*i.e.* contexts). These relationships may reveal selective vulnerabilities with a potentially wider therapeutic window, due to sparing healthy tissue outside of the biological context where SL occurs. The mts_patternDetection() function can be used to detect these context-specific gene dependency patterns.

Predicting tissue-specific GDRs with transcriptome data

Transcriptomic data can be analysed to predict context-specific GDRs, including synthetic lethal (SL) relationships. The example below deploys mts_patternDetection() to evaluate candidate lung cancer SL relationships.

```
cellLines <- depMapTissue_subset$cell_line[
  depMapTissue_subset$tissue == "Lung Cancer"]

LungXPR_MCC <- lapply(mixturemodelClusters, function(GMM) {
  GMM[GMM$Sample %in% cellLines, ]
})

Lung_SL_depletionPattern <- mts_patternDetection(
  mixModelClusters1 = LungXPR_MCC,
  directionality = "depletion",
  qVal = 1,
  SyntheticLethalityPrediction = TRUE)
```

```
Lung_SL_depletionPattern %>%
  arrange(Actual_Count / Expected_Count) %>%
  head(10) %>%
  kable(row.names = FALSE, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results" = 12))
```

Results

Gene1	Gene2	Gene1_ClusterData	Gene2_ClusterData	Actual_Count	Expected_Count	TotalCluster_Combinations	Cluster_Combination	Sample_Count	p_value	Effect_Size	q_value
DNAJC15	PSMB8	1	1	0	1.2336449	9	1_x_1	107	0.2891495	0.9861954	1
DNAJC15	TTC7B	1	1	0	0.4112150	9	1_x_1	107	0.6623195	0.9861954	1
DNAJC15	CCDC88A	1	1	0	0.2056075	9	1_x_1	107	0.8139916	0.9861954	1
PSMB8	SNAP25	1	1	0	1.2897196	12	1_x_1	107	0.2731988	0.9861954	1
PSMB8	TTC7B	1	1	0	0.1121495	9	1_x_1	107	0.8938580	0.9861954	1
ACSL1	CCDC88A	1	1	0	0.9532710	6	1_x_1	107	0.3838349	0.9861954	1
ATP1B1	SNAP25	1	1	0	1.0747664	8	1_x_1	107	0.3395274	0.9861954	1
ATP1B1	TTC7B	1	1	0	0.0934579	6	1_x_1	107	0.9107391	0.9861954	1
ATP1B1	CCDC88A	1	1	0	0.0467290	6	1_x_1	107	0.9543363	0.9861954	1
ATP1B1	TUBA1A	1	1	0	1.9158879	4	1_x_1	107	0.1446775	0.9861954	1

When MYC and CCDC88A have low expression, there is a depletion of lung cancer cell lines (red) in the bottom left cell of the MultiSEp plot, consistent with an SL relationship and visualised in Figure 6 with mts_plotClusterDistribution():

```
mts_plotClusterDistribution(mixModelClusters = mixturemodelClusters,
                            TScluster1 = LungXPR_MCC,
                            gene1 = "MYC", gene2 = "CCDC88A")
```

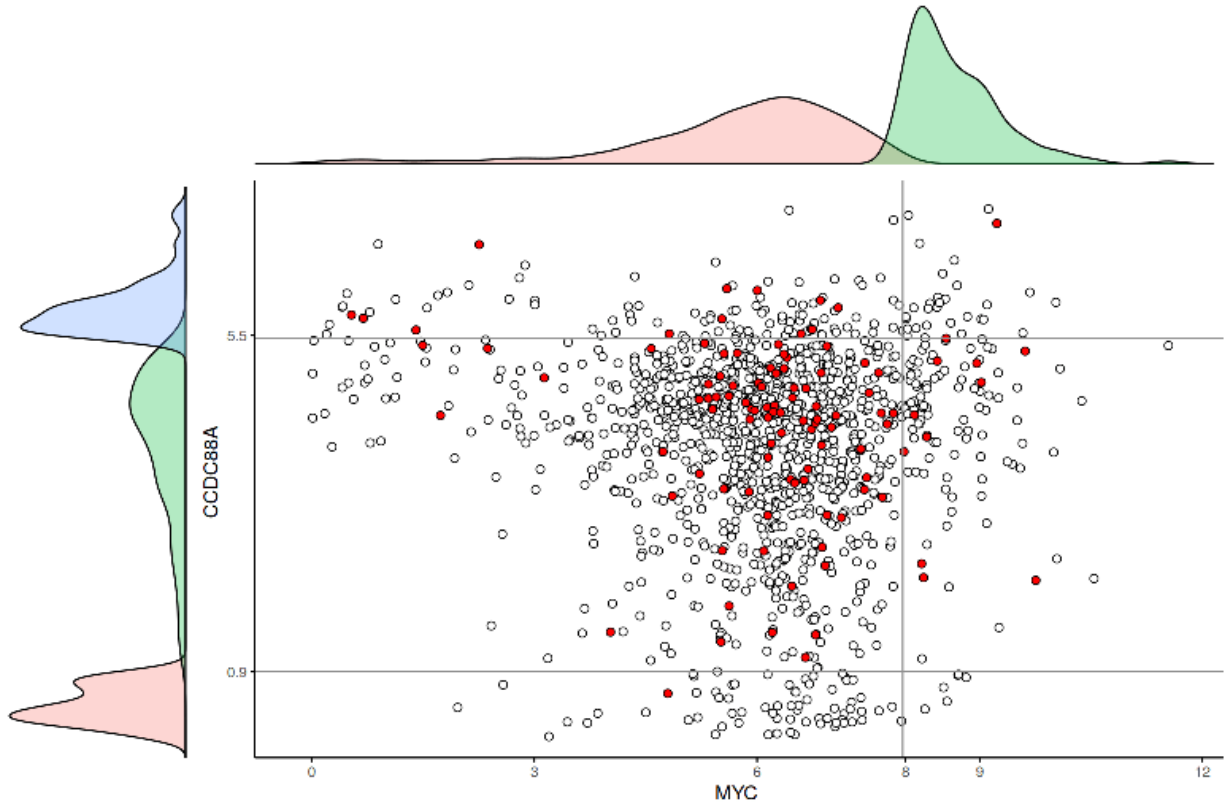

Figure 6: MultiSEp plot for CCDC88A and MYC gene expression, highlighting lung cancer cell lines. Clusters from Gaussian mixture modelling of the gene expression data are shown at the top and left-hand-side for MYC, CCDC88A respectively. The cluster boundaries form a contingency table and each circle in the table represents a cell line. Lung cancer cell lines are coloured red and the remaining cell lines, from various tissues, are white. The lung cell lines are depleted from the bottom-left contingency table cell, consistent with a tissue-specific SL relationship between CCDC88A and MYC.

Predicting tissue-specific GDRs with transcriptome and CRISPR data

This toy example shows how mts_patternDetection() can investigate SL relationships in a tissue-specific context with CRISPR and gene expression (XPR) data; specifically for lung cancer cell lines. Please note that the effect size threshold should not normally be specified and the q-value may be set to 0.05 for standard usage.

```
LungCRISPR_MCC <- lapply(CRISPR_clusters, function(GMM) {
  GMM[GMM$Sample %in% cellLines, ]
})

Lung_SL_enrichmentPattern <- mts_patternDetection(
  mixModelClusters1 = LungXPR_MCC,
  mixModelClusters2 = LungCRISPR_MCC,
  include_reverse_pairs = TRUE,
  SyntheticLethalityPrediction = TRUE,
  directionality = "enrichment",
  qVal = 1) # the default value is 0.05
```

```
Lung_SL_enrichmentPattern[Lung_SL_enrichmentPattern$Gene1=='ACSL1',] %>%
  kable(row.names = FALSE, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results" = 12))
```

Results											
Gene1	Gene2	Gene1_ClusterData	Gene2_ClusterData	Actual_Count	Expected_Count	TotalCluster_Combinations	Cluster_Combination	Sample_Count	p_value	Effect_Size	q_value
ACSL1	DNAJC19	1	1	30	30.504673	4	1_x_1	107	0.5793444	0.6047963	1
ACSL1	PSMB5	1	1	101	101.046729	4	1_x_1	107	0.6140702	0.9461640	1
ACSL1	ATP1B3	1	1	8	7.626168	4	1_x_1	107	0.4971162	0.5188724	1

We can visualise the candidate lung cancer SL relationship between ACSL1 and PSMB5 with the mts_plotClusterDistribution() function (Figure 7). The lung cancer samples (red) are largely located in the bottom left cell of the contingency table, which is consistent with a candidate SL relationship.

```
mts_plotClusterDistribution(mixModelClusters = mixturemodelClusters,
                            mixModelClusters2 = CRISPR_clusters,
                            TScluster1 = LungXPR_MCC,
                            TScluster2 = LungCRISPR_MCC,
                            gene1 = "ACSL1", gene2 = "PSMB5")
```

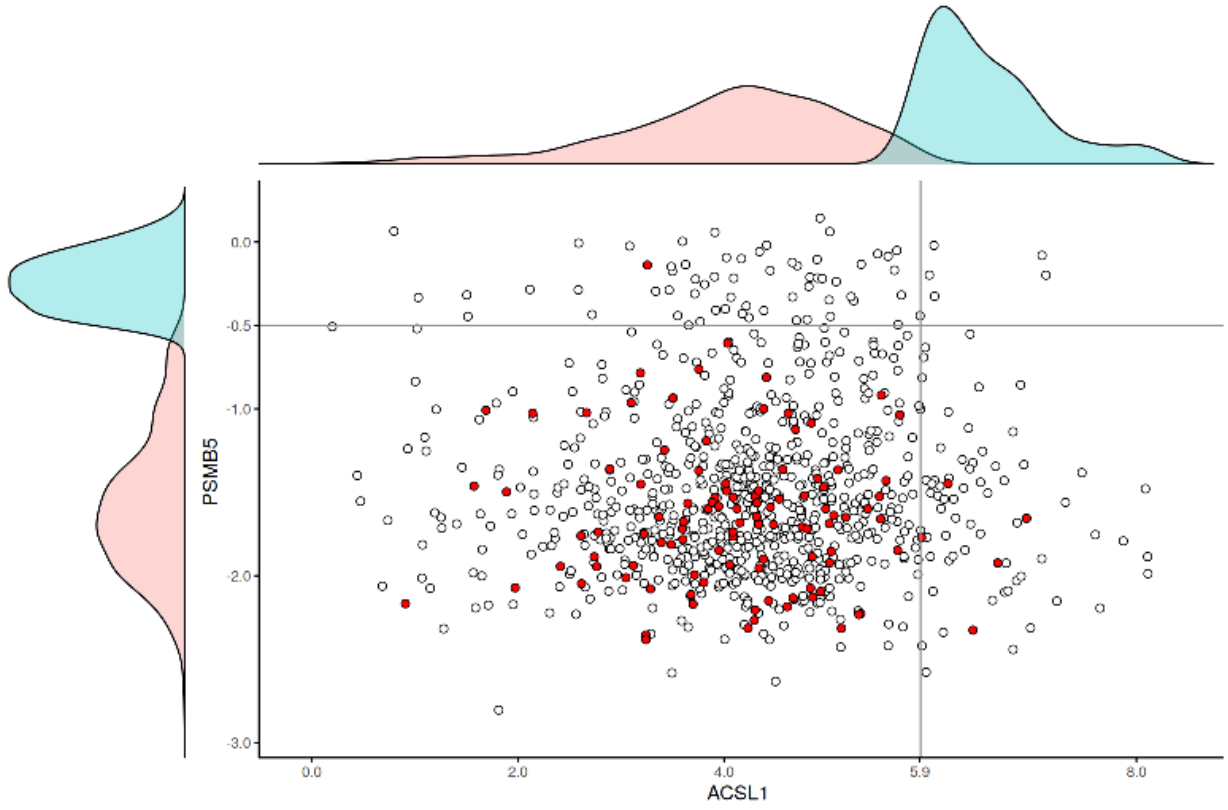

Figure 7: MultiSEp plot for PSMB5 CRISPR scores and ACSL1 gene expression, highlighting lung cancer cell lines. Clusters from Gaussian mixture modelling of ACSL1 gene expression are shown at the top and the results of partitioning the PSMB5 CRISPR scores at a value of -0.5 are shown by the distributions at the left-hand side. The cluster boundaries form a contingency table and each circle in the table represents a cell line. Lung cancer cell lines are coloured red and the remaining cell lines, from various tissues, are white. The lung cell lines are largely located in the bottom-left contingency table cell, consistent with a tissue-specific SL relationship between PSMB5 and ACSL1.

Predicting SL with CRISPR and gene expression (SynLeGG)

In the quickstart section we outlined mts_GeneDepCrispr() which is the simplest way to invoke the SynLeGG methods for predicting SL. Here we consider individual steps in the pipeline in more detail. mts_clusterAvg() takes as input a log2 gene expression matrix, and a gene effect score matrix in gene by sample format, performs cluster assignment and calculates cluster arithmetic means for each CRISPR gene.

```
clusterAssign <- mts_clusterAvg(exprsMatrix = depMapXPR_subset[1:5,],
                                depMapCRISPRscores_subset[c(4,27),])
```

The object clusterAssign is a list where each item corresponds to a gene result. Each gene result is a list of two data frames. Data frame 1 is a table of mean gene effect scores for every CRISPR gene:

```
knitr::kable(clusterAssign[[1]][[1]], format = "latex", longtable = FALSE)%>%
  kable_styling(latex_options = c("striped"))
```

	Cluster 1	Cluster 2	Cluster 3
NMT1	-0.7626355	-0.7691766	-0.8170489
CHMP4B	-0.8955272	-0.9207313	-1.0578216

Data frame 2 returns the cluster assignment and log2 gene expression value for each sample:

```
knitr::kable(head(clusterAssign[[1]][[2]], 10))%>%
  kable_styling(latex_options = c("striped"))
```

Sample	Values	Cluster_Assignment
NIHOVCAR3	0.0000000	1
HEL	6.9182670	3
HEL9217	7.1325768	3
LS513	4.9307373	3
C2BBE1	0.0426443	1
253J	0.0143553	1
HCC827	0.0840643	1
ONCODG1	0.1110313	1
HS294T	0.0840643	1
NCIH1581	3.2047668	3

The clusterAssign object is input for the mts_Crispr() function which performs statistical evaluation of dependency relationships by calculating a log2 fold-change and two-tailed t-test p-value between the CRISPR scores for each consecutive pair of mRNA expression clusters. The results are a ranked table of candidate synthetic lethal gene pairs, ordered by qvalue.

```
ttestCrisprResults <- mts_Crispr(resultList = clusterAssign,
                                 exprsMatrix = depMapXPR_subset[1:5,],
                                 depMapCRISPRscores_subset[c(4,27),])
```

The ttestCrisprResults output table contains lots of useful information including ranked gene pair names and the number of gene expression clusters (the CRISPR data is not clustered for this method).

```
knitr::kable(head(ttestCrisprResults[,1:3]), row.names = FALSE) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  add_header_above(c("Results: Gene_Summary"= 3))
```

Results: Gene_Summary		
mRNA_gene	crispr_gene	num_modes
NMT2	NMT1	2
STX2	NMT1	2
DNAJC15	CHMP4B	3
RPP25	CHMP4B	4

The number of samples per gene expression cluster (nMode) and the mean CRISPR score (mMode) per gene expression cluster are also available:

```
knitr::kable(head(ttestCrisprResults[,4:13]), row.names = FALSE, format = "latex",
↪  booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results: Mode_Information"= 10))
```

Results: Mode_Information									
nMode1	nMode2	nMode3	nMode4	nMode5	mMode1	mMode2	mMode3	mMode4	mMode5
624	272	NA	NA	NA	-0.9262627	-0.4434879	NA	NA	NA
351	545	NA	NA	NA	-0.9121036	-0.6944372	NA	NA	NA
145	730	21	NA	NA	-1.0550129	-0.9320655	-0.7125406	NA	NA
72	130	323	371	NA	-1.1334616	-1.0705946	-0.9474262	-0.8666924	NA

Finally, ttestCrisprResults contains some important statistics for each gene pair. Considering consecutive clusters, the gene effect score log2 fold change values are given (in the 'shift' columns) - this is the difference in average gene effect score between neighbouring clusters; p-values and q-values are also calculated for these changes. Negative fold-changes indicate a lower gene effect score in the lower expression cluster.

```
knitr::kable(head(ttestCrisprResults[,14:25]), row.names = FALSE, format = "latex",
↪  booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results: Statistics"= 12))
```

Results: Statistics											
shift2_1	shift3_2	shift4_3	shift5_4	pvalue2_1	pvalue3_2	pvalue4_3	pvalue5_4	qvalue2_1	qvalue3_2	qvalue4_3	qvalue5_4
-0.4827748	NA	NA	NA	0.000000	NA	NA	NA	0.0000000	NA	NA	NA
-0.2176664	NA	NA	NA	0.000000	NA	NA	NA	0.0000000	NA	NA	NA
-0.1229473	-0.2195250	NA	NA	0.003793	0.0356539	NA	NA	0.0075859	0.0713079	NA	NA
-0.0628669	-0.1231685	-0.0807338	NA	0.315942	0.0171982	0.0309604	NA	0.3159420	0.0274221	0.0619209	NA

Figure 8 shows the CRISPR gene effect score for the gene NMT1, split according to the NMT2 gene expression clusters. NMT1 CRISPR scores are significantly more negative in the low NMT2 expression cluster, predicting an SL relationship.

```
data("depMapTissue_subset")
mts_plotCRISPRGeneCluster(mrna_gene = "NMT2", crispr_gene = "NMT1",
                          crisprMatrix = depMapCRISPRscores_subset[c(4,27),],
                          tissueMatrix = depMapTissue_subset,
                          resultList = clusterAssign, plotType = "Integrated")
```

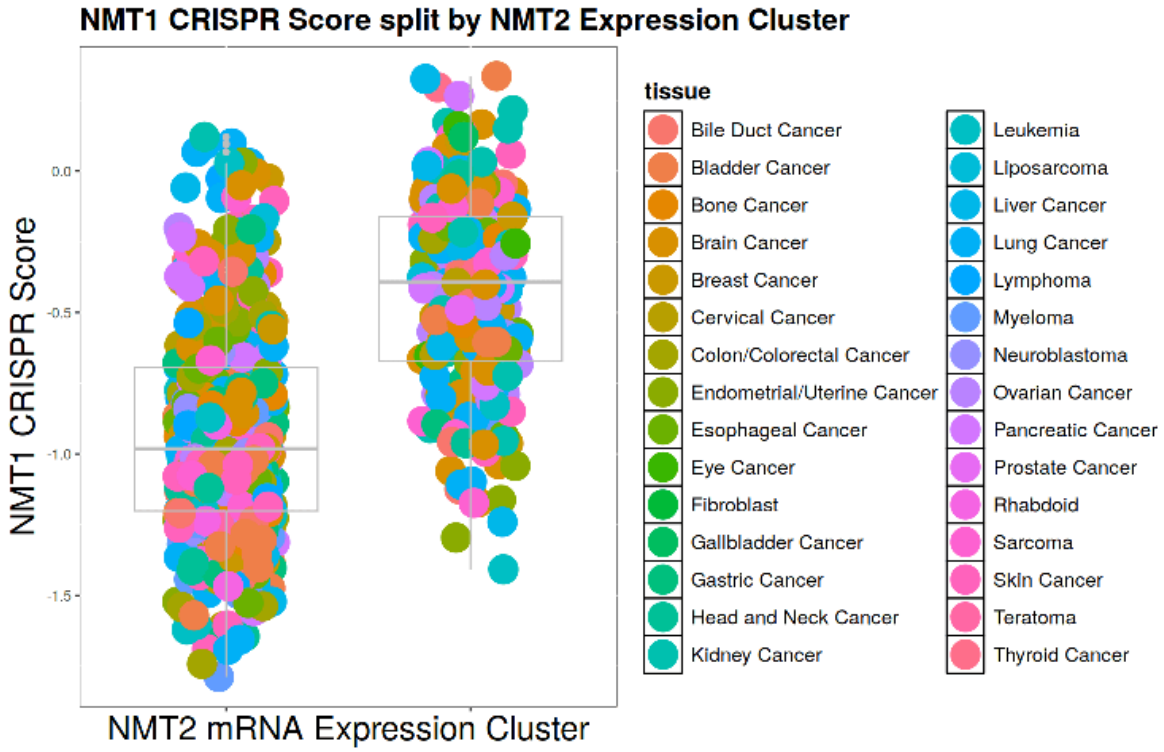

Figure 8: NMT1 CRISPR Scores are more negative with low NMT2 gene expression. The y axis shows NMT1 CRISPR scores and each box corresponds to a NMT2 gene expression cluster. The cell lines (circles) are coloured according to tissue type, please see the key for details.

It is also possible to view the gene expression distribution in the MultiSEp clusters (Figure 9):

```
data("depMapTissue_subset")
mts_plotCRISPRGeneCluster(mrna_gene = "NMT2", crispr_gene = "NMT1",
                          crisprMatrix = depMapCRISPRscores_subset[c(4,27),],
                          tissueMatrix = depMapTissue_subset,
                          resultList = clusterAssign, plotType = "mrna_only")
```

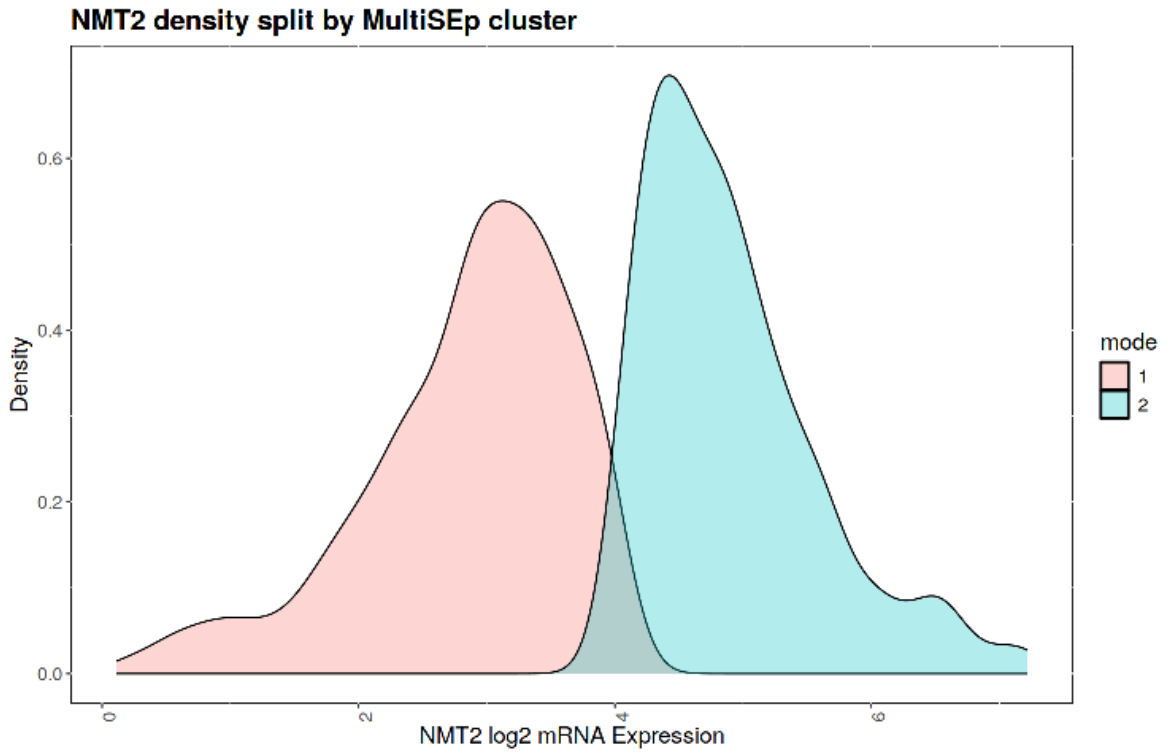

Figure 9: Visualising MultiSEp gene expression clusters. The x-axis corresponds to NMT2 log2 mRNA expression and each curve represents a gene expression cluster, the allocation of colourings to cluster identity (mode) is shown in the key.

Or a waterfall plot of the CRISPR data and gene expression clusters (Figure 10):

```
data("depMapTissue_subset")
mts_plotCRISPRGeneCluster(mrna_gene = "NMT2", crispr_gene = "NMT1",
                          crisprMatrix = depMapCRISPRscores_subset[c(4,27),],
                          tissueMatrix = depMapTissue_subset,
                          resultList = clusterAssign, plotType = "crispr_only")
```

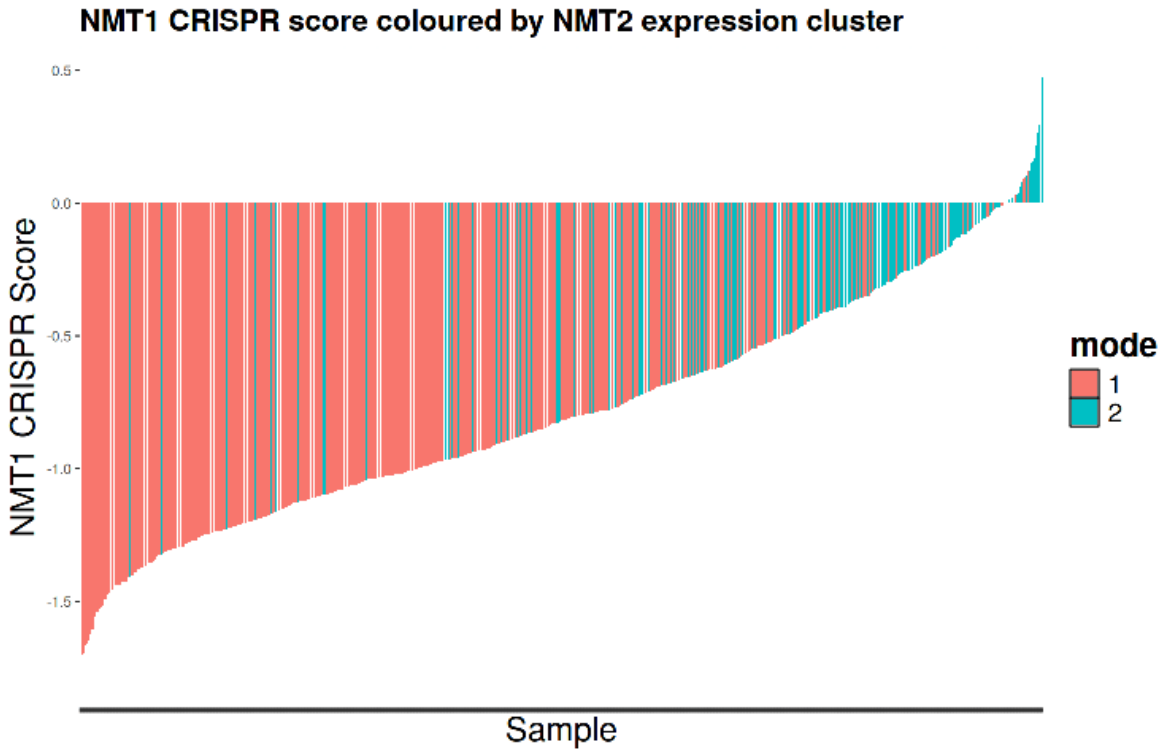

Figure 10: Waterfall plot of NMT1 CRISPR score and NMT2 gene expression cluster. The bars are ordered from low to high CRISPR score and coloured according to the MultiSEp gene expression cluster, allocation of colourings to cluster identity is given in the key. Mode (cluster) 1 is enriched for low CRISPR scores, indicating a greater effect on cell viability/growth with loss of NMT1 when NMT2 has low expression.

Tissue-specific GDR detection (SynLeGG)

GDRs are often restricted to a single tissue type, or a subset of tissues. Many genes are expressed in a subset of tissues, which is one factor that leads to tissue-specific dependencies and is sometimes referred to as tissue penetrance. MultiSEp includes functionality to evaluate tissue-specific GDRs:

```
ttestCrisprResultsTS <- mts_CrisprTS(resultList = clusterAssign,
                                     crisprMatrix = depMapCRISPRscores_subset[c(4,27),],
                                     fcVal = -0.1, pVal = 0.25,
                                     tissueMatrix = depMapTissue_subset,
                                     allDisRes = ttestCrisprResults, cores=1)
```

As with the pan-tissue results, the output (i.e. ttestCrisprResultsTS) contains lots of useful information including ranked gene pair names, tissue and cluster number:

```
knitr::kable(head(ttestCrisprResultsTS[,1:4]), row.names = FALSE) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  add_header_above(c("Results: Gene_Summary"= 4))
```

Results: Gene_Summary			
mRNA_gene	crispr_gene	tissue	num_modes
NMT2	NMT1	Bone Cancer	2
NMT2	NMT1	Gastric Cancer	2
NMT2	NMT1	Bladder Cancer	2
NMT2	NMT1	Brain Cancer	2
NMT2	NMT1	Ovarian Cancer	2
NMT2	NMT1	Leukemia	2

The number of samples per gene expression cluster and the average CRISPR score per gene expression cluster are shown below. This table could be quite sparse due to low tissue-specific sample number, or non-represented modes for some tissues.

```
knitr::kable(head(ttestCrisprResultsTS[,5:14]), row.names = FALSE, format = "latex",
↪  booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results: Mode_Information"= 10))
```

Results: Mode_Information									
nMode1	nMode2	nMode3	nMode4	nMode5	mMode1	mMode2	mMode3	mMode4	mMode5
18	7	0	0	0	-0.9529078	-0.4115328	0	0	0
23	4	0	0	0	-1.0081713	-0.5885459	0	0	0
21	8	0	0	0	-1.1790424	-0.3118338	0	0	0
29	30	0	0	0	-0.7529485	-0.4702167	0	0	0
21	20	0	0	0	-0.8636750	-0.3892459	0	0	0
38	5	0	0	0	-1.0040554	-0.8374052	0	0	0

The results also include some important statistics for each gene pair. A set of tissue-specific consecutive cluster gene effect shift values (log2 fold change) are returned, which indicate the difference in average gene effect score between ascending, neighbouring clusters. The p-values and q-values for these differences are also available:

```
knitr::kable(head(ttestCrisprResultsTS[,15:26]), row.names = FALSE, format = "latex",
↪  booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Results: Statistics"= 12))
```

Results: Statistics												
shift2_1	shift3_2	shift4_3	shift5_4	pvalue2_1	pvalue3_2	pvalue4_3	pvalue5_4	qvalue2_1	qvalue3_2	qvalue4_3	qvalue5_4
-0.5413750	0	0	0	0.0003702	0	0	0	0.0011724	NA	NA	NA
-0.4196254	0	0	0	0.0865012	0	0	0	0.1027202	NA	NA	NA
-0.8672086	0	0	0	0.0000325	0	0	0	0.0002059	NA	NA	NA
-0.2827319	0	0	0	0.0041498	0	0	0	0.0112637	NA	NA	NA
-0.4744291	0	0	0	0.0000209	0	0	0	0.0001987	NA	NA	NA
-0.1666502	0	0	0	0.4284209	0	0	0	0.4522221	NA	NA	NA

For the functional paralogues NMT1 and NMT2, the mutually exclusive loss relationship is most statistically significant in Ovarian Cancer (Figure 11):

```
mts_plotCRISPRGeneCluster(mrna_gene = "NMT2", crispr_gene = "NMT1",
                          crisprMatrix = depMapCRISPRscores_subset[c(4,27),],
                          tissueMatrix = depMapTissue_subset,
                          diseaseFilter = "Ovarian Cancer",
                          resultList = clusterAssign, plotType = "Integrated")
```

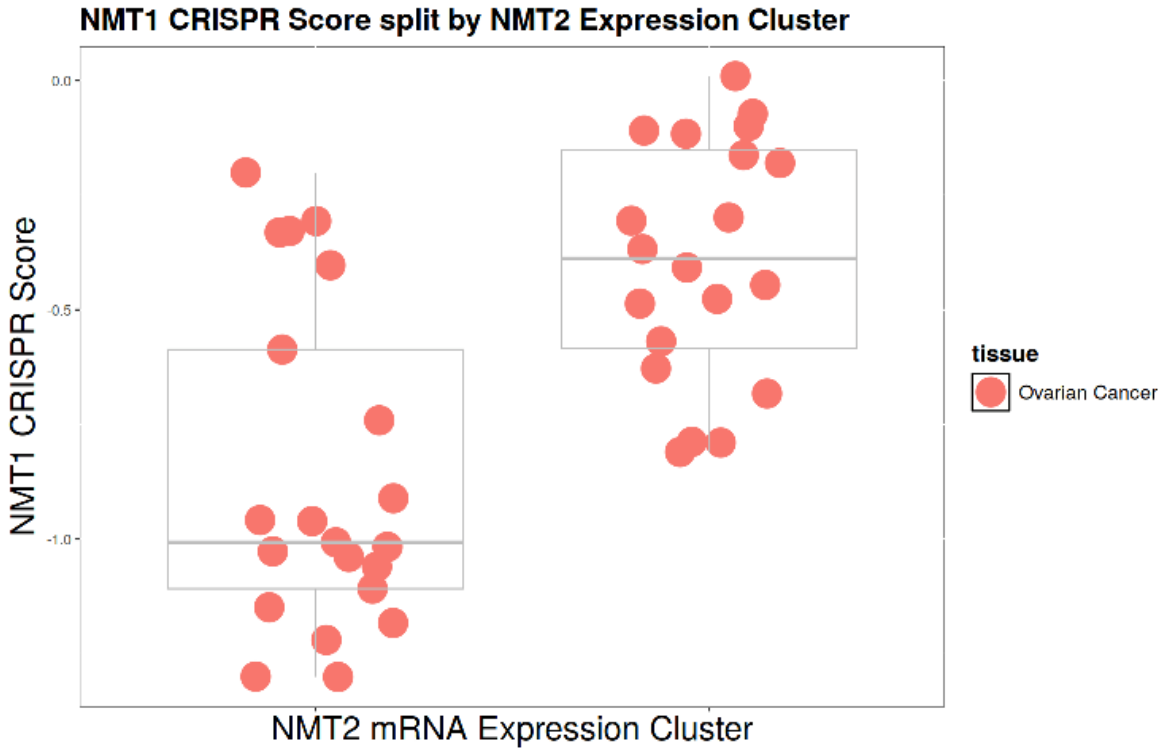

Figure 11: NMT1 CRISPR scores for NMT2 Gene expression clusters in Ovarian Cancer. The y axis shows NMT1 CRISPR scores and each box corresponds to a NMT2 gene expression cluster. Only Ovarian cancer cell lines are shown (circles).

Induced dependency prediction (t-test)

Prediction of Synthetic Lethality (SL) is canonically based upon a pattern of mutually exclusive loss. However it is possible that increased expression may be a signature of dependency, sometimes called synthetic dosage lethality. The mts_InducedDependency() function defaults to return GDRs where cell death is enhanced for loss of gene 1 and high expression of gene 2. Here, the clusterAssign object is input into the mts_InducedDependency() function which performs statistical evaluation of dependency relationships by calculating a log2 fold-change and two-tailed t-test p-value between the CRISPR scores for each consecutive pair of mRNA expression clusters. For induced dependency, a positive fold-change threshold is needed (default):

```
ttestIDResults <- mts_InducedDependency(resultList = clusterAssign,
                                exprsMatrix = depMapXPR_subset[1:5,],
                                crisprMatrix = depMapCRISPRscores_subset[c(4,27),], fcVal
                                ↪  = 0.1)
```

The mts_InducedDependency() output is structured in the same way as for mts_Crispr(). Positive shift values (fold-changes) indicate lower gene 1 effect scores (implying cell death/growth inhibition) in the higher gene 2 expression cluster:

```
knitr::kable(head(ttestIDResults[,14:25]), row.names = FALSE, format = "latex", booktabs
↪  = TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down", "hold_position")) %>%
  add_header_above(c("Statistics"= 12))
```

Statistics												
shift2_1	shift3_2	shift4_3	shift5_4	pvalue2_1	pvalue3_2	pvalue4_3	pvalue5_4	qvalue2_1	qvalue3_2	qvalue4_3	qvalue5_4
0.2415025	NA	NA	NA	0.0000000	NA	NA	NA	0.0000000	NA	NA	NA
0.3888484	NA	NA	NA	0.0000000	NA	NA	NA	0.0000000	NA	NA	NA
0.0252042	0.1370903	NA	NA	0.7110523	0.0567287	NA	NA	0.9194376	0.1134573	NA	NA

For example, CHMP4B CRISPR gene effect scores are lower for high STX2 expression, predicting an induced dependency (synthetic dosage lethal) relationship (Figure 12):

```
data("depMapTissue_subset")
mts_plotCRISPRGeneCluster(mrna_gene = "STX2", crispr_gene = "CHMP4B",
                              crisprMatrix = depMapCRISPRscores_subset,
                              tissueMatrix = depMapTissue_subset,
                              resultList = clusterAssign, plotType = "Integrated")
```

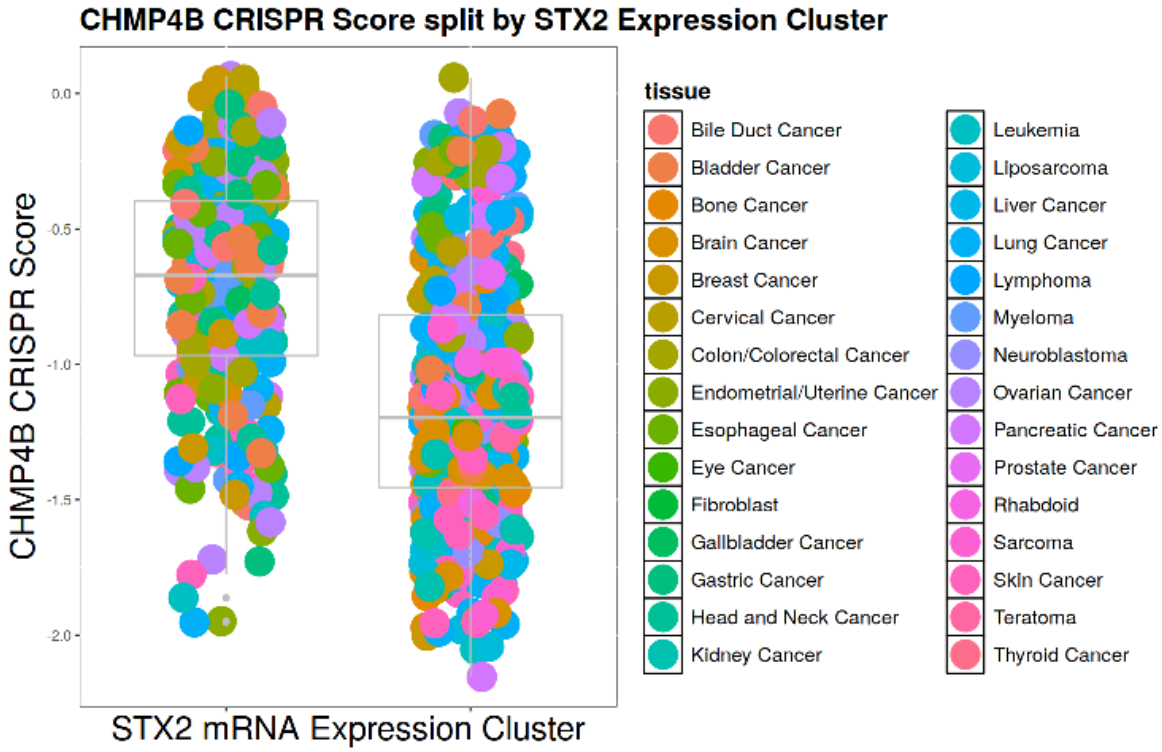

Figure 12: Candidate induced dependency relationship between CHMP4B and STX2. CHMP4B CRISPR Scores are more negative with high STX2 gene expression, consistent with an induced dependency relationship. The y axis shows CHMP4B CRISPR scores and each box corresponds to a STX2 gene expression cluster. The cell lines (circles) are coloured according to tissue type, please see the key for details.

We can again run the mts_CrisprTS() pipeline to assess tissue-specific penetrance for induced dependency:

```
ttestIDResultsTS <- mts_CrisprTS(resultList = clusterAssign,
                                 crisprMatrix = depMapCRISPRscores_subset[c(4,27),],
                                 fcVal = 0.1, # set to 0.1 to predict induced dependency
                                 ↪  relationships
                                 pVal = 0.25,
                                 tissueMatrix = depMapTissue_subset,
                                 allDisRes = ttestIDResults, cores=1)
```

In the case of the functional paralogues STX2 and CHMP4B, the induced dependency relationship is most statistically significant in Breast Cancer (Figure 13):

```
mts_plotCRISPRGeneCluster(mrna_gene = "STX2", crispr_gene = "CHMP4B",
                          crisprMatrix = depMapCRISPRscores_subset,
                          tissueMatrix = depMapTissue_subset,
                          diseaseFilter = "Breast Cancer",
                          resultList = clusterAssign, plotType = "Integrated")
```

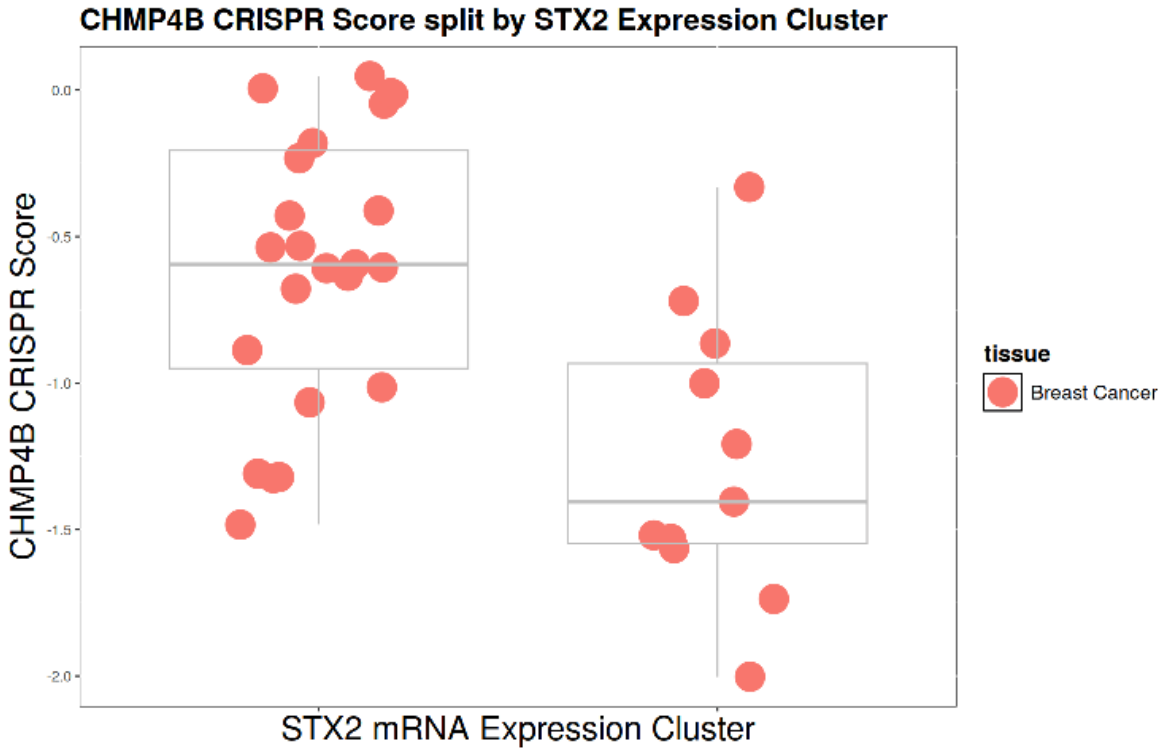

Figure 13: Candidate induced dependency relationship between CHMP4B and STX2 in breast cancer. CHMP4B CRISPR Scores are more negative with high STX2 gene expression, consistent with an induced dependency relationship. The y axis shows CHMP4B CRISPR scores and each box corresponds to a STX2 gene expression cluster. Only breast cancer cell lines are shown, where this predicted induced dependency relationship has the strongest p-value.

Mutation and expression GDRs (SynLeGG)

The mts_Mutation() function takes the clusters from mts_mixModelCluster(), a gene by sample mutation matrix containing 'WT' or a mutation call, and optionally a tissue to sample tissue map. mts_Mutation() detects dependencies between mutations and gene expression clusters by performing the chi-squared test, which produces p-values. Analysis may be performed across all tissues, or in a tissue-specific manner.

```
mixModelClusters <- mts_mixModelCluster(dataMatrix = depMapXPR_subset[c(1,12,15,20,25),])

multisepMutAll <- mts_Mutation(resultList = mixModelClusters, mutMatrix =
↪  depMapMUT_subset,
                               pVal = 0.01)
```

The results are a data frame containing data for the dependency of mutations with one or more gene expression modes (clusters), quantified by chi-squared p-values, and the number of mutations in each mode. Both classical SL and induced dependency GDR patterns are evaluated. The mutation_per_mode attribute gives details of the number of mutations identified in the different gene expression modes (clusters):

```
knitr::kable(head(multisepMutAll), row.names = FALSE) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  add_header_above(c("Mutation Cluster Enrichment"= 5))
```

Mutation Cluster Enrichment				
mRNA_gene	mutation_gene	tissue	chiSqPvalue	mutation_per_mode
CDKN1A	TP53	All	0.0e+00	390,463
BAX	TP53	All	0.0e+00	374,479
FERMT1	TP53	All	0.0e+00	481,143,229
BAX	SRCIN1	All	2.0e-07	92,147
EIF1AY	MT-ATP6	All	1.2e-06	80,59,22,61
FERMT1	APC	All	3.7e-06	139,25,49

The top hit in the example above is an enrichment of TP53 mutation in the low expression cluster of CDKN1A (Figure 14). However, wild-type TP53 is expected to activate and lead to higher expression of CDKN1A. Therefore this example is not a true induced dependency (synthetic dosage lethal) relationship and likely represents a false positive.

```
mts_plotMutation(resultList = mixModelClusters, mrna_gene = "CDKN1A", mut_gene = "TP53",
                 mutMatrix = depMapMUT_subset, tissueMatrix = depMapTissue_subset)
```

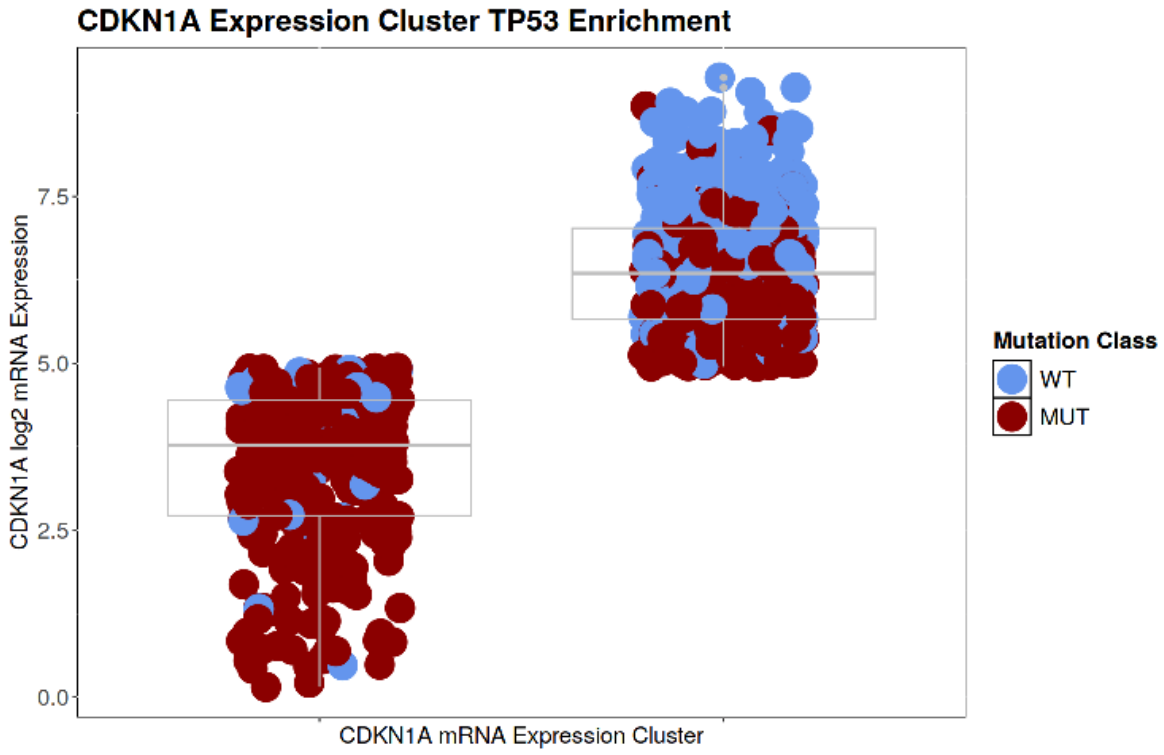

Figure 14: TP53 mutations associate with lower CDKN1A expression. This example shows how mutations may be visualised across gene expression clusters. The y axis shows expression of CDKN1A and each box corresponds to the CDKN1A expression cluster. The samples (cell lines) are coloured according to the TP53 mutation class, which is detailed in the key.

Addition of a tissue matrix allows mts_Mutation() to be run in a tissue-specific manner.

```
multisepMutTS <- mts_Mutation(resultList = mixModelClusters,
                              mutMatrix = depMapMUT_subset, pVal = 0.01,
                              tissueMatrix = depMapTissue_subset)
```

Tissue-specific results are now available:

```
knitr::kable(head(multisepMutTS, 5), row.names = FALSE) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  add_header_above(c("Mutation Cluster Enrichment"= 5))
```

Mutation Cluster Enrichment				
mRNA_gene	mutation_gene	tissue	chiSqPvalue	mutation_per_mode
EIF1AY	FRAS1	Head and Neck Cancer	3.00e-07	0,0,0,1
EIF1AY	VCAN	Bile Duct Cancer	1.40e-06	0,0,1
FERMT1	MUC4	Lymphoma	2.75e-05	2,0,0
FERMT1	LRP1	Neuroblastoma	4.54e-05	0,0,1
EIF1AY	TENM2	Kidney Cancer	6.52e-05	0,0,0,1

The enrichment of TP53 mutations in the CDKN1A low expression cluster is particularly clear in Ovarian cancer (Figure 15):

```
mts_plotMutation(resultList = mixModelClusters, mrna_gene = "CDKN1A",
                 mut_gene = "TP53", mutMatrix = depMapMUT_subset,
                 tissueMatrix = depMapTissue_subset,
                 diseaseFilter = "Ovarian Cancer")
```

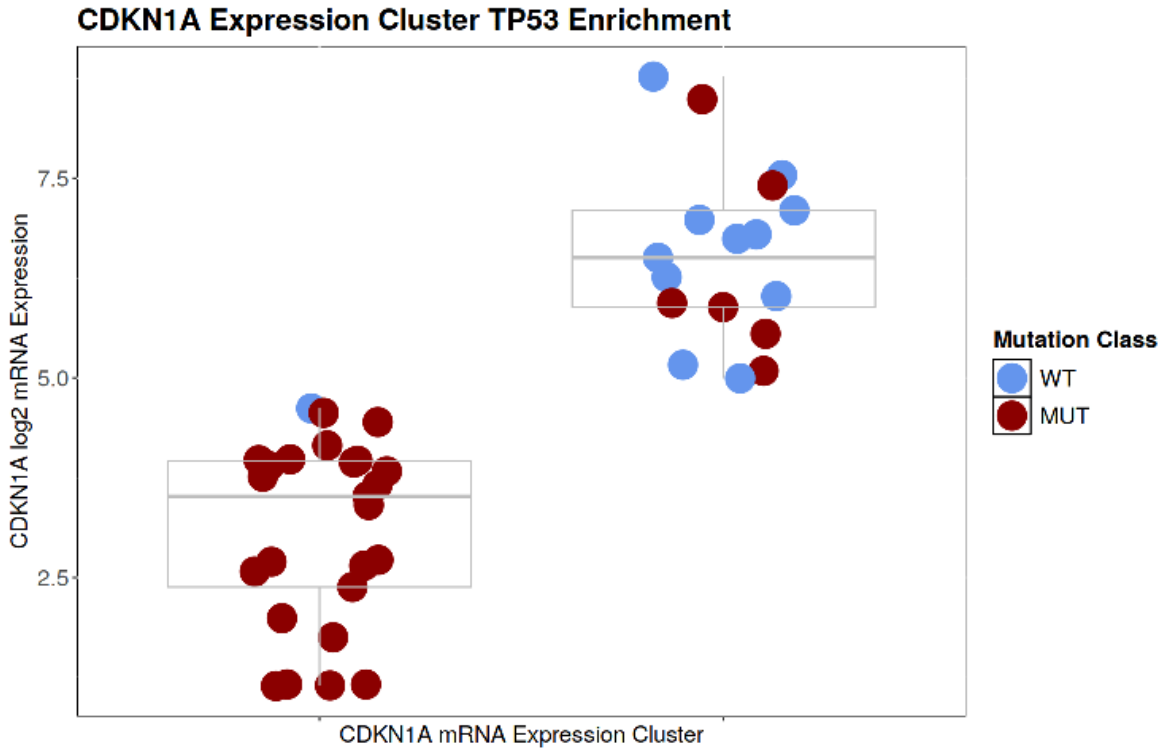

Figure 15: Tissue-specific analysis of TP53 mutations and CDKN1A expression. The enrichment of TP53 mutations in the low CDKN1A expression cluster is shown for Ovarian cancer cell lines. The y axis shows expression of CDKN1A and each box corresponds to the CDKN1A expression cluster. Samples are coloured according to the TP53 mutation class, which is detailed in the key.

Analysis of mutational neighbourhoods in SL networks for therapeutic target prioritisation

The mts_targetCoverageFromMutations() function can integrate mutational data with a network of predicted SL relationships in order to estimate the proportion of a population that could benefit from inhibiting any one of the candidate target genes. For example, the population could be cancer patients. We reason that an inactivating mutation in a synthetic lethal partner gene exposes (cancer) cells to killing when the functional gene in the pair is inhibited by a drug. mts_targetCoverageFromMutations() takes as input an object with gene pairs (two columns, corresponding to each gene in the pair) and a gene by sample mutation matrix containing 'WT' or a mutation call (e.g. details of amino acid changes produced by point mutations or INDELs). Gene pairs of interest may be provided through the *edgeData* argument; alternatively, results generated with mts_Mutation() may be assigned to *edgeData*. Increasing the value of *cores* is recommended if possible:

```
data("depMapMUT_subset")
netN <- mts_targetCoverageFromMutations(mutData=depMapMUT_subset,
                                        edgeData=multisepMutAll, cores=1)
```

The mts_targetCoverageFromMutations() results are a data frame containing: the candidate target gene ('Gene'), neighbours ('NeighbourGenes'), the samples that have a mutation in a neighbouring gene ('MutatedSamples'), and the population frequency of mutations in any of the genes neighbouring the candidate target gene ('NeighbourMutationFrequency').

```
knitr::kable(head(netN[1:4]), row.names=FALSE) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Gene	NeighbourCount	NumSamplesWithMutation	NeighbourMutationFrequency
FERMT1	33	1709	0.9833142
BAX	25	1663	0.9568470
EIF1AY	16	1628	0.9367089
CDKN1A	4	1365	0.7853855
TTC7B	3	818	0.4706559
TP53	4	0	0.0000000

Ordering by the neighbour mutation frequency reveals the candidate target genes with highest predicted population coverage. In this toy example, AP1M1 has 44 neighbours and 1721 samples have mutations in one of the neighbouring genes. Please note that this function is intended for use with patient data to estimate population coverage, whereas the example here is with cell line data; additionally the data were not filtered to ensure that the mutation produces loss of function (e.g. selecting 'high impact' mutations) - which is an important step.

4. Running MultiSEp in a High-Performance Computing (HPC) Environment for Evaluation of Context-specific SL at Genome Scale

This section presents an approach for predicting synthetic lethal (SL) relationships at genome scale with high-performance computing (HPC). We investigate candidate SL relationships for Multiple Myeloma gene expression from the CoMMpass Study (Skerget *et al.* 2024). This 'all-vs-all' comparison involves running the SL discovery workflow for approximately 105 million gene pairs, therefore use of a HPC cluster is required for timely production of results.

Step 1: Produce gene expression clusters

The first step in predicting the synthetic lethal network is to obtain gene expression clusters with Gaussian mixture modelling (GMM) using the mts_mixModelCluster_XPR() function to analyse the MMRF CoMMpass transcriptomic dataset (n=718 samples). When run in a HPC environment, where an individual cluster node may have a large number of cores, you may increase the value of the cores argument (for example, set to 50 below). We recommend saving the output of mts_mixModelCluster_XPR() as an .RData file for downstream analysis, for example:

```
CoMMpassXPRClusters <- mts_mixModelCluster_XPR(
  dataMatrix = CoMMpass_transcriptomicData,
  GeneXPRthresh = 10, # data is not logged
  NumSampleThresh = 20,
  cores = 50)

save(CoMMpassXPRClusters, file = "CoMMpassXPRClusters.RData")
```

Step 2: Preparing the clustered genes for parallel processing

The second step runs the mts_genepairsChunkGeneration() function to prepare the data for parallel processing and involves splitting the CoMMpassXPRClusters object into multiple .RData objects, each containing a subset of the genes to be analysed. mts_genepairsChunkGeneration() also removes any genes where expression clustering failed, for example if there is only one expression value for the gene. The num_tasks argument

stores the number of 'chunks' (tasks) to be run in the array job on the HPC cluster and determines the number of files to be saved to the current working directory. For this exemplar, involving a genome scale multiple myeloma GDR network, mts_mixModelCluster_XPR() found clusters for 14,549 genes. Accordingly, all vs all GDR discovery requires binomial tests for 105,829,426 gene pairs; when num_tasks is set to 1000, there will be 105,830 gene pairs in 999 'chunks' (tasks) and 105,256 pairs analysed in the 1000th 'chunk' (task):

```
result <- mts_genepairsChunkGeneration(mixModelClusters1 = CoMMpassXPRClusters,
                                       num_tasks = 1000,
                                       output_dir =
                                       ↪  "/example/outputdirectory/array_chunks/")
```

Step 3: Predicting SL with `mts_omics()`

To calculate 105,829,426 binomial test depletion p-values (undirected pairs for the 14,549 genes) a script is required to run mts_omics() in the HPC environment. The code below is run as a Rscript within a SLURM array job (with 1000 tasks), and where the SLURM task ID is given as a command-line argument. Please note that the cluster node cores requested via SLURM should match the cores argument - set to 10 in the code below; we recommend 4 Gb RAM per CPU for this application (specified in the bash script below).

```
library(MultiSEp)
# Identify the SLURM task ID
args <- commandArgs(trailingOnly=TRUE)
SlurmTaskID  <- args[1]
chunkdirectory  <- "/example/outputdirectory/array_chunks/"
filename <- paste0(chunkdirectory,"genepairs_chunk_", SlurmTaskID, ".RData")

# Run binomial test
load(filename)
load("/example/outputdirectory/CoMMpassXPRClusters.RData")
SL_CoMMpass <- mts_omics(
    genepairs = subset_genepairs,
    mixModelClusters1 = CoMMpassXPRClusters, # the GMM clustering object
    SyntheticLethalityPrediction = TRUE,
    p_adjustMethod = "BY",
    qVal = 2,
    effectsize = TRUE,
    effectsize_threshold = 0,
    directionality = "depletion",
    cores = 10 # this number matches the cpus-per-task in the bash script below
)
# Save the results to file
output_directory2 <- "/example/outputdirectory/"
text_file <- paste0(output_directory2, "SL_CoMMpass_", SlurmTaskID, ".txt")
write.table(SL_CoMMpass, file = text_file, quote = FALSE, sep = "\t",
            row.names = FALSE)
```

In this example, we run a bash script to call the above Rscript:

```
#!/bin/bash
#SBATCH --job-name=mts_omics_array
#SBATCH --mail-type=END
#SBATCH --mail-user=youruser@email.address
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10
```

```
#SBATCH --mem-per-cpu=4G
#SBATCH --partition=optional_queue_name_on_your_cluster
#SBATCH --chdir=/path/to/your/directory/for/logfiles/log/
#SBATCH --array=1-1000

## other commands that you might need to set up the environment, for example "module load
↪  R"

Rscript /path/to/directory/where/you/have/the/above/Rscript/rscriptName.R
↪  $SLURM_ARRAY_TASK_ID
```

When run as a SLURM array job with 1000 tasks there are 1000 files which may be combined as follows:

```
awk 'FNR == 1 && NR != 1 { next } { print }' SL_CoMMpass_{1..1000}.txt >
↪  combined_SL_CoMMpass.txt
```

The analysis for predicting synthetic lethal depletion patterns was performed in parallel for each subset of gene pairs, therefore adjusting p-values for multiple comparisons and q-value filtering should be calculated on the combined set of gene pairs:

```
networkBinomial = read.table("combined_SL_CoMMpass.txt",
                             header = TRUE)
# Adjust p-values for multiple comparisons with the Benjamini & Yekutieli (BY) method
p_adjusted = p.adjust(networkBinomial$p_value, method = "BY")
networkBinomial$New_q_value = p_adjusted

# Filtering by recomputed q-value < 0.05 and effect size >= 0.9621389
networkBinomialFiltered = subset(networkBinomial,
                                 New_q_value < 0.05 & Effect_Size >= 0.9621389)

# Save to a file
write.table(networkBinomialFiltered,
            "multiSEp_SL_XPR_qval05.txt",
            quote = FALSE, row.names=FALSE)
```

In the example above, the predicted multiple myeloma SL network is saved to the file multiSEp_SL_XPR_qval05.txt

Step 4: Identifying deleterious mutations to inform drug discovery

A gene that has an SL relationship with one or more genes that are mutated may be an attractive drug target for precision oncology. In this case study, mutational data was sourced from the CoMMpass Study (Skerget *et al.* 2024). Our analysis only utilised deleterious mutations predicted to be high impact by the variant effect prediction tools, SnpEff and SnpSift. The bash command below can be run from the command line to extract high impact mutations:

```
(head -n 1 Somatic_Observed_SNV_INDEL_mutations.txt; grep HIGH
↪  Somatic_Observed_SNV_INDEL_mutations.txt) >
↪  HighImpact_Somatic_Observed_SNV_INDEL_mutations.txt
```

The following python script may be used to process the data into a format suitable for analysis by MultiSEp - creating a mutational score matrix in gene by patient format with values either “WT” or “MUT”:

```
"""Build a matrix of genes (rows) by samples (columns) from a list of
high-impact variant calls. Each cell is 'MUT' if the gene carries >=1 variant
```

```
in that sample, otherwise 'WT'.
"""
import pandas as pd
import os
# directory of the script
script_dir = os.path.dirname(os.path.realpath(__file__))
data_path = "HighImpact_Somatic_Observed_SNV_INDEL_mutations.txt"
data = pd.read_csv(data_path, sep='\t')

matrix = (pd.crosstab(data['GENEID'], data['sample'])
            .gt(0)
            .replace({True: 'MUT', False: 'WT'}))
# Name the row and column headers used in the saved file.
matrix.index.name = 'Gene'
matrix.columns.name = 'Sample'

print(matrix)
print(matrix.isnull().values.any())   # expect False: every cell is filled
print(matrix.isnull().sum().sum())     # expect 0

# save the matrix to a tab-separated file in the output folder
output_filename = "MutationalMatrix_CoMMpassSLnet.txt"
output_path = os.path.join(script_dir, "..", "output", output_filename)
matrix.to_csv(output_path, sep='\t', index=True)
```

Step 5: Predicting the population coverage achieved by inhibition of a candidate target gene

The mutational score matrix and predicted SL network are analysed by the mts_targetCoverageFromMutations() function in order to predict the proportion of patients that may benefit from targeting a candidate gene.

```
SLedges <- read.table("multiSEp_SL_XPR_qval05.txt", header=TRUE)
MutationMatrix <- read.table("MutationalMatrix_CoMMpassSLnet.txt",
                             sep ="\t", header=TRUE, row.names=1)

PopulationCoverage <- mts_targetCoverageFromMutations(
  mutData = MutationMatrix, edgeData = SLedges, cores = 10)
```

Visualisation of predicted SL network for Multiple Myeloma

The network resulting from the above analysis may be visualised with applications such as Cytoscape (Shannon *et al.* 2003). Figure 16 visualises MultiSEp results for the Myeloma transcriptome data (MMRF CoMMpass, Skerget *et al.* 2024):

```
knitr::include_graphics("figures/CoMMpassSLnetwork.png")
```

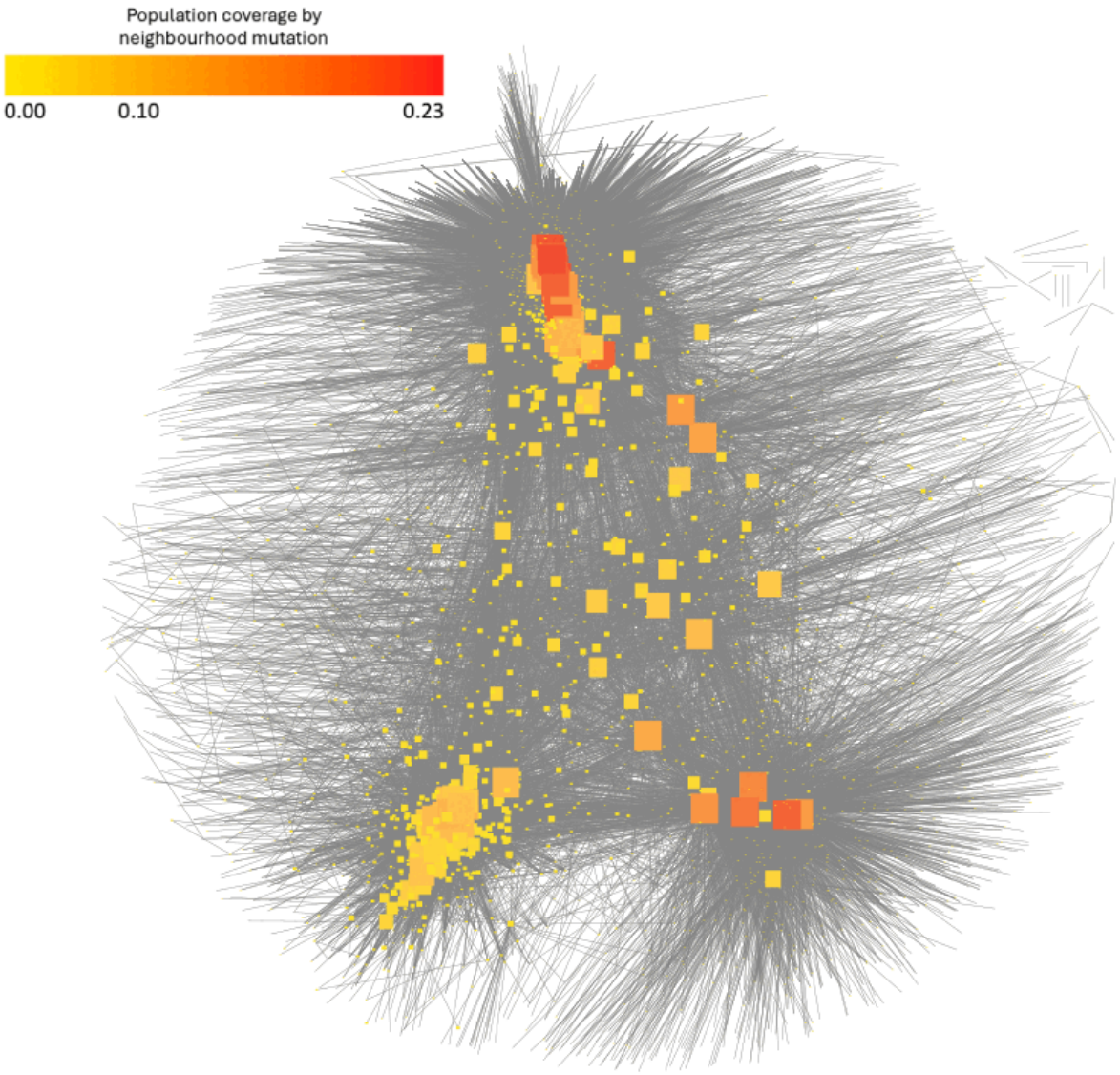

Figure 16: Predicted SL network for Multiple Myeloma. Edges (connections) represent predicted SL relationships from MultiSEp analysis of the MMRF CoMMpass transcriptome data. The network (4,199 genes, 36,595 edges, extitq<0.05) contains 0.035% of the 105,829,426 gene pairs evaluated by MultiSEp. Node (gene) colouring and size indicates the proportion of patients that are predicted to respond to therapeutic inhibition of the candidate target gene. Larger genes and warmer colours correspond to a higher predicted response rate. The smallest genes, with lowest predicted population coverage are shown at the same width and colour as the edges.

5. References

Arafeh, R., Shibue, T., Dempster, J. M., Hahn, C. W., & Vazquez, F. (2025). The present and future of the Cancer Dependency Map. *Nature Reviews Cancer* 25, 59–73. https://doi.org/10.1038/s41568-024-00763-x

Lubbock, A. L. R., Katz, E., Harrison, D. J., & Overton, I. M. (2013). TMA Navigator: network inference, patient stratification and survival analysis with tissue microarray data. *Nucleic Acids Research* 41(W1), W562–W568. https://doi.org/10.1093/nar/gkt529

Skerget, S., Penaherrera, D., Chari, A., Jagannath, S., Siegel, D. S., Vij, R., ... & Keats, J. J. (2024). Comprehensive molecular profiling of multiple myeloma identifies refined copy number and expression subtypes. *Nature Genetics* 56, 1878–1889. https://doi.org/10.1038/s41588-024-01853-0

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B., & Ideker, T. (2003). Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research* 13(11), 2498–2504. https://doi.org/10.1101/gr.1239303

Wappett, M., Harris, A., Lubbock, A. L. R., Lobb, I., McDade, S., & Overton, I. M. (2021). SynLeGG: analysis and visualization of multiomics data for discovery of cancer 'Achilles Heels' and gene function relationships. *Nucleic Acids Research* 49(W1), W613–W618. https://doi.org/10.1093/nar/gkab338