DKOsimR Tutorial

Tutorial: How to generate synthetic CRISPR data using DKOsimR?

Abbreviations: SKO, single knockout; DKO, double knockout; %, percentage; GI, genetic interaction; std. dev., standard deviation.

1. Introduction

This tutorial introduces DKOsimR package for generating synthetic CRISPR double knockout data. It mainly includes:

- Full list of all tunable parameters to initialize the simulation.
- Descriptions on how users may generate synthetic CRISPR data in two modes:
  - default simulation, with 4 gene class: negative, wild-type, non-targeting control, positive
  - simulation applicable to approximate real lab data, with 3 gene class: negative (essential), unknown, non-targeting control
- Examples of applying Genetic Interaction detection method on simulated datasets

2. Install and Load DKOsimR

Simply install and load the package into your R session with following commands:

```
if(!requireNamespace("devtools", quietly = TRUE))
  install.packages("devtools")
devtools::install_github("yuegu-phd/DKOsimR", quiet = TRUE)
#> Installing 1 packages: data.table
library(DKOsimR)
```

Make sure all required dependencies are installed using `devtools::install(dependencies = TRUE)`

3. List of tunable parameters:

Let's go through the designed parameters in simulation framework. Below list out all tunable parameters and how it is represented in DKOsimR.

- Initialized Library Parameters:
  - `sample_name`: name of the simulation run
  - `coverage`: cell representation per guide
  - `n`: number of unique single gene
  - `n_guide_g`: number of guide per gene
  - `moi`: multiplicity of infection - % of cells that are transfected by any virus
  - `sd_freq0`: dispersion of initial counts distribution
- GI Parameters:
  - `p_gi`: % of genetic interaction presence
  - `sd_gi`: std. dev. of re-sampled phenotype with gi presence
- Guide Parameters:
  - `p_high`: % of high-efficacy guides

  - `mode`: CRISPR mode:
    * use `CRISPRn-100%Eff` if need 100% efficient guides without randomization
    * use `CRISPRn` if need high-efficient guides draw from distribution

- Gene Class Parameters:
  1. % of theoretical phenotype to each gene class - make sure they add up to 1

  - `pt_neg`: % negative
  - `pt_pos`: % positive
  - `pt_wt`: % wild-type
  - `pt_ctrl`: % non-targeting control

  2. Mean and std. dev. of theoretical phenotype

  - `mu_neg`: mean of negative genes
  - `sd_neg`: std. dev. of negative genes
  - `mu_pos`: mean of positive genes
  - `sd_pos`: std. dev. of positive genes
  - `sd_wt`: std. dev. of wild-type genes

- Bottleneck Parameters:

  - `size.bottleneck`: bottleneck size - threshold indicating the ceiling of cell growth
  - `n.bottlenecks`: number of bottleneck encounters - how many times do we encountering bottlenecks?
  - `n.iterations`: number of maximum doubling cycles, by default, we assume a maximum of 30 doublings if we didn't encounter bottleneck

- Randomization Parameter:

  - `rseed`: values used for random number generator - use same number to control same sets of genes having GI

4. Running Simulation

To generate synthetic double knockout data, by default, values parameters of simulated CRISPR screens are set based on empirical assumptions as follows:

- Initialized Library Parameters:

  - `coverage`: 100
  - `n_guide_g`: 3
  - `moi`: 0.3
  - `sd_freq0`: 1/3.29 (chosen by setting a 10-fold difference between 95th and 5th percentiles of SKO counts distribution)

- GI Parameters:

  - `p_gi`: 0.03
  - `sd_gi`: 1.5

- Guide Parameters:

  - `p_high`: 1
  - `mode`: CRISPR mode: `CRISPRn-100%Eff`

- Gene Class Parameters: *1. % of theoretical phenotype to each gene class - make sure they add up to 1*

  - `pt_neg`: 0.15
  - `pt_pos`: 0.05
  - `pt_wt`: 0.75
  - `pt_ctrl`: 0.05

2. *Mean and std. dev. of theoretical phenotype*
   - `mu_neg`: -0.75
   - `sd_neg`: 0.1
   - `mu_pos`: 0.75
   - `sd_pos`: 0.1
   - `sd_wt`: 0.25

- Bottleneck Parameters:
  - `size.bottleneck`: 2
  - `n.bottlenecks`: 1
  - `n.iterations`: 30
- Randomization Parameter:
  - `rseed`: NULL

To run simulation by default, simply name your simulation by `sample_name` and specify the number of single genes by `n`. Be cautious that number of genes in each gene class has to be an integer. A quick Simulation Settings Summary would be returned for each run, and the Run Time in the unit of hours would be collected after one successful run. An example running code is

```
dkosim(sample_name="test", n=40)
#>
#> # ------------------------------------------------------------
#> # Simulation Settings Summary:
#> # ------------------------------------------------------------
#> # Sample Name: test
#> # Number of Genes: 40
#> # Cell Library Size (Initial): 714000
#> # Coverage: 100 x
#> # Number of Single Knockout(SKO): 40
#> # Number of Double Knockout(DKO): 780
#> # Number of Guides per Gene: 3
#> # Number of Constructs: 7140
#> # Variance of Initialized Counts: 0.09
#>
#> # Genetic Interactions (GI):
#> ## Proportion of GI(%): 3
#> ## Number of Interacting Gene Pairs: 23
#> ## Variance of re-sampled phenotypes w/ GI: 2.25
#>
#> # Proportion of Each Initialized Gene Class (by theoretical phenotypes):
#> ## Negative(%): 15 ~ TN( -0.75 , 0.01 ,-1,-0.025)
#> ## Positive(%): 5 ~ TN( 0.75 , 0.01 ,0.025, 1)
#> ## Wild-Type(%): 75 ~ TN(0, 0.0625 ,-0.025, 0.025)
#> ## Non-Targeting Control(%): 5 ~ Delta(0)
#>
#> # Proportion of Guides (by efficacy):
#> ## High-efficacy(%): 100 ~ 1
#> ## Low-efficacy(%): 0 ~ TN(0.05, 0.0049, 0, 1)
#>
#> # Multiplicity of Infection (MOI): 0.3
#> # Percentage of viral particles delivered in cells during transfection(%): 22.22 ~ Poisson( 0.3 )
#> # Resampling Size based on MOI (Passage Size): 317367
#> # Bottleneck Size ( 2 x Initial Guide-Level Library Size): 1428000
```

```
#> # Number of Bottleneck Encounters (Number of Passages): 1
#> # Total Available Doublings: 30
#> # Number of Replicates: 2
#> # Pseudo-count: 5e-07
#>
#> # ------------------------------------------------------------
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
#> Loading required package: iterators
#> Loading required package: parallel
#> Warning: package 'data.table' was built under R version 4.3.3
#>
#> Attaching package: 'data.table'
#> The following objects are masked from 'package:dplyr':
#>
#>     between, first, last
#> [1] "repA completed" "repB completed"
#> [1] "number of cores 14"
#> [1] "Run Time (hrs):  0.00348305555555556"
```

Alternatively, you may adjust values to any tunable parameters as desired, but please make sure your input on percentage of each gene class add up to 1 for all classes, and each initialized number of genes is an integer. You may also change the output directory using `path` in the function; by default, the output simulated data and log is under the same directory of current project workspace. The randomization seed can also be specified by `rseed` to ensure same subsets of gene-pairs has GI in multiple run.

An example running code is

```
dkosim(sample_name="test",
       coverage=10,
       n=60,
       n_guide_g=2,
       sd_freq0 = 1/3.29,
       moi = 0.3,
       p_gi=0.03,
       sd_gi=1.5,
       p_high=1,
       mode="CRISPRn-100%Eff",
       pt_neg=0.15,
       pt_pos=0.05,
       pt_wt=0.75,
       pt_ctrl=0.05,
       mu_neg=-0.75,
       sd_neg=0.1,
       mu_pos=0.75,
       sd_pos=0.1,
       sd_wt=0.25,
       size.bottleneck = 3,
       n.bottlenecks= 2,
```

```
       n.iterations = 30,
       rseed = 111,
       path = ".")
#>
#> # ------------------------------------------------------------
#> # Simulation Settings Summary:
#> # ------------------------------------------------------------
#> # Sample Name: test
#> # Number of Genes: 60
#> # Cell Library Size (Initial): 72000
#> # Coverage: 10 x
#> # Number of Single Knockout(SKO): 60
#> # Number of Double Knockout(DKO): 1770
#> # Number of Guides per Gene: 2
#> # Number of Constructs: 7200
#> # Variance of Initialized Counts: 0.09
#>
#> # Genetic Interactions (GI):
#> ## Proportion of GI(%): 3
#> ## Number of Interacting Gene Pairs: 53
#> ## Variance of re-sampled phenotypes w/ GI: 2.25
#>
#> # Proportion of Each Initialized Gene Class (by theoretical phenotypes):
#> ## Negative(%): 15 ~ TN( -0.75 , 0.01 ,-1,-0.025)
#> ## Positive(%): 5 ~ TN( 0.75 , 0.01 ,0.025, 1)
#> ## Wild-Type(%): 75 ~ TN(0, 0.0625 ,-0.025, 0.025)
#> ## Non-Targeting Control(%): 5 ~ Delta(0)
#>
#> # Proportion of Guides (by efficacy):
#> ## High-efficacy(%): 100 ~ 1
#> ## Low-efficacy(%): 0 ~ TN(0.05, 0.0049, 0, 1)
#>
#> # Multiplicity of Infection (MOI): 0.3
#> # Percentage of viral particles delivered in cells during transfection(%): 22.22 ~ Poisson( 0.3 )
#> # Resampling Size based on MOI (Passage Size): 48005
#> # Bottleneck Size ( 3 x Initial Guide-Level Library Size): 216000
#> # Number of Bottleneck Encounters (Number of Passages): 2
#> # Total Available Doublings: 30
#> # Number of Replicates: 2
#> # Pseudo-count: 5e-06
#>
#> # ------------------------------------------------------------
#>
#> [1] "repA completed" "repB completed"
#> [1] "number of cores 14"
#> [1] "Run Time (hrs):  0.0018969444444444"
```

5. Laboratory Data Pattern Approximation

To utilize the simulation framework in approximating actual laboratory data pattern, `dkosim_lab` was designed to include three initialized gene class. You may specify the percentage of essential genes by `pt_neg`, unknown by `pt_unknown`, and non-targeting controls by `pt_ctrl` accordingly. In our designed framework, essential genes is defined by negative genes; unknown genes might compose of genes with theoretical phenotype

in either negative, positive, wild-type, or non-targeting control gene class.

By default, `dkosim_lab` run simulation with following parameter

- Initialized Library Parameters
  - `coverage`: 100
  - `n_guide_g`: 3
  - `moi`: 0.3
  - `sd_freq0`: 1/2.56 (chosen by setting a 10-fold difference between 90th and 10th percentiles of SKO counts distribution)
- GI Parameters:
  - `p_gi`: 0.03
  - `sd_gi`: 1.5
- Guide Parameters:
  - `p_high`: 0.75
  - `mode`: CRISPR mode: `CRISPRn`
- Gene Class Parameters:
  1. *% of theoretical phenotype to each gene class - make sure they add up to 1*
     - `pt_neg`: 0.15
     - `pt_unknown`: 0.80
     - `pt_ctrl`: 0.05
  2. *Mean and std. dev. of theoretical phenotype*
     - `mu_neg`: -0.03
     - `sd_neg`: 0.25
     - `sd_unknown`: 0.25
- Bottleneck Parameters:
  - `size.bottleneck`: 2
  - `n.bottlenecks`: 1
  - `n.iterations`: 30
- Randomization Parameter:
  - `rseed`: NULL

An example running code is

```
dkosim_lab(sample_name="test_lab", n=20)
#>
#> # ------------------------------------------------------------
#> # Simulation Settings Summary:
#> # ------------------------------------------------------------
#> # Sample Name: test_lab
#> # Number of Genes: 20
#> # Cell Library Size (Initial): 177000
#> # Coverage: 100 x
#> # Number of Single Knockout(SKO): 20
#> # Number of Double Knockout(DKO): 190
#> # Number of Guides per Gene: 3
#> # Number of Constructs: 1770
#> # Variance of Initialized Counts: 0.15
#>
```

```
#> # Genetic Interactions (GI):
#> ## Proportion of GI(%): 3
#> ## Number of Interacting Gene Pairs: 6
#> ## Variance of re-sampled phenotypes w/ GI: 2.25
#>
#> # Proportion of Each Initialized Gene Class (by theoretical phenotypes):
#> ## Negative/Essential(%): 15 ~ TN( -0.03 , 0.0625 ,-1,-0.025)
#> ## Unknown(%): 80 ~ TN(0, 0.0625 ,-0.5, 0.5)
#> ## Non-Targeting Control(%): 5 ~ Delta(0)
#>
#> # Proportion of Guides (by efficacy):
#> ## High-efficacy(%): 75 ~ TN(0.9, 0.1, 0.6, 1)
#> ## Low-efficacy(%): 25 ~ TN(0.05, 0.0049, 0, 0.6)
#>
#> # Multiplicity of Infection (MOI): 0.3
#> # Percentage of viral particles delivered in cells during transfection(%): 22.22 ~ Poisson( 0.3 )
#> # Resampling Size based on MOI (Passage Size): 78675
#> # Bottleneck Size ( 2 x Initial Guide-Level Library Size): 354000
#> # Number of Bottleneck Encounters (Number of Passages): 1
#> # Total Available Doublings: 30
#> # Number of Replicates: 2
#> # Pseudo-count: 5e-06
#>
#> # ------------------------------------------------------------
#>
#> [1] "repA completed" "repB completed"
#> [1] "number of cores 14"
#> [1] "Run Time (hrs):  0.000696388888888889"
```

Alternatively, you may adjust values to any tunable parameters as desired, or using the parameters described in the full article to approximate actual lab data pattern in double CRISPR screens.

6. Applying Genetic Interaction Detection Methods on simulated data

Here, we show how we may apply dLFC (Dede et al., 2020) to calculate GI on simulated data, as example to analyze the simulation output.

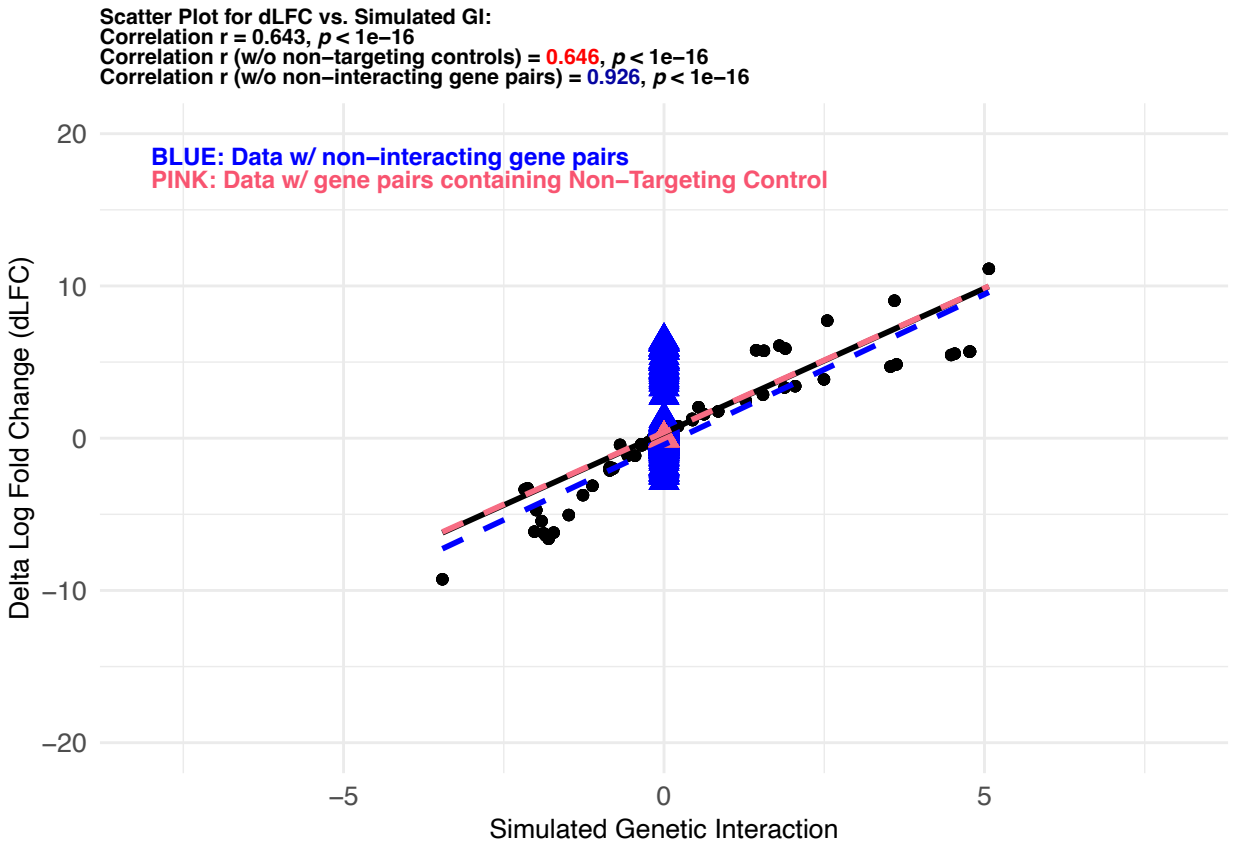
```
knitr::opts_knit$set(root.dir = getwd())
# read example simulated data
sample_name = "example_data"
pseudo_counts = 5e-07
repA_path <- here::here("data", paste0(sample_name, "_repA.csv"))
repB_path <- here::here("data", paste0(sample_name, "_repB.csv"))

# each simulation is containing two replicates A and B, and we take average of the LFC for analysis
cell_lib_guide2_A = read.csv(repA_path) %>%
  dplyr::select(-X)

cell_lib_guide2_B = read.csv(repB_path) %>%
  dplyr::select(-X) %>%
  dplyr::select(guide_id, counts_guide_t0,
                counts_guide_t1, counts_guide_t2,
                rel_freq_guide_t0, rel_freq_guide_t2, LFC)
```

```r
cell_lib_guide2 = merge(cell_lib_guide2_A, cell_lib_guide2_B, by = "guide_id") %>%
  dplyr::rename(counts_guide_t0_1 = counts_guide_t0.x,
                counts_guide_t0_2 = counts_guide_t0.y,
                counts_guide_t1_1 = counts_guide_t1.x,
                counts_guide_t1_2 = counts_guide_t1.y,
                counts_guide_t2_1 = counts_guide_t2.x,
                counts_guide_t2_2 = counts_guide_t2.y,
                rel_freq_guide_t0_1 = rel_freq_guide_t0.x,
                rel_freq_guide_t0_2 = rel_freq_guide_t0.y,
                rel_freq_guide_t2_1 = rel_freq_guide_t2.x,
                rel_freq_guide_t2_2 = rel_freq_guide_t2.y,
                LFC_t2_1 = LFC.x,
                LFC_t2_2 = LFC.y) %>%
  mutate(guide1_type = ifelse(guide1_type == 1, "high", "low"),
         guide2_type = ifelse(guide2_type == 1, "high", "low"),
         construct_type = ifelse(is.na(gene2_behavior), gene1_behavior,
                                 paste0(gene1_behavior, " + ", gene2_behavior)),
         LFC_t2_1 = log2(((rel_freq_guide_t2_1 + pseudo_counts) /
                            (rel_freq_guide_t0_1 + pseudo_counts))),
         LFC_t2_2 = log2(((rel_freq_guide_t2_2 + pseudo_counts) /
                            (rel_freq_guide_t0_2 + pseudo_counts))))%>%
  mutate(LFC = (LFC_t2_1 + LFC_t2_2)/2) # aggregate the LFC between 2 replicates by averaging


# read the simulated data and prepare sample name
sim_data = cell_lib_guide2 %>%
  mutate(construct_type = ifelse(KO_type == "SKO", "SKO",
                                 ifelse(gene1_behavior == "Non-Targeting Control"
                                        | gene2_behavior == "Non-Targeting Control", "SKO", "DKO" )))

nc_gene = unique(as.character(filter(sim_data, gene1_behavior == "Non-Targeting Control")$gene1))

# calculate single mutant/knockout fitness
gene_SMF = sim_data %>%
  filter(construct_type == "SKO", gene1_behavior != "Non-Targeting Control") %>%
  group_by(gene1) %>%
  dplyr::summarise(SMF = mean(LFC))
# calculate mean LFC for gene pairs that don't contain control
gene_LFC = sim_data %>%
  filter(construct_type == "DKO") %>%
  group_by(gene_pair_id) %>%
  dplyr::summarise(LFC_mean = mean(LFC))
# calculate dLFC = LFC_mean - (SMF1 + SMF2)
sim_data_dLFC = left_join(sim_data, gene_SMF, by = "gene1") %>%
  dplyr::rename(gene1_SMF = SMF) %>%
  left_join(gene_SMF, by = c("gene2"="gene1")) %>%
  dplyr::rename(gene2_SMF = SMF) %>%
  left_join(gene_LFC, by = "gene_pair_id") %>%
  mutate(gene1_SMF = ifelse(is.na(gene1_SMF), 0, gene1_SMF),
         gene2_SMF = ifelse(is.na(gene2_SMF), 0, gene2_SMF),
         LFC_mean = ifelse(construct_type == "SKO",
                           ifelse(gene1_SMF == 0, gene2_SMF, gene1_SMF), LFC_mean)) %>%
  mutate(dLFC = LFC_mean - (gene1_SMF+gene2_SMF))
```

```r
# add data frame for the non-control group
control = sim_data_dLFC %>% filter(gene1 %in% nc_gene | gene2 %in% nc_gene)
sim_dLFC_noncontrol = sim_data_dLFC %>% anti_join(control, by = "gene1_gene2_id")
# add data frame for the non-interacting group
control0 = sim_data_dLFC %>% filter(interaction_gene == 0)
sim_dLFC_noncontrol0 = sim_data_dLFC %>% anti_join(control0, by = "gene1_gene2_id")

# compute correlation and p-values
cor_test_all <- cor.test(sim_data_dLFC$interaction_gene,
                         sim_data_dLFC$dLFC, method = "pearson")
cor_test_noncontrol <- cor.test(sim_dLFC_noncontrol$interaction_gene,
                                sim_dLFC_noncontrol$dLFC, method = "pearson")
cor_test_noncontrol0 <- cor.test(sim_dLFC_noncontrol0$interaction_gene,
                                 sim_dLFC_noncontrol0$dLFC, method = "pearson")
# show correlation value
print(cor_test_all)
#>
#>  Pearson's product-moment correlation
#>
#> data:  sim_data_dLFC$interaction_gene and sim_data_dLFC$dLFC
#> t = 106.64, df = 16108, p-value < 2.2e-16
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.6341445 0.6522501
#> sample estimates:
#>       cor
#> 0.6432872
print(cor_test_noncontrol)
#>
#>  Pearson's product-moment correlation
#>
#> data:  sim_dLFC_noncontrol$interaction_gene and sim_dLFC_noncontrol$dLFC
#> t = 102.06, df = 14533, p-value < 2.2e-16
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.6365640 0.6555061
#> sample estimates:
#>       cor
#> 0.6461345
print(cor_test_noncontrol0)
#>
#>  Pearson's product-moment correlation
#>
#> data:  sim_dLFC_noncontrol0$interaction_gene and sim_dLFC_noncontrol0$dLFC
#> t = 53.569, df = 475, p-value < 2.2e-16
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.9123672 0.9380457
#> sample estimates:
#>       cor
#> 0.9262742
```

Visualization on scatterplot to the calculated GI in the example

```
#> Ignoring unknown labels:
#> * colour : "Gene-gene Interaction Type"
#> `geom_smooth()` using formula = 'y ~ x'
#> `geom_smooth()` using formula = 'y ~ x'
#> `geom_smooth()` using formula = 'y ~ x'
```



Reference

Gu, Y., Hart, T., Leon-Novelo, L., Shen, J.P.. Double-CRISPR Knockout Simulation (DKOsim): A Monte-Carlo Randomization System to Model Cell Growth Behavior and Infer the Optimal Library Design for Growth-Based Double Knockout Screens. bioRxiv 2025.09.11.675497. DOI: 10.1101/2025.09.11.675497.

Dede, M., McLaughlin, M., Kim, E., & Hart, T. (2020). Multiplex enCas12a screens detect functional buffering among paralogs otherwise masked in monogenic Cas9 knockout screens. Genome biology, 21(1), 262. DOI: 10.1186/s13059-020-02173-2