Plan for `sceptre` implementation

This document outlines a plan for implementing `sceptre`. I propose two separate interfaces. First, the `sceptre` Nextflow pipeline implements `sceptre` via Nextflow and `ondisc`; this interface is appropriate for moderately-sized to large data. Next, the `sceptre` R studio interface implements `sceptre` exclusively within R; this latter interface is appropriate for small data. I begin by describing a design plan for the former interface.

1 `sceptre` Nextflow pipeline

The sceptre Nextflow pipeline that I propose is general: it handles both low and high MOI data and is able to carry out both undercover and discovery analyses. This generality enables us to build a single Nextflow pipeline, sidestepping the need to create multiple pipelines that carry out different tasks. First, I describe the arguments (passed via command line) of the Nextflow pipeline.

Nextflow pipeline arguments

The command line arguments are as follows (roughly ordered from most important to least important). The most critical arguments are 1-7.

- `multimodal_odm` (*required*). The multimodal ODM containing the data. The multimodal ODM should contain two modalities: first, the response modality, and second, the gRNA modality. The response modality should contain the IDs of the response features as row names. The gRNA modality, meanwhile, should contain the IDs of the individual gRNAs as row names. Additionally, the gRNA modality should contain a feature covariate data frame containing (i) the gRNA group to which each individual gRNA belongs (if an ungrouped analysis is desired, then each gRNA should be put into a group of size one) and (ii) the type (either "targeting" or "non-targeting") of each gRNA. The gRNA modality typically should be a matrix of UMI counts; however, if the user wishes to assign gRNAs to cells herself/himself, then the gRNA modality alternately can be a logical matrix. Finally, the multimodal ondisc matrix should contain a "global cell covariates" field containing cell-specific covariates. The multimodal ODM is passed as

three separate arguments: the metadata.rds file, the response modality backing .odm file, and the gRNA modality backing .odm file.

The multimodal ODM also should contain a `formula` and `moi` in its `global_misc` field. `formula` is an R formula object describing how to adjust for the covariates. The terms in the formula should be columns of the global cell covariate matrix. Next, `moi` is a string indicating the MOI of the data (either "low" or "high").

- `response_modality_name` and `grna_modality_name` (*required*). The name of the response modality (typically "gene") and gRNA modality (typically "grna_expression") within the multimodal ODM.

- `undercover` (*required*). A string (taking value either "true" or "false") indicating whether to run an undercover analysis ("true") or a discovery analysis ("false").

- `result_fp` (*required*). The name of the file in which to store the results (should be a full file path).

- `pairs_to_analyze` (*optional*). A data frame giving the set of response-gRNA group pairs to analyze. If no argument is supplied, then all pairs are analyzed. If undercover is set to "true," this argument is ignored (and negative control pairs are automatically constructed and analyzed).

- `resampling_mech` (*optional*; default "marginal" for low MOI and "conditional" for high MOI). The resampling mechanism to use. By default, this argument is set to "marginal" for low MOI data and "conditional" for high MOI data.

- `test_stat` (*optional*; default "full"). A string indicating the test statistic to use (either "full" or "distilled"); the full statistic typically is more powerful but a bit slower to compute.

- `gene_precomputations` (*optional*). The matrix of gene precomputations. This matrix may have been obtained from a previous run of the pipeline and can be passed to reduce compute. However, this is entirely optional.

- `side` (*optional*; default "both"). A string (taking the value "both," "left," or "right") indicating the sidedness of the test.
- `effective_ss_thresh` (*optional*; default 7). The threshold for the number of treatment cells with nonzero expression that a pair must have to be analyzed.
- B_1 (*optional*; default 5,000). Number of resampled test statistics to use in the first round of adaptive permutation testing.
- B_2 (*optional*; default 250,000). Number of resampled test statistics to use in the second round of adaptive permutation testing.
- `p_thresh` (*optional*; default 0.01). P-value cutoff used to determine whether to proceed to the second round of permutation testing.
- `gRNA_thresh` (*optional*; default 3). Threshold to use to assign gRNAs to cells (relevant only for high MOI data).
- Parameters related to the the computation itself (e.g., number of gene pods, number of gRNA pods, number of pair pods, amount of time per process, amount of memory per process). Currently, `gene_pod_size` and `pair_pod_size`.
- Parameters related to the undercover pipeline. In particular, the number of pairs (`n_undercover_pairs`) and the group size (`undercover_grp_size`).

Nextflow pipeline output

The pipeline outputs a data frame with the following columns: response ID, gRNA group, p-value, log fold change, z-score, N treatment cells with nonzero expression, and N control cells with nonzero expression. The pipeline also outputs the matrix of gene pre-computations. (The gene pre-computations coincide across undercover and discovery analyses, so if the user first runs an undercover analysis, s/he can then run a discovery analysis using the gene pre-computations obtained from the undercover analysis to save compute.)

Process 1: Argument checking, data processing, and QC

Next, I describe the processes that make up the Nextflow pipeline. The first process performs some basic argument checking and data processing. There are several steps.

1. *Argument checking.* We check the input arguments for correctness and print informative error messages if there are problems. For example, the response ID column in the pairs to analyze data frame must be a subset of the row names of the response modality, etc.

2. *Multimodal ODM processing.* We perform some light processing of the cell covariate matrix. In particular, we apply the formula object to convert the cell covariate matrix into a design matrix that can be directly used in the regression. We also update the names of the response modality, gRNA modality, and gRNA group column. Finally, we "thin" the multimodal ODM, removing all unnecessary information.

3. *QC.* We apply a bit of automated cell-wise QC to the data. (Note: this QC sits on top of any QC that user might have applied before invoking the pipeline.) Our QC might include (i) removing cells with high mitochondrial gene content, (ii) removing cells with aberrantly high or low gene expression, and for low MOI data, (iii) removing cells that have high UMI counts across multiple gRNAs.

4. *Assigning gRNAs to cells (low MOI only).* We assign a single gRNA to each cell via the maximum operation. When "undercover" is set to "true," we map each individual NT gRNA to the indices of the cells in which that NT gRNA is present. We also map the label "non-targeting" to the set of cells that received a non-targeting gRNA. When "undercover" is set to "false," by contrast, we map each gRNA *group* to the set of cells in which any constituent gRNA of the group is present; we combine the NT gRNAs into a single "group." We save this map (implemented as a named list) in the "misc" field of the gRNA ODM.

5. *Generating the set of pairs to analyze.* If no `pairs_to_analyze` data frame has been specified, then all gRNA groups are paired to all genes. If `undercover` has been set to `true`, then we generate negative control

pairs. To this end, NT gRNAs are randomly assigned to groups of the requested size, and these groups are paired to genes. gRNA group-gene pairs are downsampled so that the number of negative control pairs is equal to the number that the user requests. Importantly, we sort the data frame of pairs first by gRNA group and then by gene.

6. *Assigning responses and pairs to pods.* We assign a "pod ID" to each response, gRNA, and pair.

We output the pod information and updated multimodal ODM from this process.

Process 2: Gene pre-computation

The second process carries out the gene pre-computations. We only perform this process if the gene pre-computations have not been supplied. We parallelize over gene pods. For a given pod, we iterate over genes within the process. There are two steps.

1. First, if we are in low MOI, we subset the design matrix. To this end, we obtain the indices of the NT cells (from the gRNA ODM) and subset the design matrix according to these indices.

2. We loop over the genes, load the expression vector of a given gene into memory. If in low MOI, we subset NT cells.

3. We regress the expression vector onto the covariate matrix. We output the fitted regression coefficients and size parameter.

4. Save the fitted regression coefficients in a gene-by-coefficient matrix.

The process outputs the matrix of fitted regression coefficients.

Process 3: Combine the gene pre-computations

We assemble the matrices of fitted regression coefficients across pods into a single matrix.

Process 4: Gene-to-gRNA association tests

The fourth process is to carry out the gene-to-gRNA group association tests. We proceed gRNA group-by-gene, testing for association between a given gRNA group and all genes. Thus, fix a given gRNA group.

Step 1: Update the gRNA ODM and pairs to analyze

If we are running an undercover analysis, then the gRNA ODM must be updated. If in high MOI, we change the target type of the input NT gRNAs to "targeting" and assign them to a group "undercover." If in low MOI, we assign the input NT gRNAs to a group "undercover" and update the indices of the "non-targeting" group to exclude the undercover gRNAs. We design a function `send_odm_undercover` to convert the input gRNA ODM (`grna_odm`) into an undercover gRNA ODM. We call this function if we are conducting an undercover analysis; otherwise, we skip this function. (See code block below for an example implementation; `nt_grnas` refers to the NT gRNA(s) that are to be sent undercover.)

```
if (undercover) {
curr_grna_odm <- send_odm_undercover(grna_odm, nt_grnas)
curr_grna_group <- ‘‘undercover’’
} else {
curr_grna_odm <- grna_odm
curr_grna_group <- grna_group
}
```

Step 2: Obtain the observed indices, subset the covariate matrix

The second step is to obtain the set of observed indices (i.e., the indices of the cells in which the treatment gRNA group is present). In high MOI this is simple: we load the indicator vector for the current gRNA group and obtain the indices of the cell in which the gRNA group is present. In low MOI we need two pieces of information: first, the indices to subset the gene expression vector (the so-called `subset_idxs`), and second, the indices of the treatment cells (`treatment_idxs`) *within* the `subset_idxs`. We can obtain these two pieces of information from the list of gRNA group indices within the gRNA ODM. In the low MOI case we subset the global cell covariate matrix according to the `subset_idxs`.

Step 3: Obtain the synthetic indices

The next step is to obtain the list of synthetic gRNA indicators. We design a function, `generate_synthetic_grna_indicators`, to carry out this task. The function takes as input (i) the covariate matrix, (ii) the vector of observed gRNA indicator indices, (iii) the total number of cells (which varies across low and high MOI), and (iv) the number of vectors to sample. It then generates a list of synthetic gRNA indicators. Let n_{treat} be the number of treatment cells (i.e., the number of cells that received the targeting gRNA). Let n_{C} be the number of cells that received an NT gRNA, and let n_{tot} be the total number of cells. We handle the following four cases.

- **Marginal resampling, low MOI**. We sample n_{treat} integers (without replacement) from the set $\{1, \ldots, n_{\text{C}} + n_{\text{treat}}\}$ B_1 times.
- **Marginal resampling, high MOI**. We sample n_{treat} integers (without replacement) from the set $\{1, \ldots, n_{\text{tot}}\}$ B_1 times.
- **Conditional resampling, low MOI**. Fit a logistic regression of gRNA presence/absence onto the technical factors (using only the treatment cells and the NT cells). Sample B_1 gRNA indicator vectors from the fitted logistic regression model. Store only the indices of the ones in these vectors.
- **Conditional resampling, high MOI**. Similar to above, but use all the cells to fit the logistic regression model.

We generate B_1 synthetic gRNA indicator vectors. We append the vector of observed gRNA indicators to the end of the list of synthetic gRNA indicators. (NOTE: We could instead try to run a separate gRNA pre-computation process to fit the logistic regression models, but I think that the challenges associated with programming this are not worth the mild computational benefits.)

Step 4: Get the necessary information for a given gene (i.e., expressions and fitted means)

We load a given gene's expression vector into memory. If we are in low MOI, we subset the gene so that we are working only with treatment and control cells (in other words, we subset the expression vector according to `subset_idxs`). We then recompute this gene's fitted means using its fitted regression coefficients and the covariate matrix.

Step 5: P-value calculation and log-fold change estimate (round 1 inner loop)

We next calculate the log-fold change and compute the initial p-value p_1 for the pair. We pass all information necessary to compute the vector of test statistics — namely, the fitted means, the fitted size parameter, the raw expressions, the design matrix, and the list of gRNA indexes — into a function to compute the test statistics. There are two such functions: the first computes distilled score statistics, and the second full score statistics. The function returns a vector of B_1+1 statistics (the first B_1 of which are the null statistics, and the last of which is the observed statistic). Using these statistics we compute an empirical p-value p_1. We also compute an estimate of the log-fold change using the distilled model.

Step 6: Determine which p-values to recompute, and generate the second round of synthetic gRNA indices (outer loop)

We examine the pairwise p-values that we computed in the previous stage. We identify the pairs whose p-value falls below `p_thresh`. If at least one pair has a p-value below this threshold, then we compute a list of synthetic gRNA indices (following Step 1) of length B_2.

Step 7: P-value calculation (round 2 inner loop)

Let a gene be given. We load this gene's expressions into memory and then recompute its fitted means (following Step 3). We then compute its vector of B_2 null test statistics. Finally, we compute its p-value by comparing the original test statistics to the vector of null test statistics, fitting a GPD to the tail of the empirical resampling distribution. (Gene's function will go here.)

Step 8: Updating the results (outer loop)

Finally, we update our results data frame, replacing the initial p-value with the more precise p-value for those pairs for which we computed a more precise p-value. We then combine the results data frames across gRNAs and output this aggregated data frame from the process.

Process 5: Combine results

Finally, we combine the results across all pair pods to produce the final output.