METHOD ARTICLE

REVISED An R-based reproducible and user-friendly preprocessing pipeline for CyTOF data [version 2; peer review: 2 approved]

Helena L. Crowell[1,2*], Stéphane Chevrier[3*], Andrea Jacobs[3], Sujana Sivapatham[3], Tumor Profiler Consortium, Bernd Bodenmiller[3*], Mark D. Robinson[1,2*]

[1]Institute of Molecular Life Sciences, University of Zurich, Zurich, 8057, Switzerland
[2]SIB Swiss Institute of Bioinformatics, Zurich, 8057, Switzerland
[3]Department of Quantitative Biomedicine, University of Zurich, Zurich, 8057, Switzerland

* Equal contributors

v2 **First published:** 22 Oct 2020, **9**:1263
https://doi.org/10.12688/f1000research.26073.1
Latest published: 08 Aug 2022, **9**:1263
https://doi.org/10.12688/f1000research.26073.2

Abstract

Mass cytometry (CyTOF) has become a method of choice for in-depth characterization of tissue heterogeneity in health and disease, and is currently implemented in multiple clinical trials, where higher quality standards must be met. Currently, preprocessing of raw files is commonly performed in independent standalone tools, which makes it difficult to reproduce. Here, we present an R pipeline based on an updated version of CATALYST that covers all preprocessing steps required for downstream mass cytometry analysis in a fully reproducible way. This new version of CATALYST is based on Bioconductor's SingleCellExperiment class and fully unit tested. The R-based pipeline includes file concatenation, bead-based normalization, single-cell deconvolution, spillover compensation and live cell gating after debris and doublet removal. Importantly, this pipeline also includes different quality checks to assess machine sensitivity and staining performance while allowing also for batch correction. This pipeline is based on open source R packages and can be easily be adapted to different study designs. It therefore has the potential to significantly facilitate the work of CyTOF users while increasing the quality and reproducibility of data generated with this technology.

Keywords

CyTOF, Preprocessing, Normalization, Debarcoding, Compensation, Gating, Batch correction, Reproducibility

Open Peer Review

Approval Status ✓ ✓

	1	2
version 2 (revision) 08 Aug 2022	✓ view	
version 1 22 Oct 2020	? view	✓ view

1. **Marie Trussart**, Walter and Eliza Hall Institute of Medical Research, Parkville, Australia

2. **Felix Hartmann**, Stanford University, Palo Alto, USA
Sean C. Bendall, Stanford University, Stanford, USA

Any reports and responses or comments on the article can be found at the end of the article.

This article is included in the RPackage gateway.

This article is included in the Bioconductor gateway.

Corresponding author: Helena L. Crowell (helena.crowell@uzh.ch)

Author roles: Crowell HL: Conceptualization, Formal Analysis, Methodology, Software, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; **Chevrier S**: Conceptualization, Data Curation, Formal Analysis, Investigation, Project Administration, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; **Jacobs A**: Investigation, Writing – Review & Editing; **Sivapatham S**: Investigation, Writing – Review & Editing; **Bodenmiller B**: Funding Acquisition, Project Administration, Writing – Review & Editing; **Robinson MD**: Funding Acquisition, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: MDR and HLC were supported by the Swiss National Science Foundation [310030_175841, CRSII5_177208], and MDR acknowledges support from the University Research Priority Program Evolution in Action at the University of Zurich. SC, AJ, SS and BB were jointly funded by a public-private partnership involving Roche, ETH Zurich, University of Zurich, University Hospital Zurich, and University Hospital Basel.

How to cite this article: Crowell HL, Chevrier S, Jacobs A *et al.* **An R-based reproducible and user-friendly preprocessing pipeline for CyTOF data [version 2; peer review: 2 approved]** F1000Research 2022, **9**:1263 https://doi.org/10.12688/f1000research.26073.2

First published: 22 Oct 2020, **9**:1263 https://doi.org/10.12688/f1000research.26073.1

REVISED **Amendments from Version 1**

Minor text revisions have been made to clarify and be consistent in the terminology used (e.g., acquisition vs. run vs. batch). We have further adapted reference-related metadata to systematically include 7 previously acquired experiments (rather than sometimes 7, other times 8). Under "Batch alignment", we have also included a figure to compare the expression distributions before and after batch correction.

In order to improve the workflow's usability, various updates have been made to CATALYST. E.g., to enable analysis of samples with panel discrepancies, the prepData() function now supports flexible mechanisms to consolidate these (e.g., via a table of channel aliases, or by renaming channels according to a reference sample). Furthermore, the normCytof() function now has an additional argument to specify custom DNA channel masses. Accordingly, the workflow now uses most current software versions, namely, R v4.2 with Bioconductor v3.15.

Any further responses from the reviewers can be found at the end of the article

Introduction

Over the past decade, mass cytometry (CyTOF) has advanced our understanding of a wide range of cellular processes, particularly in the field of immunology and tumor biology[1,2], by enabling the simultaneous measurement of 40+ parameters at the single cell level. Currently, mass cytometry is transitioning from an exploratory research approach toward a diagnostic tool used in clinical laboratories and this transition is associated with an increased need for standardization[3]. Various studies have already suggested improvements on the experimental workflows to increase the robustness of mass cytometry data by working with frozen antibody cocktails or by including shared reference samples in each independent experiment to enable for batch correction[4,5]. Similarly, advanced downstream analyses benefit from the large number of analysis tools and algorithms implemented in R, which allow for fully reproducible analyses[6].

Between data generation and downstream data analysis, data preprocessing is an multi-step procedure required to convert raw FCS files into data objects that can be input to downstream statistical analysis and visualization[7]. Upon data collection, the first step consists in concatenating files from sequential CyTOF acquisitions and removing events with unstable signal, which are usually caused by uneven flow rate or introduction of air in the fluidic system. As a second step, CyTOF data need to be corrected for time dependent signal drift, which is mostly due to cone contamination, mass calibration drift or loss of detector sensitivity over time. This correction is performed by acquiring metal tagged polystyrene beads together with the cell suspension, where bead signals can be used as a reference to normalize the cell signals[8]. Another potential artefact in CyTOF data is due to signal spillover between channels. Although lower than what is usually observed in fluorescent flow cytometry, spillover in mass cytometry can still account for up to 4% of the signal in some channels and needs to be corrected using signal compensation[9]. Sample barcoding prior to staining is a common approach used in mass cytometry to combine multiple samples in a single experiment to minimize experimental variation due to staining and CyTOF acquisition. In this case, individual cells have to be assigned to their respective sample via a process called single cell debarcoding[10]. In large studies where samples are collected over a long period of time by different users, on different machines or at different sites, an important step is to correct for batch effects, which can be achieved by including a shared control sample in each independent batch[11,12]. Finally, only live, intact single cells are relevant for the downstream analysis. Beads, doublets, debris and dead cells are excluded by gating on scatter plots[7].

Each step of the preprocessing pipeline requires expert decisions to determine the best parameters to achieve an optimal signal correction and cell selection. Moreover, all the chosen parameters should be recorded for reproducibility purposes. Despite these requirements, many current preprocessing pipelines still rely on switching between platforms that include, for example, MATLAB applications and (at least partially) closed source online platforms (e.g., Cytobank[13]). This approach necessitates uploading the data to different platforms and carrying out certain steps in a purely manual fashion, which makes it time-consuming and difficult to reproduce. This is particularly limiting in a clinical setting, where reproducibility and large-scale data analysis are required. Thus, we propose a semi-automated R-based preprocessing pipeline for CyTOF data that is: i) fully reproducible; ii) includes quality checks and, iii) has limited need for supervision once the original setup has been made. This pipeline is developed around an updated version of `CATALYST`, an R package designed for preprocessing and differential analysis of mass cytometry data[9,14]. This new version of `CATALYST` is based on Bioconductor's `SingleCellExperiment` class, the standard for high dimensional single cell data analysis. This pipeline can easily be adapted to each CyTOF user's needs and will accelerate CyTOF data preprocessing while improving the quality of mass cytometry data generated.

Data description

The data used in this pipeline were generated in the context of the Tumor Profiler project, a multi-center observational study investigating the relevance of different innovative technologies, including CyTOF, imaging mass

cytometry, single-cell DNA and RNA sequencing, as well as *ex vivo* drug testing to improve the diagnostic of advanced cancer patients[15].

The samples of interest included tumor biopsies and blood samples collected at the University Hospital Zurich in spring 2020. These samples were assessed by mass cytometry in the context of a set of references including commercially available cell lines, PBMCs from healthy donors and PHA activated PBMCs. PBMCs from patients and healthy donors were collected based on a ficoll gradient[16], and tumor samples were dissociated as previously described[17]. Once in single-cell suspension, all samples were stained for 5 min on ice with Cell-ID™ Cisplatin-194Pt (#201194, Standard BioTools) to identify dead cells and subsequently fixed with PFA 1.6% (#15710, Electron Microscopy Sciences). Samples were stored as dry pellet at −80°C until CyTOF measurement.

The dataset used in this study was obtained from a single CyTOF experiment, also called batch, where nine references, two blood samples and two tumor samples were barcoded with a 20-well barcoding plate[17]. Reference samples were selected to contain positive and negative populations for each marker included in the study's antibody panel. This design was chosen to enable for quality control and batch correction across independent experiments based on quantile scaling as described in 11. Pooled cells were stained with a 40-Ab panel designed to perform an in-depth characterization of the samples' immune compartment. DNA intercalation was performed with a 1h incubation in Cell-ID™ Intercalator-Ir (#201192B, Fluidigm). Finally, the cell suspension was diluted 1:10 in Maxpar® Cell Acquisition Solution (#201240, Fluidigm) and 10% of EQ Four Element Calibration Beads (#201078, Fluidigm), and acquired on a Helios™ upgraded CyTOF 2 system at a flow rate of 150 events per second.

Throughout this workflow, we will make use of a set of metadata for standard preprocessing steps (normalization, debarcoding and compensation), as well as various quality controls previously acquired over seven independent experiments. An overview of the metadata used is given in Table 1.

Table 1. Overview of metadata files used throughout this pipeline, including each file's description, dimensionality (if appropriate), and purpose for preprocessing or quality control.

Description	Purpose
normalization_beads.fcs	
Beads identified using `CATALYST` during the normalization step of a previous CyTOF experiment.	Used as reference beads to correct for changes in signal sensitivity over time.
ref_bead_counts.csv	
A table of mean dual counts for the 6 different bead channels (columns) obtained from 7 previous CyTOF experiments (rows).	Used as a reference to assess the measurement sensitivity.
debarcoding_scheme.csv	
A binary barcoding scheme of 6-*choose*-3 = 20 barcodes with columns corresponding to barcode channel masses (101, 104, 105, 106, 108, 110) and rows corresponding to barcodes (7 empty, 9 references, 2 PBMC and 2 tumor samples)	Used for single-cell deconvolution of multiplexed samples.
spillover_matrix.csv	
A spillover matrix calculated with `CATALYST` from beads single-stained with each of the 40 antibodies included in the panel used in this study. The matrix contains, for each measurement channel (rows), the percentage of signal received by all other channels (columns).	Used for correction of spillover.
ref_cell_counts.csv	
A table of the number of cells measured in 7 previous experiments, each including 4 cell line, 3 PBMC and 2 tumor references samples (63 samples in total).	Used to assess reference sample cell yields in the current in comparison to previous experiments
sample_cell_counts.csv	
A table of the number of cells measured in 7 previous experiments, each including 2 PBMC and 2 tumor samples (28 samples in total).	Used to assess sample cell yields in the current in comparison to previous experiments
ref_marker_levels.csv	
A table of the 98th expression percentiles for each target (columns) across 7 previous experiments (rows).	Used to assess the staining efficiency of the current experiment

Data organization

Most data used and returned throughout this workflow are kept in an object of Bioconductor's `SingleCellExperiment` (SCE) class from the `SingleCellExperiment` package[18]. This data structure can store all single-cell related data (measurement data and transformations thereof; cell, feature and experiment-wide metadata; dimensionality reductions), allowing for synchronized and thus less error-prone data manipulation.

The key component of SCEs are matrix-like `assays`, where rows are features (targets) and columns are observations (cells), that store the measurement data and any data derived thereof. Metadata associated with cells are stored under `colData`, feature metadata under `rowData`, and any experiment-wide metadata may be stored in the `metadata` slot. Lastly, the SCE can store an arbitrary number of dimensionality reductions under `reducedDims`. For a more detailed description of usage and structure of SCEs, we refer to the *SingleCellExperiment* package's documentation.

Results

The pipeline presented here describes all steps required to process raw mass cytometry data to a state where the user may proceed with downstream analyses (e.g., dimensionality reduction, differential analysis, trajectory inference). The process includes the concatenation of the individual acquisitions, the exclusion of part of the acquisition with unstable signal, the correction for time-dependent signal drift via bead normalization, the correction for signal spillover via compensation, the selection of cells of interest via automated gating, and the correction for batch effects. The workflow is exemplified on data from a single CyTOF experiment collected via three successive acquisitions (individual FCS files) of 15 barcoded samples mixed with calibration beads.

Throughout, raw measurement data (FCS files) as well as all metadata (for debarcoding, normalization, compensation, and quality control) are expected to be located inside a `data/` subdirectory (relative to where the code is being run); otherwise, the presented file paths require modification.

We use `CATALYST`[9] to perform key preprocessing steps, including: concatenation, normalization, debarcoding and compensation; *openCyto*[19] and *flowWorkspace*[20] for gating; *ggplot2*[21], *ggcyto*[22] and *patchwork* for visualization; *flowCore*[23], `r CRANpkg("reshape2")´[24] and *dplyr*[25] for data accession and manipulation; and *mvtnorm* to compute polygonal live gates. Thus, our workflow is limited to the following dependencies:

```
library(CATALYST)
library(dplyr)
library(flowCore)
library(flowWorkspace)
library(ggcyto)
library(ggplot2)
library(mvtnorm)
library(openCyto)
library(patchwork)
library(reshape2)
```

Besides standard preprocessing steps, we include quality control (QC) steps to assess CyTOF sensitivity, staining efficacy, and cell yield; these rely on results from previous experiments (n = 7) as a reference. For consistent visualization, we define a common plotting theme for boxplots that are used to compare the current to previous experiments:

```
qc_theme <- list(
  theme_bw(base_size = 8), theme(
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    plot.title = element_text(face = "bold"),
    axis.text = element_text(color = "black"),
    axis.text.x = element_text(angle = 45, hjust = 1)))
```

Constructing a `SingleCellExperiment`

By default, *flowCore*'s `read.FCS()` function, which underlies `read.flowSet()` for reading in a set of FCS files, transforms channel intensities and removes events with extreme values. To omit this behavior, we recommend

reading in files with arguments `transformation = FALSE` and `truncate_max_range = FALSE`; by default, files will be read in by `CATALYST`'s `prepData()` function with these settings.

As described above, the SCE class allows the keeping of multiple data transformations in a single object. Thus, when applying a transformation to arrive at expression-like data, we can store the transformed data in a separate assay without overwriting the raw ion count data. In this way, any data generated and used throughout preprocessing (e.g., normalized, compensated or batch-corrected counts and their arcsinh-transformed counterparts) can be in principal retained, and written to intermediate FCS files for backup or quality control outside of R. However, it is worth noting that this procedure could lead to a shortage of memory for large datasets, in which case overwriting the data at each step is advisable; if not specified otherwise, `CATALYST` overwrites by default.

A SCE can be constructed using `CATALYST`'s `prepData()` function, which accepts a path to a directory with one or many FCS files, a character vector of FCS filenames, a single or list of `flowFrame`(s), or a `flowSet` (*flowCore* package[23]). By default (`transform = TRUE`), an arcsinh-transformation with `cofactor = 5` is applied to the input (count) data, and the resulting expression matrix is stored in the `exprs` assay slot of the output SCE:

```
# construct 'SingleCellExperiment'
fcs <- list.files("data", "acquisition", full.names = TRUE)
(sce <- prepData(fcs, transform = TRUE, cofactor = 5))
```

```
## class: SingleCellExperiment
## dim: 63 368152
## metadata(2): experiment_info chs_by_fcs
## assays(2): counts exprs
## rownames(63): 75As CD15 ... 208Pb CD45
## rowData names(4): channel_name marker_name marker_class use_channel
## colnames: NULL
## colData names(1): sample_id
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

Initially, our SCE has two assays containing dual ion counts (assay `counts`) and cofactor-5 arcsinhtransformed counts (assay `exprs`). The cofactor used for transformation is stored inside the object's internal metadata (`int_metadata(sce)$cofactor`), and the FCS file of origin for each cell under cell metadata column `sample_id` (accessible via `colData(sce)$sample_id` or, equivalently, `sce$sample_id`). In our dataset, FCS files correspond to acquisitions rather than biological samples. Thus, we rename the cell metadata variable `sample_id` to `file_id` to avoid ambiguity:

```
i  <- match("sample_id", names(colData(sce)))
names(colData(sce))[i]  <- "file_id"
```

The total number of cells across all acquisitions corresponds to the number of columns in the SCE (`ncol(sce)`: 368152). We can summarize the number of cells in each file by tabulating the `file_ids`:

```
data.frame(
    file_id = levels(sce$file_id),
    n_cells = tabulate(sce$file_id))
```

```
##   file_id n_cells
## 1      V1   48675
## 2      V2  125607
## 3      V3  193870
```

In both mass and flow cytometry, each feature has both a channel and target associated with it. As can be seen from printing the `sce` variable above, `prepData()` defaults to using targets as rownames (when available). We can retrieve each feature's measurement channel using the `channels()` accessor, and use channel metals and masses to extract the indices of features that are relevant to different preprocessing steps. Namely, we assign channels measuring DNA to the variable `dna` (here, Ir191 and Ir193), and channels for live gating (here, Ir191 for DNA and Pt194 for cisplatin) to `live`:

```
# store character vector or channels names
chs <- channels(sce)
# store DNA & live channel indices
dna <- grep("^Ir", chs)
live <- grep("191|194", chs)
```

Filtering for stable signal

High quality data generation requires a stable signal throughout the acquisition. Various issues can lead to signal change over time, including unstable flow rate, introduction of air or introduction of metal contamination in the system. These changes in signal intensity can vary in terms of duration and intensity, and can affect all or only a subsets of channels simultaneously. In order to detect regions of the acquisition affected by signal instability, we display the signal for selected channels as a function of time in a scatter plot (Figure 1).

```
# plot channels of interest vs. time
coi <- chs[c(dna[1], which(rowData(sce)$use_channel))]
plotScatter(sce, chs = c("Time", coi), label = "both") +
    labs(y = "expression") +
    scale_x_continuous(
        expression("Time ("*10^6~"ms)"),
        labels = function(u) u/1e6) +
    theme_bw(base_size = 8) + theme(
        aspect.ratio = 2/3,
        panel.grid = element_blank(),
        axis.text = element_text(color = "black"),
        strip.background = element_rect(fill = NA))
```

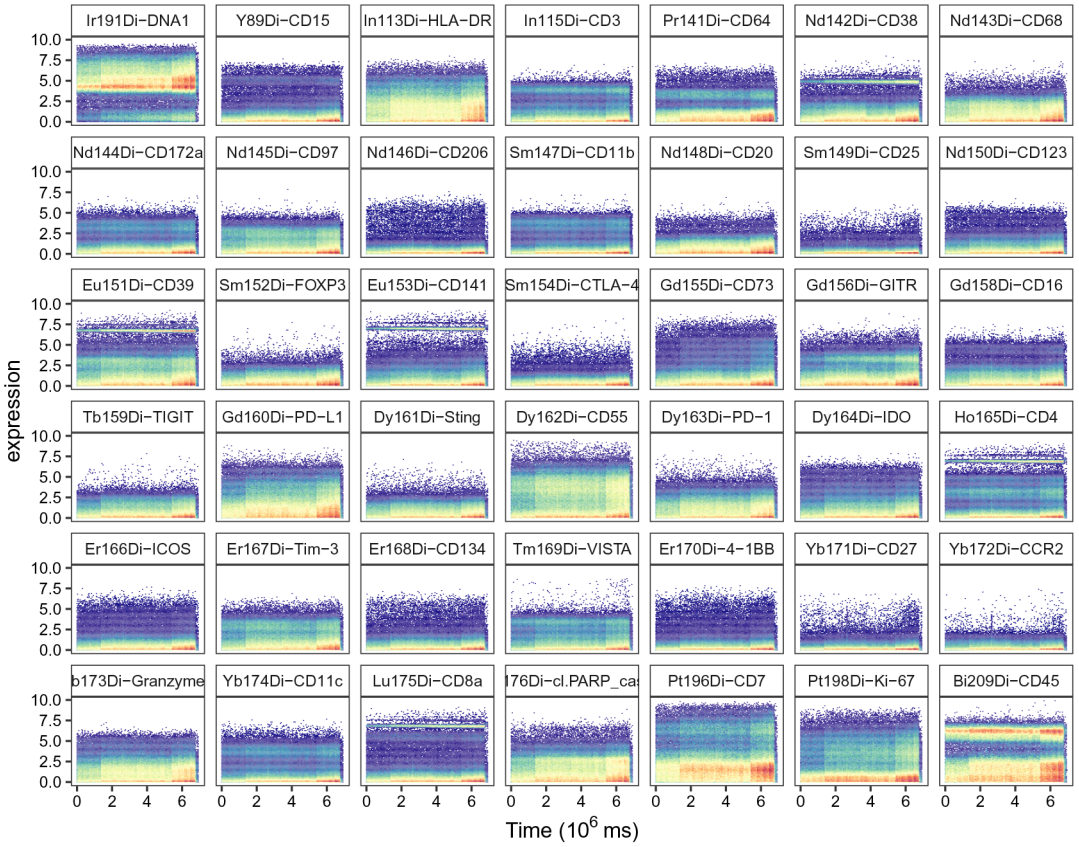

Figure 1. Scatter plots of DNA channel Ir191 and the 41 channels measuring antigens against time. Bins are colored by cell density; y-axis corresponds to cofactor-5 arcsinh-transformed dual counts.

In this particular experiment, we do not observe time-related signal instability. In case part of the acquisition should be excluded, this could be done by manually gating on the region with stable signal, and subsequent subsetting to only retain cells that fall within the gate's boundaries (argument `pop = "+"`). Vice versa, it is possible to select a region with unstable signal, and remove it from the SCE object (`pop = "-"`). For the sake of completeness, we include how a region of unstable signal could be excluded via manual gating:

```
# construct 'GatingSet'
ff <- sce2fcs(sce[dna, ], assay = "exprs")
gs <- GatingSet(flowSet(ff))

# apply rectangular gate to exclude unstable signal
min_t <- ...
max_t <- ...
gs_add_gating_method(
    gs, alias = "stable",
    pop = "-", parent = "root",
    dims = paste0("Time,", chs[dna[1]]),
    gating_method = "boundary",
    gating_args = sprintf("min=c(%s,0),max=c(%s,10)", min_t, max_t))

# plot scatter of DNA vs. Time
ggcyto(gs,
    aes_string("Time", chs[dna[1]])) +
    geom_hex(bins = 128) +
    geom_gate("stable") +
    facet_null() + theme_bw() +
    ggtitle(NULL) + theme(
        legend.position = "none",
        panel.grid = element_blank())

# subset selected events
sce <- sce[, gh_pop_get_indices(gs, "stable")]
```

Normalization

In the case of mass cytometry, signal drift during acquisition due to a progressive loss of sensitivity must be accounted and normalized for. A widely established strategy is to mix samples with polystyrene beads embedded with metal lanthanides, allowing monitoring of instrument performance throughout data acquisition[8]. These beads are in turn used to estimate and correct for the signal's time drift. When independent experiments have to be analyzed in the same context, variation due to changes in instrument performance over time combined with intervals between scheduled maintenance have to be taken into account as well. In this case, the bead signal should be normalized to a set of reference beads from an earlier experiment. This ensures that different experiments are normalized to the same level, independent of the CyTOF's sensitivity.

A MATLAB tool to perform normalization outside of R was available until recently at nolanlab/beadnormalization; current R implementations are available through *CATALYST* and *premessa*. `CATALYST` provides an extension of bead-based normalization as described by Finck *et al.* [8], with automated identification of bead singlets (used for normalization), as well as of bead-bead and cell-bead doublets (to be removed), thus eliminating the need for manual gating. This is implemented as follows:

1. beads are initially identified as those events that have their highest signals in the bead channels
2. cell-bead doublets are removed by applying a separation cutoff on the distance between the lowest bead and highest non-bead channel signal
3. events passing all vertical gates defined by the lower bounds of bead signals are removed (these include bead-bead and bead-cell doublets)
4. bead-bead doublets are removed by applying a default *median* ± 5 *mad* rule to events identified in step 2; the remaining bead events are used for normalization

The above procedure is carried out by a single function, `normCytof()`, which takes as input a SCE and a set of arguments that control the normalization parameters and output format. Here, we specify `dna = 191` (Ir191) and `beads = "dvs"`, corresponding to DVS Science beads (lanthanides Ce140, Eu151, Eu153, Ho165, Lu175). Secondly, we provide the path to a set of reference beads (argument `norm_to`) that are used to compute baseline intensities for normalization. Lastly, we set `overwrite = FALSE` to retain both raw and normalized data, and `remove_beads = TRUE` to exclude bead and doublet events:

```
# specify path to reference beads
ref_beads <- file.path("data", "normalization_beads.fcs")
# apply bead-based normalization
system.time(res <- normCytof(sce, beads = "dvs", dna = 191,
    norm_to = ref_beads, remove_beads = TRUE, overwrite = FALSE))
```

```
##    user  system elapsed
##  20.134   1.343  21.963
```

When `remove_beads = TRUE` (the default), `normCytof()` will return a list of three SCEs containing filtered, bead and remove events, respectively, as well as two `ggplot` objects:

```
names(res)
```

```
## [1] "data"    "beads"   "removed" "scatter" "lines"
```

The first SCE (`res$data`) contains the filtered data with the additional assay slot `"normed"` housing normalized expressions. The remaining two SCEs are data subsets that contain any events identified as beads (slot `beads`) and all removed events (including beads, bead-bead and bead-cell doublets; slot `removed`), respectively; thus, the `beads` themselves are a subset of the `removed` events. Here, we compare the number and percentage of cells contained in each subset:

```
# view no. of remaining, bead & removed events
ns <- sapply(res[1:3], ncol)
ps <- sprintf("%1.2f", ns/ncol(sce)*100)
data.frame(t(cbind("# events" = ns, "% of total" = ps)))
```

```
##              data beads removed
## # events   337525 27544   30627
## % of total  91.68  7.48    8.32
```

As a first quality control plot, `res$scatter` (Figure 2) renders scatter plots of bead channels (x-axis) versus DNA (y-axis), where events identified as beads as well as their expression range are highlighted in color; bead events should have low DNA intensity (since they are not cells) and high intensities across all bead channels.

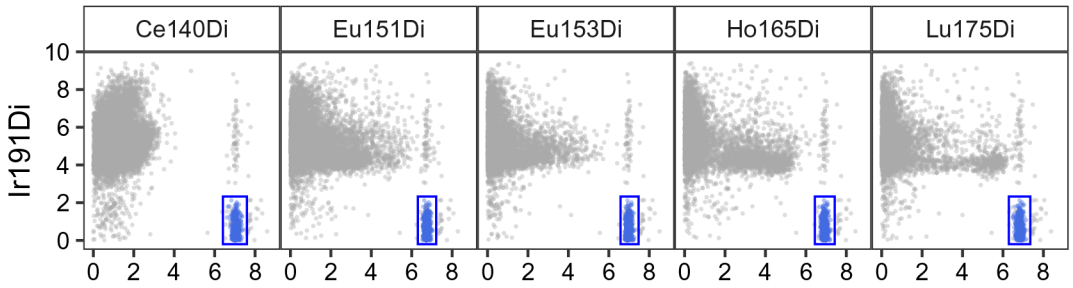

Figure 2. Scatter plots for bead channels vs. DNA. Events identified as beads are colored in blue; for each bead channel, expression ranges across all bead events are indicated as rectangular gates. Events were downsampled to at most 10,000 for visualization.

Secondly, `res$lines` (Figure 3) displays smoothed median bead intensities before and after normalization; these typically decrease with time prior to normalization, and should be approximately constant and centered around the baseline after normalization. In our dataset, normalization is performed based on previously acquired reference beads. Thus, baseline values correspond to the reference bead's mean bead channel intensities. As shown in Figure 3, the bead channel levels are considerably lower after normalization, indicating higher sensitivity in the current experiment. Importantly, the slight decrease in signal over time is no longer present after normalization.

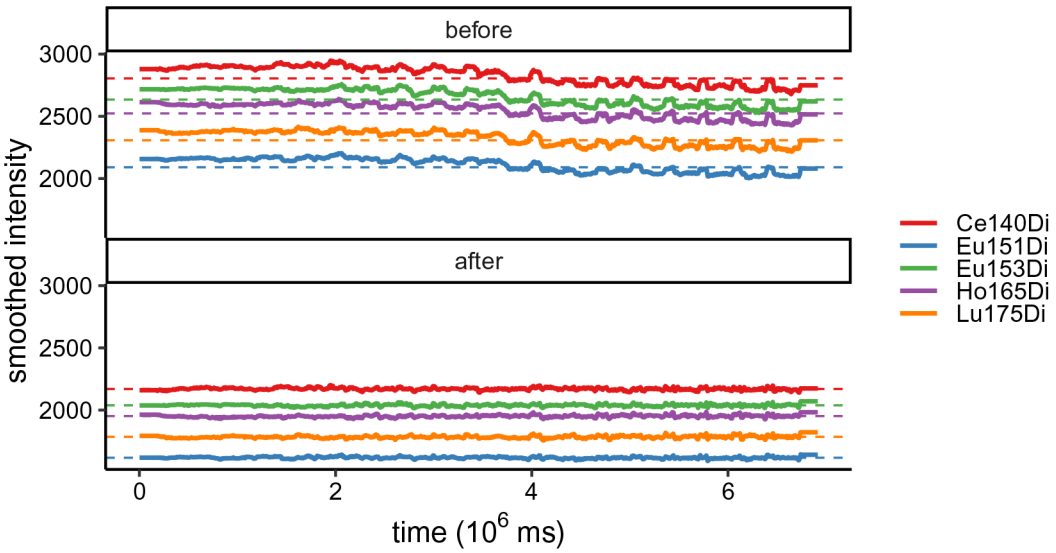

Figure 3. Running-median smoothed bead intensities vs. time before and after normalization; colored by bead channel.

In order to assess the sensitivity of the CyTOF during acquisition and identify potential issues that would have remained undetected during the tuning of the instrument, we compute the mean bead channel counts across events identified as beads (`res$beads` subset). A logical vector of which channels correspond to beads is stored under `rowData` column `bead_ch`, which we can use to subset the `counts` assay to include bead channels only.

```
# compute mean bead channel counts for current experiment
is_bead <- rowData(res$beads)$bead_ch   # get bead channels
bead_cs <- counts(res$beads)[is_bead, ] # subset counts
rownames(bead_cs) <- chs[is_bead]       # use channels as names
(bead_ms <- rowMeans(bead_cs))          # compute means
```

```
##  Ce140Di  Eu151Di  Eu153Di  Ho165Di  Lu175Di
## 2842.462 2111.367 2660.618 2538.095 2323.409
```

To assess the measurement sensitivity during the current experiment, we compare the mean bead channel counts computed above to those obtained from 7 previously acquired experiments available in metadata table `ref_bead_counts.csv`. The resulting boxplot (Figure 4) shows that the current experiment's sensitivity is relatively high, but well in the range of previous experiments.

```
# read in reference mean bead channel counts
ref <- read.csv(file.path("data", "ref_bead_counts.csv"))

# join into single tidy data.frame
df <- bind_rows(ref, bead_ms, .id = "group")
df <- melt(df, id.var = "group")

# boxplot of reference vs. current experiment's mean bead channel counts
ggplot(df, aes(variable, value)) +
    geom_boxplot(data = df[df$group == 1, ]) +
    geom_point(data = df[df$group == 2, ],
        col = "red", pch = 4, stroke = 1) +
    labs(x = "bead channel", y = "mean count") +
    qc_theme + ggtitle(
        "QC on bead channel counts",
        "[-] = reference | x = current experiment")
```

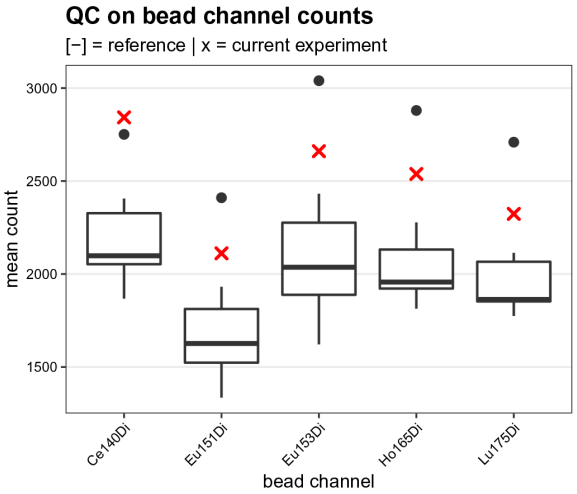

Figure 4. Bead channel count quality control. Boxplot comparing the mean dual ion counts (y-axis) across bead channels (x-axis) obtained for the current experiment (red crosses) to those from 7 previously acquired reference experiments (boxes).

After normalization, we overwrite the input dataset with the filtered subset that no longer includes bead events, or bead-bead and bead-cell doublets:

```
sce <- res$data
```

Debarcoding

In mass cytometry, samples are often labeled with unique sample-specific barcodes and pooled together for processing and measurement, an approach termed *multiplexing*[26]. The most widely used barcoding scheme is based on Zunder *et al.*[10], and relies on binary palladium-based mass-tag cell barcoding. Here, each sample $i = 1,..., n$ is either positive or negative for each of m palladium isotopes, resulting in an m-choose-k barcoding scheme, where k is the number of positive barcodes. For example, labeling of three out of six palladium isotopes will result in $\binom{m}{k} = \binom{6}{3} = 20$ unique barcodes. In order to recover the individual samples for further analysis, the pooled dataset is debarcoded (or *deconvoluted*) computationally.

The single cell debarcoding (SCD) algorithm first sorts each cell's barcode intensities to assign preliminary barcode IDs such that a cell is assigned to the barcode population for which its barcode intensities are highest. Next, intensities within each barcode population are scaled using the 95th expression quantiles, and thereby

brought to a comparable scale. Finally, events whose separation between highest negative and lowest positive barcode intensity is below a threshold value (*separation cutoff*) are left unassigned.

In the initial SCD algorithm, sample yields are determined by a single global cutoff on the separation between positive and negative barcode populations. Naturally, this procedure is suboptimal when yields as a function of the applied cutoff do not decline simultaneously. To optimize cell yields in such cases, *CATALYST* provides an option to automatically estimate or specify *sample-specific* separation cutoffs.

The SCD algorithm is implemented in `CATALYST` as a three-step procedure: i) preliminary barcode assignment (`assignPrelim()`); ii) automated estimation of sample-specific separation cutoffs (`estCutoffs()`); and, iii) application of cutoffs to arrive at final barcode assignments (`applyCutoffs()`).

Preliminary barcode assigment

For our dataset, a 6-choose-3 = 20 barcoding scheme was used (Figure 5). Five barcodes were unused (`empty_1-5`), resulting in 15 samples (9 references, 6 samples of interest). We first read the corresponding `debarcoding_scheme.csv` into R:

```
# read in debarcoding scheme
fn <- file.path("data", "debarcoding_scheme.csv")
bc_key <- read.csv(fn, row.names = 1, check.names = FALSE)

# all barcodes are positive for exactly 3 barcoding channels
all(rowSums(bc_key) == 3)
```

```
## [1] TRUE
```

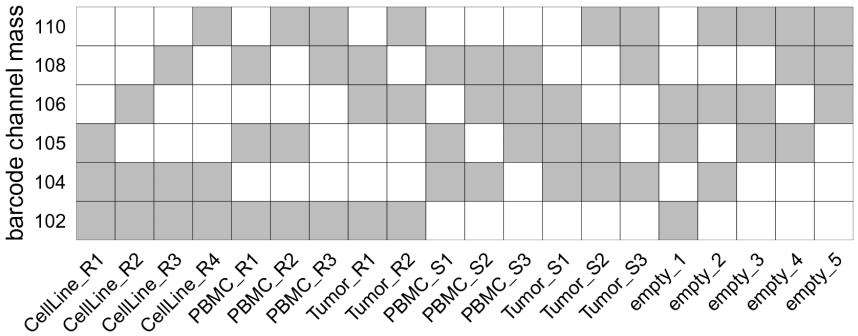

Figure 5. 6-choose-3 palladium isotope debarcoding scheme. Rows correspond to palladium isotopes (barcode channels), columns to barcode identifiers (samples). Each sample is negative (white) or positive (grey) for 3 out of 6 barcode channels, resulting in 20 unique barcode combinations.

During this first debarcoding step, each event is preliminarily assigned to a barcode according to its top-*k* expressed barcode channels. Here, events whose expression is highest for a combination of barcode channels that does *not* appear in the debarcoding scheme (`bc_key`) will be given barcode ID 0 (for "unassigned"). Thus, we can remove empty barcodes from the `bc_key` variable such that events assigned to these barcodes are left unassigned from the start. Alternatively, one could perform debarcoding using the non-subsetted key, and filter out empty barcodes downstream.

```
# remove empty barcodes from debarcoding scheme
is_empty <- grepl("empty", rownames(bc_key))
bc_key <- bc_key[!is_empty, ]
bc_ids <- rownames(bc_key)
```

For preliminary barcode assignment, we use `CATLAYST's assignPrelim()` function, providing the input data (`sce`) and debarcoding scheme (`bc_key`). If not specified otherwise, `assignPrelim()` will default to using the `exprs` assay slot (argument `assay`). Because we ran `normCytof()` with `overwrite = FALSE`,

this assay contains arcsinh-transformed *raw* counts; we set `assay = "normexprs"` in order to use the normalized values instead:

```
# do preliminary barcode assignments
system.time(sce <- assignPrelim(sce, bc_key, assay = "normexprs"))
```

```
##    user  system elapsed
##  14.290 0.347 14.770
```

In the returned SCE, feature metadata (`rowData`) column `is_bc` indicates whether or not a channel corresponds to a barcode channel:

```
# view barcode channels
channels(sce)[rowData(sce)$is_bc]
```

```
##      MCB1      MCB2      MCB3      MCB4      MCB5      MCB6
## "Pd102Di" "Pd104Di" "Pd105Di" "Pd106Di" "Pd108Di" "Pd110Di"
```

For each event, barcode identifiers are stored in `colData` column `bc_id`. After this preliminary round of assignment, 57980/337525 events (17.18%) have been left unassigned:

```
# tabulate number of (un)assigned events
table(sce$bc_id == 0)
```

```
##
##  FALSE   TRUE
## 279545  57980
```

Furthermore, for each cell, the barcode channel expressions are scaled relative to the 95th expression percentiles of its respective barcode population. The scaled data is stored in assay slot `scaled`. Based on these scaled barcode channel intensities, a separation value is computed as the distance between highest negative and lowest positive barcode channel; separations are stored in `colData` column `delta`.

Estimation of separation cutoffs

To decide on separation cutoffs, we consider yields upon debarcoding as a function of the applied cutoff (Figure 6). Commonly, this function will be characterized by an initial weak decline, where doublets are excluded, and subsequent rapid decline in yields to zero. In-between, low numbers of counts with intermediate barcode separation give rise to a plateau. Ideally, the applied separation cutoffs should provide a balance between high cell yield and low assignment uncertainty, marking the approximate midpoint of the yield function's plateau region.

```
plotYields(sce, which = "0")
```

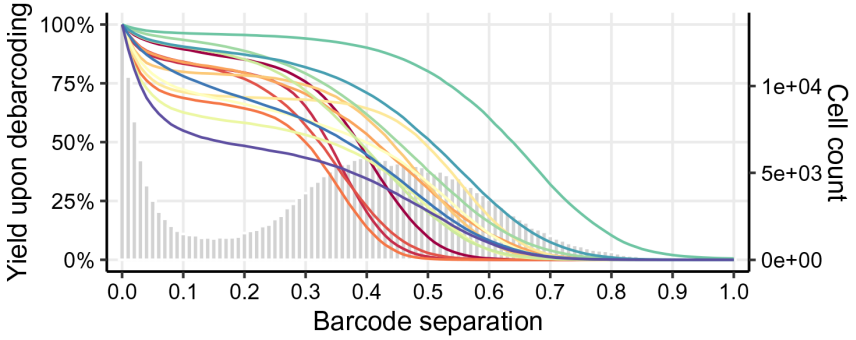

Figure 6. Yield plot for a 6-choose-3 debarcoding scheme. Shown is the distribution of barcode population separations (histogram) and cell yields by sample (lines) as a function of the applied separation cutoff. Left axis corresponds to cell yield in percent; right axis to the total number of cells.

Instead of a single global cutoff, we estimate a sample-specific cutoff to account for barcode population yields that decline in an asynchronous fashion. To this end, we fit both a linear and a three-parameter log-logistic model to each yield function. For the linear fit, we estimate the cutoff as the value for which yields have declined to 50%. For the log-logistic fit, we compute the cutoff as the value for which there is minimal yield decline by minimizing each yield function's 1st derivative. For each barcode, the final cutoff estimate is computed as the mean of both estimates, weighted with the goodness (residual sum of squares) of each fit (see Methods for details). Thus, the choice of thresholds for the distance between negative and positive barcode populations is: i) automated and ii) independent for each barcode. Nevertheless, reviewing barcode-specific yield plots and, in rare cases, refining the estimated separation cutoffs is advisable (see Figure 7).

Cutoff estimation is performed by `CATALYST`'s `estCutoffs()` function, which takes as input a SCE as returned by `assignPrelim()`; that is, preliminary barcode assignments are required to estimate separation cutoffs. `estCutoffs()` will store sample-specific cutoff estimates under `metadata` slot `sep_cutoffs`, but will leave barcode assignments unchanged.

```
sce <- estCutoffs(sce)
metadata(sce)$sep_cutoffs
```

```
## CellLine_R1 CellLine_R2 CellLine_R3 CellLine_R4      PBMC_R1      PBMC_R2
##  0.13829607  0.13688845  0.09161274  0.12437132  0.13039323  0.18047875
##      PBMC_R3    Tumor_R1    Tumor_R2     PBMC_S1     PBMC_S2     PBMC_S3
##  0.26517442  0.21014175  0.20543502  0.10439323  0.12902725  0.24858493
##    Tumor_S1    Tumor_S2    Tumor_S3
##  0.18442675  0.14690041  0.20818048
```

We can visually inspect the estimated cutoffs using `plotYields()` with argument `which` specifying the barcode ID of interest (Figure 7). In our example, the cutoff estimate nicely marks the midpoint of the yield function's plateau or, equivalently, the valley between peaks of cell yields. To decrease the stringency of the applied cutoff, and thus increase the resulting cell yield, we could set the sample's cutoff to e.g. 0.1. Vice versa, a more stringent cutoff of e.g. 0.2 would decrease the cell yield but yield a purer population.

As an alternative to inspecting the cutoff estimate for each sample in R, we could specify `which = bc_ids` to obtain a list of yield plots for all barcodes; the generated set of plots may be written to a single PDF file via providing `plotYields()` with an `out_path` to allow for easy reviewing of the separating cutoffs currently stored within the object.

```
plotYields(sce, which = "PBMC_R1")
```

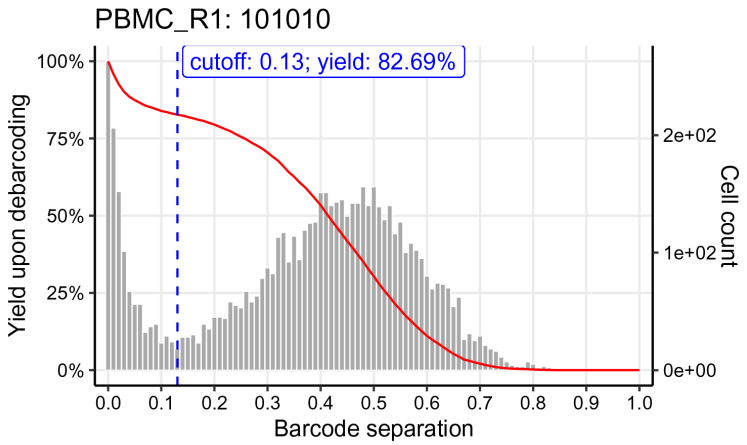

Figure 7. Yield plot for an exemplary sample, including the estimated separation cutoff. Shown is the distribution of barcode population separations (histogram) and cell yields (line) for the sample as a function of the applied sample-specific separation cutoff. Left axis corresponds to cell yield in percent; right axis to the total number of cells.

Besides a cutoff on the separation between positive and negative barcode populations, to trim outliers, the SCD algorithms applies an additional cutoff on the *Mahalanobis distance* (argument `mhl_cutoff`), a metric that quantifies the distance of a given event to the expression distribution of the barcode population it has been assigned to.

In Figure 6, we can observe that population yields decline synchronously with increasing separation cutoffs, and that we might consider applying a global separation cutoff, e.g., at ~ 0.15. For this data, yields are in fact similar, independent of whether we apply sample-specific cutoffs or a single global one. Nevertheless, applying sample-specific cutoffs is recommended in order to maximize cell yields while minimizing uncertainty in barcode assignments.

```
# store preliminary barcode IDs
bc_ids0 <- sce$bc_id

# apply global & sample-specific separation cutoff(s)
sce_glob <- applyCutoffs(sce, sep_cutoffs = 0.15, mhl_cutoff = 30)
sce_spec <- applyCutoffs(sce, mhl_cutoff = 30)

# compare cell yields for both cutoff strategies
c(global = mean(sce_glob$bc_id == 0),
  specific = mean(sce_spec$bc_id == 0))
```

```
##    global  specific
## 0.3573839 0.3584979
```

After debarcoding, we compare the number of events assigned to each barcode population before and after applying separation cutoffs, and filter out events that have been left unassigned (barcode ID 0). As shown in Figure 8, after applying the separation cutoffs, the number of unassigned cells (0) increases, while the number of cells assigned to each barcoding well decreases. We also observe a higher decrease in assigned cells for tumor samples, which underwent a dissociation process and contain more debris. Conversely, highly viable cell lines and PBMCs have a higher recovery yield.

```
# proceed with sample-specific filtering
sce <- sce_spec

# compute number of events per population
# before vs. after applying separation cutoffs
barplot(rbind(table(bc_ids0), table(sce$bc_id)),
    beside = TRUE, ylab = "cell count",
    las = 2, cex.axis = 0.5, cex.names = 0.5)
legend("topright", fill = c("black", "grey"),
    legend = c("before filtering", "after filtering"))

# remove unassigned events
sce <- sce[, sce$bc_id != 0]
```

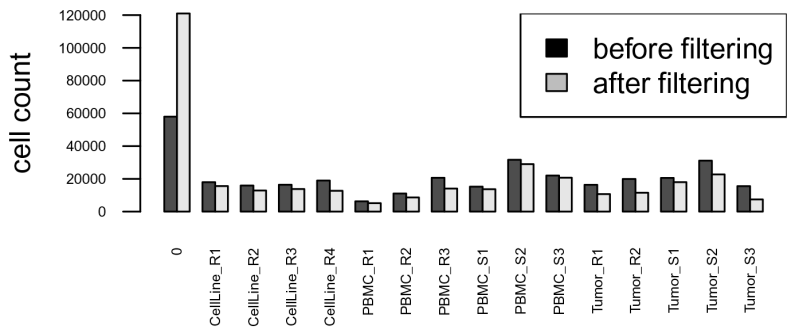

Figure 8. Barplot of cell counts before (black) and after (grey) applying separation cutoffs.

Compensation

Mass cytometry utilizes heavy metals (usually from the lanthanide series) as reporters to label antibodies. As a result, channel crosstalk originating from spectral overlap and autofluorescence is significantly less pronounced in mass cytometry compared to flow cytometry. Yet, spillover due to abundance sensitivity, isotopic impurities, and oxide formation still exists, giving rise to artefactual signal that can impede data interpretability.

A combined experimental-computational pipeline to correct for spillover in mass cytometry data has been proposed by Chevrier *et al.*[9] and is implemented in the *CATALYST* package. In brief, compensation is achieved via the following three-step approach outlined here (see for details).

1. Identification of single positive populations via deconvolution of single-stained beads (`assignPrelim()`, `estCutoffs()`, `applyCutoffs()`).
2. Estimation of a spillover matrix (SM) from the populations identified (`computeSpillmat()`).
3. Compensation via multiplication of measurement intensities by the SM's inverse, the compensation matrix (`compCytof()`).

We will apply a pre-acquired spillover matrix (metadata file `spillover_matrix.csv`). Thus, we enter at step 3, which involves only compensating the input dataset using `CATALYST's compCytof()` function. By default, `compCytof()` will reuse the cofactor stored in `int_metadata(sce)$cofactor` for computing arcsinh-transformed data from the compensated counts, thus applying the same transformation as during data preparation and normalization:

```
# read in pre-computed spillover matrix
sm <- file.path("data", "spillover_matrix.csv")
sm <- read.csv(sm, row.names = 1)
# apply NNLS compensation
system.time(
     sce <- compCytof(sce, sm, method = "nnls",
         assay = "normcounts", overwrite = FALSE))
```

```
##    user  system elapsed
##  63.538   5.880  70.095
```

To visually inspect how compensation affects signal intensities, we can generate scatter plots pre- and post-compensation; an exemplary pair of channels is shown in Figure 9. In such a plot, we can observe a slight positive association between the signals of spill-affected channels, which should be removed upon compensation.

```
i <- grep("173|174", chs, value = TRUE)
p1 <-  plotScatter(sce,
  chs = i,
  label = "channel",
  assay = "normexprs") +
  ggtitle("Uncompensated")
p2 <- plotScatter(sce,
  chs = i,
  label = "channel",
  assay = "compexprs") +
  ggtitle("Compensated") +
  ylab(NULL)
wrap_plots(p1, p2)
```

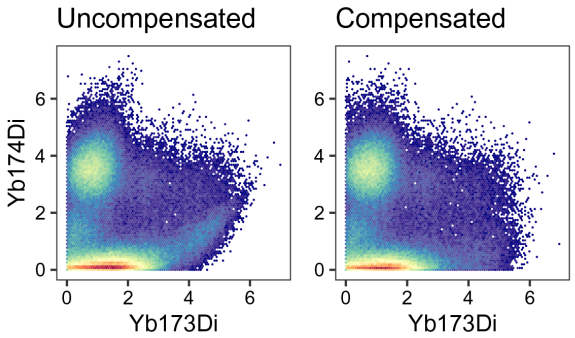

Figure 9. Scatter plots for two exemplary channels before (left) and after correction for spillover (right).

Gating

Many events acquired in mass cytometry may in fact be debris, doublets or dead cells, and should be filtered out through a gating step. Here, we suggest a strategy that first applies an elliptical gate on cell events, defined as double positive for the DNA channels Ir191/Ir193. This allows the exclusion of debris and doublets. As a second step, we discard cells that are positive for the dead cell marker Pt194.

These two steps are performed using the *openCyto* R package[19], and the resulting gates are visualized on scatter plots of the channels subjected to gating using *ggcyto*[22]. For consistent visualization, we again define a common plotting theme for scatter plots of channels `chs` that include the gating boundaries for the specified `gate_id`:

```
.scatter <- function(gs, chs, gate_id = NULL,
    subset = ifelse(is.null(gate_id), "root", "_parent_")) {
    p <- ggcyto(gs, max_nrow_to_plot = 1e5,
         aes_string(chs[1], chs[2]), subset) +
         geom_hex(bins = 100) + facet_wrap(~ name, ncol = 5) +
         (if (is.null(gate_id)) list() else geom_gate(gate_id)) +
         ggtitle(NULL) + theme_bw(base_size = 8) + theme(
             aspect.ratio = 1,
             legend.position = "none",
             panel.grid.minor = element_blank(),
             strip.background = element_rect(fill = NA),
             axis.text = element_text(color = "black"),
             axis.text.x = element_text(angle = 45, hjust = 1))
    suppressMessages(p + coord_equal(expand = FALSE,
        xlim = c(-1, 11), ylim = c(-1, 11)))
}
```

Gating on cells

In order to apply sample-specific gates, we first convert the SCE into a `flowSet` with a separate frame for each sample (argument `split_by = "bc_id"`). As gating should be performed on expression-like data (not ion counts), we further specify `assay = "exprs"` to retain the arcsinh-transformed assay slot. Thirdly, since conversion from SCE to `flowCore` data structures requires matrix transposition (rows correspond to targets in the SCE, but to events in `flowFrame/Sets`), we retain only those channels that are relevant when gating of (live) cells: DNA and dead channels, whose indices are stored in variables `dna` and `live`.

```
# subset DNA & live channels
sub <- sce[union(dna, live), ]

# add metadata variable 'i' to track cell indices
colData(sub) <- DataFrame(
    bc_id = sub$bc_id,
    i = seq_len(ncol(sce)))
```

```
# split SCE by sample
fs <- sce2fcs(sub,
    assay = "compexprs",
    split_by = "bc_id",
    keep_cd = TRUE)

# construct 'GatingSet'
gs <- GatingSet(fs)
```

We apply an elliptical gate (`gating_method = "flowClust.2d"`) to exclude the two lowest density percentiles (`quantile = 0.98`). Because the input gating set contains a separate frame for each barcode, the gate will be computed separately for each sample. In case of a single DNA channel (e.g., Rh103), one-dimensional gates (i.e., thresholds on minimum and maximum values) would be applicable instead.

```
# apply elliptical gate on DNA channels
gs_add_gating_method(gs,
    alias = "cells",
    pop = "+", parent = "root",
    dims = paste(chs[dna], collapse = ","),
    gating_method = "flowClust.2d",
    gating_args = "K=1,quantile=0.98,target=c(5,5)")
```

We use *ggcyto* to produce scatter plots of the DNA channels, with `geom_gate("cells")` to visualize the gates computed above (Figure 10):

```
# plot scatter of DNA channels split by sample
.scatter(gs, chs[dna], "cells")
```

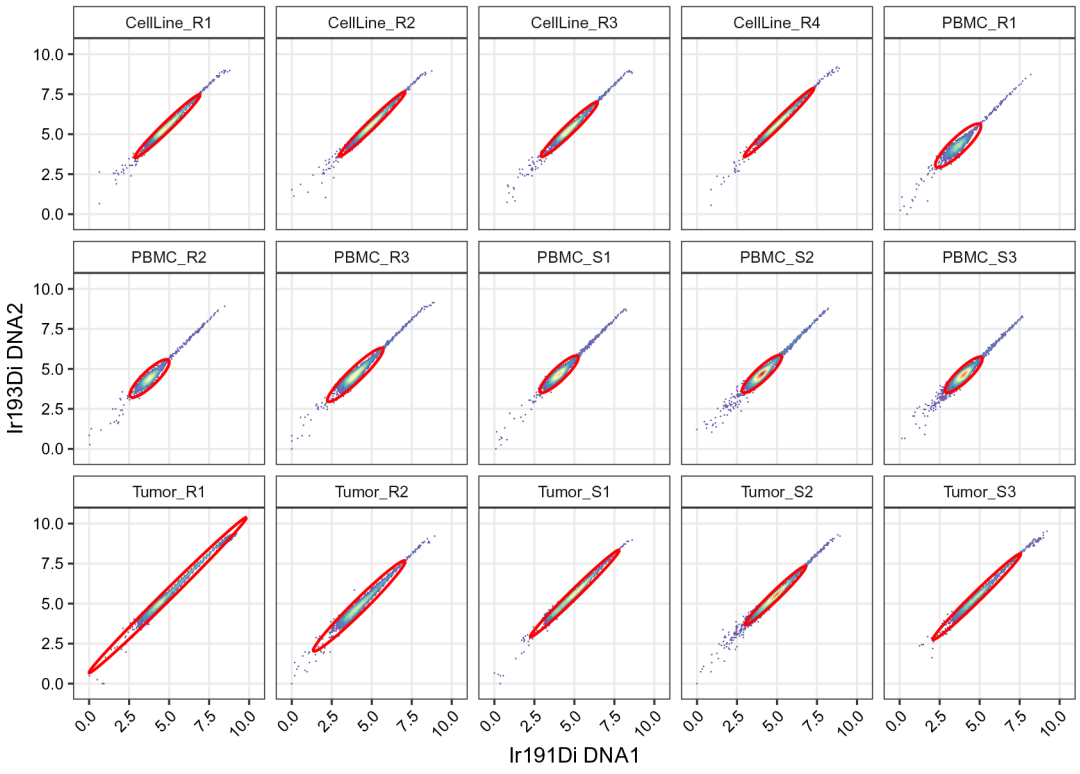

Figure 10. Scatter plots of DNA channels, split by sample and including elliptical cell gates.

Gating on live cells

The wrapper function `.live_gate()` defines a polygonal gate comprised of a line and a bivariate standard normal density Z, such that cells pass gating when i) their expression is within the `q`th quantile of Z; and, ii) their expression falls below a line parameterized by intercept `i` and slope `s`. In this way, the gate is centered around the expression peak, while excluding cells whose `dead` channel intensities increases with DNA content.

```
# define live cell gate plug-in
# x = expression matrix, q = quantile, i = intercept, s = slope
.live_gate <- function(x, q = 0.99, i = 1, s = 0.5) {
    # specifying gating function
    .gating_fun <- function(fr, pp_res, channels = NA, id = "", ...) {
        # subset channels of interest
        x <- exprs(fr[, channels])
        # scale data for comparison w/ 'qnorm()'
        x0 <- scale(x)
        # set boundary level as q-th quantile of standard normal
        z <- qnorm(q)
        # find p(x) for that level
        pd <- dmvnorm(c(z, z))[1]
        px <- dmvnorm(x0)
        # find points above boundary level
        keep1 <- px > pd
        # find points below line y = a + b * x
        keep2 <- (i + s * x0[, 1]) > x0[, 2]
        # intersection of points below line & above threshold level
        pts <- x[keep1 & keep2, ]
        # get boundary points (convex hull)
        pts <- pts[chull(pts), ]
        # return gate
        polygonGate(.gate = pts, filterId = id)
    }
    # register gate
    suppressMessages(
      foo <- register_plugins(
        fun = .gating_fun,
        methodName = "liveGate",
        dep = "mvtnorm",
        "gating"))
}
```

In contrast to the cell gates above, we apply live gates with sample-specific gating parameters. To this end, we specify a list `l` containing quantiles `q`, intercepts `i` and slopes `s` for each sample. These parameters are updated iteratively to remove dead cells while retaining cell yields as high as possible (Figure 11). After manual adjustments, we arrive at the following sample-specific gating parameters:

```
# set default parameters for all samples
l <- lapply(c(q = 0.99, i = 0.9, s = 0.4), function(u)
    setNames(rep(u, length(gs)), sampleNames(gs)))

# adjust parameters for specific samples
l$i[["PBMC_R2"]] <- 1.2
l$i[["PBMC_R3"]] <- 1.2
l$i[["PBMC_S1"]] <- 1.2
l$s[["PBMC_S2"]] <- 0.2
l$i[["PBMC_S2"]] <- 0.6
l$i[["PBMC_S3"]] <- 1.8
l$s[["Tumor_S2"]] <- 0.3
l$i[["Tumor_S2"]] <- 0.6
l$s[["Tumor_S3"]] <- 0.3
l$i[["Tumor_S3"]] <- 0.4
```

```
for (i in sampleNames(gs)) {
    # register & apply live gate with sample-specific parameters
    .live_gate(x, q = l$q[i], i = l$i[i], s = l$s[i])
    gs_add_gating_method(gs[i],
        alias = "live",
        pop = "+",
        parent = "cells",
        dims = paste(chs[live], collapse = ","),
        gating_method = "liveGate")
}
.scatter(gs, chs[live], "live")
```

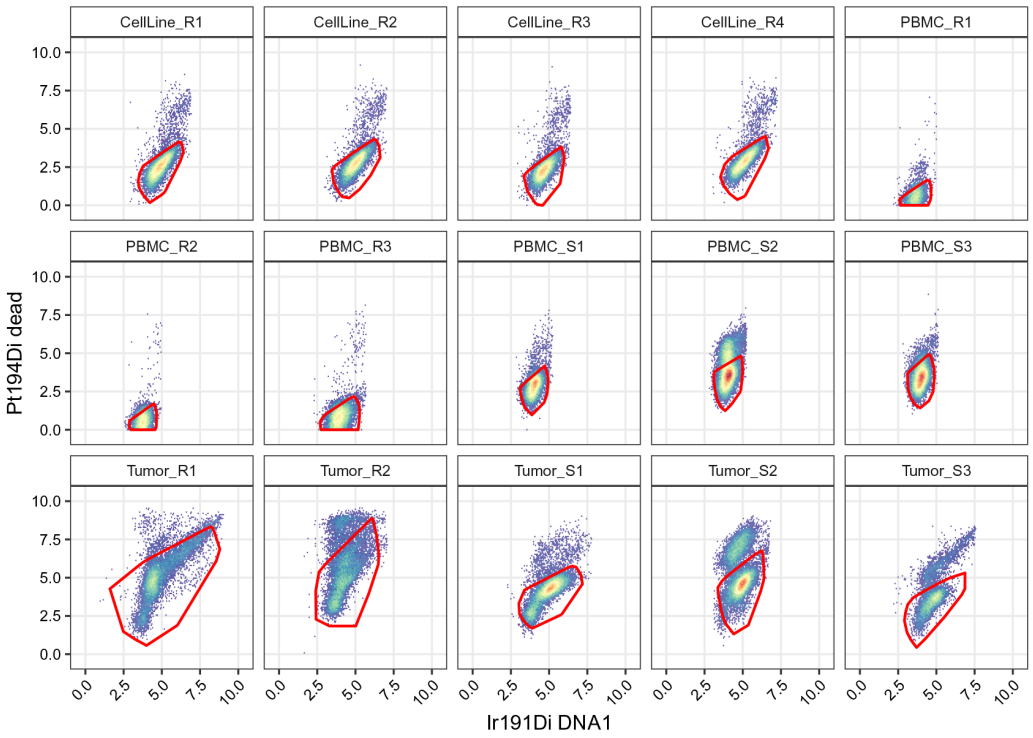

Figure 11. Scatter plots of DNA and dead cell channels, split by sample and including the live cell polygon gates.

We display the yield of `"cell"` and `"live"` gates on each samples to quickly assess the cell losses occurring at the two gating steps (Figure 12). As expected the `"cell"` gate leads to a systematic loss of around 1% of cells across all the samples. The `"live"` gate leads to a stronger reduction of cell yield in the tumor samples, consistent with the fact that those samples, which underwent enzymatic dissociation, contain more dead cells.

```
# extract gating frequencies
df <- gs_pop_get_stats(gs,
    type = "percent",
    nodes = c("cells", "live"))
df <- rename(df, gate_id = "pop")

# barplot of cell yields after cell/live gating
ggplot(df, aes(sample, percent, fill = gate_id)) +
    geom_bar(width = 2/3, stat = "identity", position = "dodge") +
    scale_x_discrete(limits = bc_ids, expand = c(0, 2/3)) +
    scale_y_continuous(labels = seq(0, 100, 25),
        limits = c(0, 1), expand = c(0, 0)) +
    labs(y = "cell yield (%)") + qc_theme
```

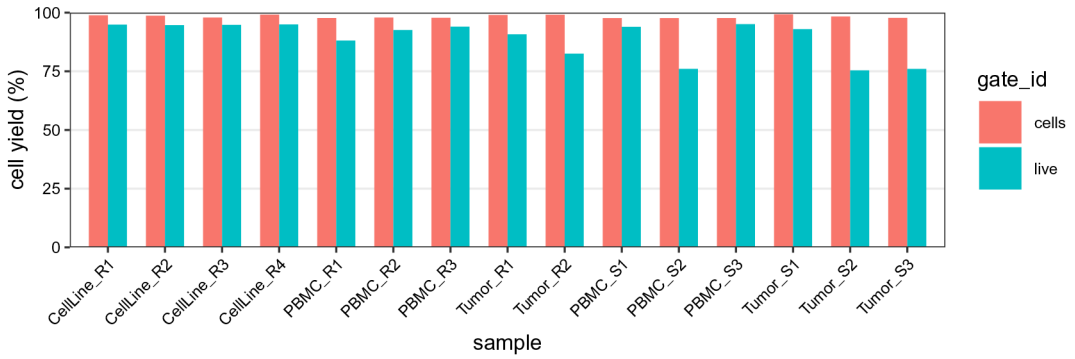

Figure 12. Barplot of cell and live gating yields. For each barcode ID (x-axis), frequencies are relative to the total number of cells in the population before gating; bars are colored by gate ID.

We extract a logical vector indicating whether a given event is included in or excluded by the `"live"` gate applied above by applying `gh_pop_get_indices` to each sample in `gs`. Secondly, we extract the cell indices from `gs` and subset the SCE to keep only cells that passed the `"live"` gate.

```
fs  <- gs_pop_get_data(gs, "live") # get data from 'GatingSet'
es  <- lapply(fs, exprs)           # get expression matrices
es  <- do.call("rbind", es)        # join into single data.frame
sce <- sce[, es[, "i"]]            # subset retained cells
```

Finally, we can again visualize scatter plots of dead channels against DNA as a quality control for the retained subset of cells (Figure 13).

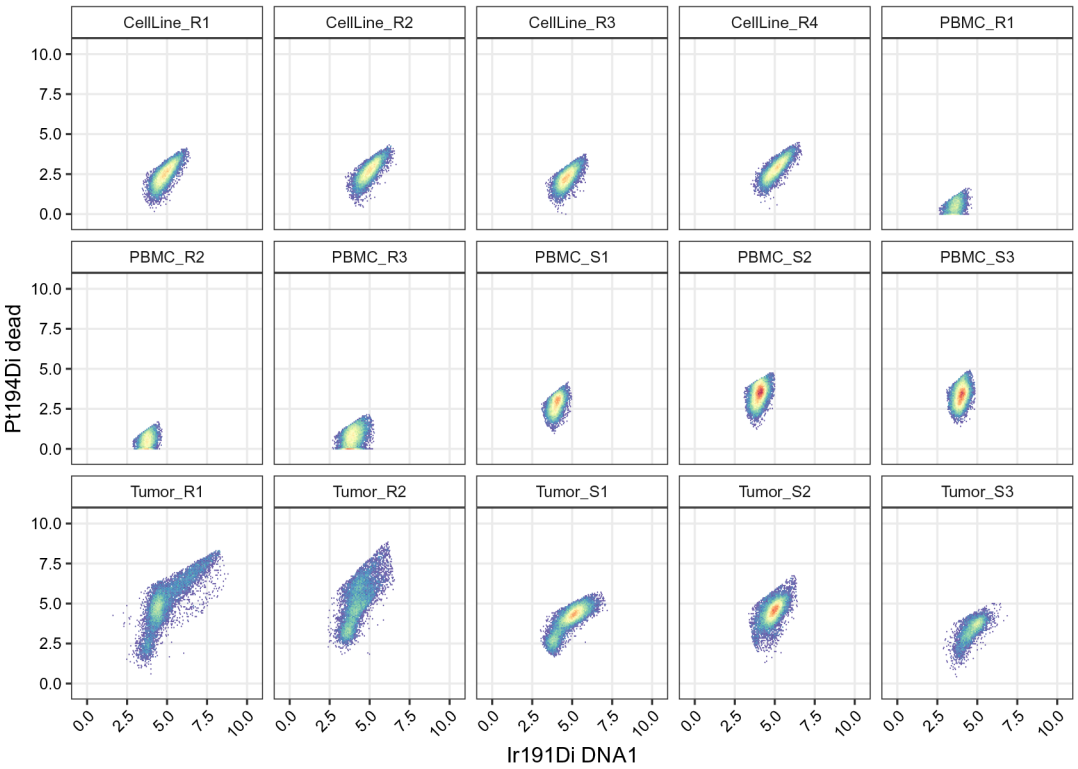

Figure 13. Scatter plots of dead cell channel against DNA, including the subset of cells remaining after live cell gating.

Quality control

Having completed the standard preprocessing steps, we proceed to investigate how the current experiment compares to prior experiments in terms of the number of cells in each reference and sample, and the expression levels of each target. Large parts of the metadata generated by now may no longer be needed, and unnecessarily increases output file sizes for large-scale datasets. Therefore, we will retain only two key cell metadata variables: `sample_id` containing the FCS filename each cell originates from, and `bc_id` containing the barcode population assignments. We secondly rename these variable to make the following quality control steps more intuitive.

```
# drop all cell metadata except file of origin & barcode IDs
colData(sce) <- colData(sce)[c("file_id", "bc_id")]

# rename cell metadata variable
i <- match("bc_id", names(colData(sce)))
names(colData(sce))[i] <- "sample"
```

In the debarcoding scheme used for deconvolution of the multiplexed samples (Section Debarcoding), barcode identifiers were chosen to contain all information relevant for each sample. This setup allows us to extract sample metadata directly from the `bc_ids`. Alternatively, and especially for more complex experimental designs, this information could be stored in a separate metadata table. Such a table could then be used to match the `bc_ids` with the listed samples, and add arbitrary metadata information (e.g., batch, patient ID, treatment).

In our example, barcode identifiers include each sample's type (`CellLine`, `PBMC` or `Tumor`), group (`R` for reference or `S` for sample of interest), and replicate number; and follow a consistent naming scheme: "<type>_<group><replicate>". We can easily extract these components and store them in the SCE's cell metadata (`colData`):

```
sce$type <- gsub("_.*", "", sce$sample)
sce$group <- gsub("[^R|S]", "", sce$sample)
i <- match(unique(sce$sample), sce$sample)
colData(sce)[sample(i, 10), ]
```

```
##  DataFrame with 10 rows and 4 columns
##      file_id      sample
##     <factor> <character>
##  1        V1 CellLine_R2
##  2        V1    Tumor_R1
##  3        V1     PBMC_R2
##  4        V1 CellLine_R3
##  5        V1    Tumor_S3
##  6        V1    Tumor_R2
##  7        V1     PBMC_S2
##  8        V1     PBMC_S3
##  9        V1 CellLine_R4
##  10       V1     PBMC_S1
##             type       group
##     <character> <character>
##  1     CellLine           R
##  2        Tumor           R
##  3         PBMC           R
##  4     CellLine           R
##  5        Tumor           S
##  6        Tumor           R
##  7         PBMC           S
##  8         PBMC           S
##  9     CellLine           R
##  10        PBMC           S
```

Quality control (QC) on reference cell counts

As a first quality control, we compare the cell counts of each reference sample (R) to those obtained from 7 previous experiments (Figure 14). Since the references are obtained from pre-barcoded aliquots of cells, the number of reference cells acquired gives direct information regarding the cell yield throughout the whole experiment: From cell barcoding to acquisition on the CyTOF. As shown in Figure 14, the current experiment tends to have a lower yield compared to average experiments.

```
# boxplot of current vs. reference cell counts
ref <- read.csv(file.path("data", "ref_cell_counts.csv"))
run <- c(table(sce$sample[sce$group == "R"]))

# join into single tidy data.frame
df <- bind_rows(ref, run, .id = "group")
df <- melt(df, id.var = "group")

ggplot(df, aes(variable, value)) +
    geom_boxplot(data = df[df$group == 1, ]) +
    geom_point(data = df[df$group == 2, ],
        col = "red", pch = 4, stroke = 1) +
    labs(x = "sample", y = "cell count") +
    qc_theme + ggtitle(
        "QC on reference cell counts",
        "[-] = reference | x = current experiment")
```

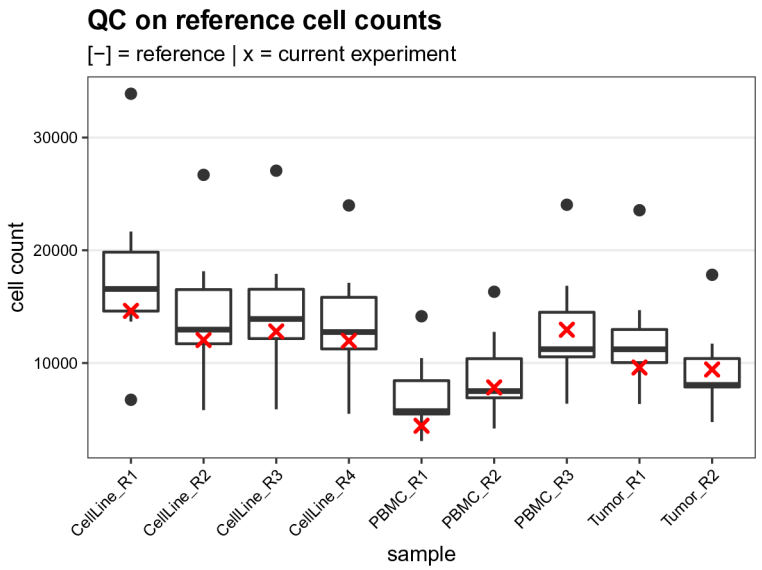

Figure 14. Reference cell count quality control. Boxplot comparing the reference cell counts obtained for the current experiment (red crosses) to those from 7 previously acquired experiments.

QC on sample cell counts

Secondly, we compare the cell counts for the 4 samples of interest (2 PBMC, 2 tumor samples) to the number of cells recorded for 14 tumor and PBMC samples each (28 samples in total) acquired in previous experiments (Figure 15). This step provides a first quality assessment of the samples of interest. Here, samples with too few cells will be less reliable, and potentially less representative of the original tissue, making conclusions from downstream analyses more difficult to draw.

```
ref <- read.csv(file.path("data", "sample_cell_counts.csv"))
run <- table(sce$sample[sce$group == "S"], dnn = "sample")
run <- as.data.frame(run, responseName = "count")
run$type <- sce$type[match(run$sample, sce$sample)]
df <- bind_rows(ref, run, .id = "group")

ggplot(df, aes(type, count)) +
    geom_boxplot(data = df[df$group == 1, ]) +
    geom_point(data = df[df$group == 2, ],
        col = "red", pch = 4, stroke = 1) +
    labs(x = "type", y = "cell count") +
    qc_theme + ggtitle(
        "QC on sample cell counts",
        "[-] = reference | x = current experiment")
```

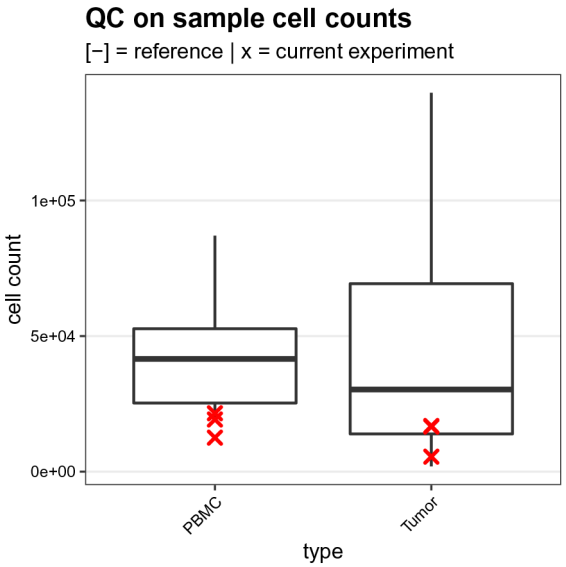

Figure 15. Sample cell count quality control. Boxplot comparing the sample cell counts obtained for the current experiment (red crosses) to those from 7 previously acquired experiments.

QC on mean marker intensities

As the third and final quality control, we compare the 98th expression quantiles across all targets of interest over the pooled references to those obtained from 7 previously acquired experiments available in metadata table `ref_marker_levels.csv` (Figure 16). We chose the 98th percentile to account for the fact that some populations are rare, and we are particularly interested in assessing signal stability for positive cells rather than the median of the population. Since the pooled references are identical from one experiment to another, this gives a direct indication of the current experiment's staining efficacy and enables early identification of antibody degradation over time.

```
# read in reference data
ref <- file.path("data", "ref_marker_levels.csv")
ref <- read.csv(ref, check.names = FALSE)

# compute 98th expression quantiles
# for reference samples in current experiment
es <- assay(sce, "compexprs")
es <- es[names(ref), sce$group == "R"]
run <- rowQuantiles(es, probs = 0.98)
```

```
# join into single tidy data.frame
df <- bind_rows(ref, run, .id = "group")
df <- melt(df, id.var = "group")

ggplot(df, aes(variable, value)) +
    geom_boxplot(data = df[df$group == 1, ]) +
    geom_point(data = df[df$group == 2, ],
        col = "red", stroke = 0.5) +
    labs(x = "target", y = "98th expression quantile") +
    qc_theme + ggtitle(
        "QC on marker levels",
        "[-] = referece | o = current experiment")
```

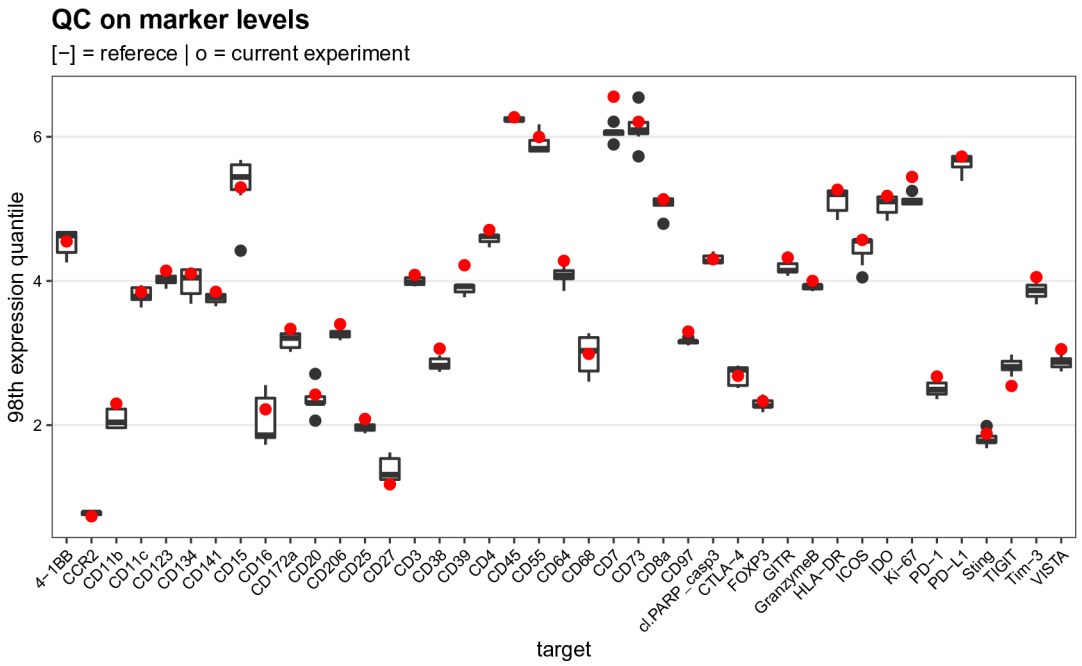

Figure 16. Mean marker expression quality control. Boxplot comparing the mean marker expression obtained for the current experiment (shown in red) to those from 7 previously acquired experiments.

Batch alignment

Each CyTOF experiment contains the same set of references. Similar to the approach used by Schuyler *et al.*[11], we use these references as anchors to calculate a channel-specific correction factor by dividing the 98th percentile measured in the current experiment by the average 98th percentile obtained across the first seven experiments of the project. The signal observed in each channel for the samples of interest is then divided by these correction factors derived from the reference samples.

```
# compute 98th count quantiles via back-transformation
# (using same cofactor as always) & average across replicates
cf <- int_metadata(sce)$cofactor
qs <- colMeans(sinh(ref)*cf)

# initialize correction factor of 1 for all channels
cfs <- setNames(rep(1, nrow(sce)), rownames(sce))
```

```
# compute batch correction factors for relevant channels
cs <- assay(sce, "compcounts")
csR <- cs[colnames(ref), sce$group == "R"]
run <- rowQuantiles(csR, probs = 0.98)
cfs[colnames(ref)] <- run / qs

# apply marker-specific batch correction (bc)
cs <- sweep(cs, 1, cfs, "/")
assay(sce, "bccounts") <- cs

# apply arcsinh-transformation
assay(sce, "bcexprs") <- asinh(cs/cf)
```

To visually assess the effect of the batch correction applied above, we compare the expression distributions before and after scaling (Figure 17). We additionally include 98th expression percentiles of both the (un)corrected samples as well as of the references used for computing correction factors. Percentiles are aligned with the references' upon correction while, even for the most affected channels (largest deviation from the references and, consequently, highest batch correction factors), distributions are very similar before and after scaling.

```
# subset most affected channels
top <- names(rev(sort(abs(cfs-1))))[seq(6)]
sub <- sce[top, sce$group == "R"]

# construct table of expressions
# before & after correction
as <- c(before = "compexprs", after = "bcexprs")
es <- lapply(as, function(a)
    data.frame(
        id = a,
        t(assay(sub, a)),
        check.names = FALSE))
df <- do.call(rbind, es)
df <- melt(df, id.var = "id")
df$id <- factor(df$id, as, names(as))

# compute 98th percentiles of samples
q98_df <- df %>%
    group_by(id, variable) %>%
    summarize_at("value", quantile, 0.98)

# compute 98th percentiles of references (average across 7)
ref_df <- data.frame(variable = colnames(ref), value = colMeans(ref))[top, ]
ref_df <- bind_rows(.id = "id", list(before = ref_df, after = ref_df))

ggplot(df, aes(value, ..density../max(..density..), col = id)) +
    facet_wrap(~ variable) +
    geom_density(size = 0.5, show.legend = FALSE) +
    geom_vline(data = ref_df, aes(xintercept = value), lty = 2) +
    geom_point(data = q98_df, aes(value, 0.5, col = id), size = 2) +
    scale_color_manual(NULL, values = c( "royalblue", "tomato")) +
    scale_x_continuous(limits = c(-0.5, NA)) +
    labs(x = "expression", y = "scaled density") +
    qc_theme + theme(
        aspect.ratio = 2/3,
        panel.grid = element_blank(),
        legend.key.size = unit(0.5, "lines"))
```

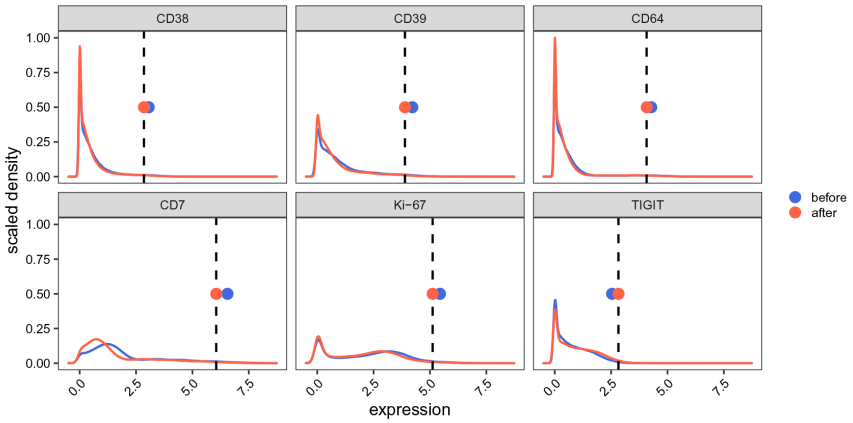

Figure 17. Batch alignment quality control. Expression distributions before (blue) and after (red) quantile scaling using 7 previously acquired experiments as reference. Included are the 6 most affected channels (i.e., highest absolute correction factors). Dashed lines indicate 98th expression percentiles averaged across references; points represent the respective distributions' 98th percentiles.

Discussion

In this workflow, we have presented a pipeline for reproducible and highly-automated preprocessing of CyTOF data, based on an updated version of `CATALYST`. Our pipeline covers four standard steps: Normalization for signal time-drift using bead standards (Section Normalization), single-cell deconvolution of multiplexed samples (Section Debarcoding), correction for spillover via compensation (Section Compensation), and gating for live cells (Section Gating). Moreover, we have included various quality control steps that compare the current experiment to a set of reference data (Section Quality control). These steps ensure that measurement sensitivity, gating cell yields, sample cell counts, and expression levels lie within the expected range.

A key advantage of both using and developing Bioconductor packages is that they utilize common data structures, thereby greatly facilitating interaction between them. For example, many of the data structures used in scRNA-seq data analysis have only become established relatively recently. Meanwhile, the cytometry community has been relying on the FCS file format for data storage, and *flowCore*'s `flowFrame/flowSet` as well as *flowWorkspace*'s `GatingSet` classes for computational analyses. While there exists a lot of infrastructure around these data structures, they impede method development for newly emerging standards, and act as a barrier for interpolation of analyses across currently developed packages. This is particularly visible in the context of other fast growing single-cell data types such as scRNA-seq data analysis, where most current methods are being developed around Bioconductor's `SingleCellExperiment` class. To name just two examples, an extensive collection of visualization tools for SCEs is available through *scater*[27], including a variety of dimensionality reduction methods; and methods for differential abundance (DA) analysis (to detect subpopulations that are differently abundant between conditions) and differential state (DS) analysis (to test for subpopulation-specific expression changes across conditions) are implemented in *diffcyt*[28].

The SCE class allows storing multiple `assays` that can, for example, contain raw counts, expression-like data obtained upon arcsinh-transformation, as well as any intermediate data matrices obtained after normalization, compensation and batch correction. Moreover, any event (cell) and feature (marker) metadata generated in the process can be added to the object's `colData/rowData`, alongside an arbitrary number of dimensionality reductions. Thus, SCEs present an overall more compact and less error-prone data structure for both preprocessing and downstream analysis when compared to storing the various data matrices or metadata in separate variables, which would have to be combined for certain computations, separately subsetted and saved to independent outputs.

There is an obvious benefit for the mass cytometry community to take advantage of these new infrastructure developments. However, it is equally important to maintain backward compatibility with well-established standards in the field. For example, it can be desirable to write out intermediate outputs (FCS files) after each preprocessing step, or make use of available tools that build around `flowCore`'s `flowFrame` and `flowSet` classes, or other classes derived thereof (e.g., *flowWorkspace*'s `GatingSet`). Thus, while `CATALYST`'s transition to a more recent and an arguably advantageous data structure is motivated by the ability to leverage many existing and newly-developed tools, a complete dismissal of the large infrastructure that is available in the realm of cytometry data analysis is impossible at this time. To facilitate conversion between SCEs and conventional

cytometry data structures, `CATALYST` provides the `sce2fcs()` function, which allows the user to specify which assay data to retain, whether to drop or keep available cell metadata and dimensionality reductions, and (optionally) to split the input dataset by a non-numeric variable (to, e.g., export each sample to a separate FCS file).

Although the current version of this pipeline constitutes a comprehensive approach to generate high-quality data for downstream analysis, further developments could be added in the future. In particular, it could be useful to implement an automated way of identifying and removing part of the data with unstable signal, similar to the approach proposed by *flowClean*[29], an R package designed to exclude fluorescent anomalies in flow cytometry data. Given that selection of anomalies in the dataset by the user is subjective, or that they may be altogether undetectable by eye, the advantage of such an approach would be to further standardize the process while decreasing manual work.

Recently, batch normalization has become of increased importance in order to enable integration of datasets acquired at different times, by different users and on different instruments. Here, we use scaling normalization where references are used as anchors to correct all samples included in the analysis in a channel specific way, similar to the strategy proposed by Schuyler *et al.*[11]. While this approach requires a well-defined experimental procedure where references with positive and negative subsets for each marker have to be included in each experiment, it does not make any assumptions on sample compositions. Thus, since the dataset used in this pipeline was acquired on the same instrument and stained with the same frozen antibody panel as previous experiments, scaling by expression quantiles provides an efficient way to correct for batch effects.

To increase the flexibility of batch correction in cases where the experimental variation is higher, `CATALYST` could integrate different methods that have the potential to increase batch correction efficiency. For example, *CytoNorm*[12] computes quantiles for every metacluster and for every marker after aggregation of control samples from each batch. Such an approach could be more appropriate in cases where the references' expression distributions are less aligned. An alternative method, *CytofRUV*[30], exploits the concept of pseudo-replicates to remove unwanted variation (RUV) between proteins and cells. Here, cells are grouped into subpopulations using *FlowSOM*[31] clustering. Groups of cells present across all batches are considered to be pseudo-replicates, and their deviation (residuals) from the average signal across batches is used to estimate and correct for the batch effects.

Although various methods to correct for batch effects in both the presence and absence of references have been proposed, a systematic comparison of batch correction tools for mass cytometry data is missing. Thus, whether the approach used in this study to align batches on the basis of shared references is the most accurate remains unresolved.

Our pipeline is entirely R-based and does not rely on switching between platforms. Thus, it omits the need for heavy data transfers between online cloud services, graphical user interfaces (GUI), and programming environments for different parts of preprocessing and downstream analysis. As a result, each step in the analysis is fully reproducible and any parameters used throughout can be easily modified and documented. This transition from manual, GUI-based to largely automated, programmatic data processing is indispensable for clinical and other large-scale studies, where sample throughput is high and reproducibility ever so important.

Since its first submission to Bioconductor in 2017, `CATALYST` has undergone continuous maintenance and development. The most noteworthy changes include implementation of a comprehensive visualization suite based on Nowicka *et al.*[14] 's workflow for differential discovery; and, the transition from custom data structures to using Bioconductor's `SingleCellExperiment` class for differential analysis with Bioconductor v3.11, and for preprocessing with v3.12. Taken together, these developments have transformed `CATALYST` into a one-stop tool for cytometry data analysis, enabling both data preprocessing and in-depth downstream analysis.

Methods

Normalization

Identification of bead events. Commonly, bead events are identified by manual gating on scatter plots of DNA vs. bead channels where DNA should be low, and expression should be high across all bead channels. Instead, we propose a programmatic way to identify beads that includes detection of bead-bead and cell-bead doublets.

Our normalization strategy leverages the already established SCD algorithm for preliminary tagging of events as beads. In this context, the debarcoding scheme is a $2\times(2+m)$ matrix (Figure 18). Here, columns correspond to the two DNA channels and m barcode channels; rows correspond to barcodes 0 (no bead) and 1 (is bead), where non-bead events are positive for DNA channels only (barcode 11000. . .), while bead events are negative for DNA and positive for all bead channels (barcode 00111. . .):

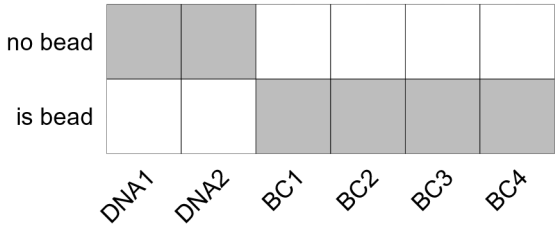

Figure 18. Schematic of the debarcoding scheme used by 'CATALYST"s 'normCytof()' function to identify bead events. Rows correspond to barcodes, columns to DNA and bead channels. Each barcode is either positive (grey) or negative (blank) for a given channel; cells (barcode 11000...) are positive for DNA and negative for bead channels, bead events (barcode 00111...) are negative for DNA and positive for bead channels.

Upon initial assignment of bead events, we apply a *median* ± *n* median absolute deviation (MAD) rule to remove low- and high-signal events from the bead population used for estimating normalization factors. As *n* decreases, bead populations become more narrow and bead-bead doublets are excluded. The extent to which bead populations are trimmed can be adjusted via argument `trim` (default 5).

Notably, slight *over*-trimming does not affect normalization. It is therefore recommended to choose a `trim` value that is small enough to assure removal of doublets at the cost of reduced bead population sizes.

Correcting for signal-decrease over time. To correct for the effect of event acquisition time on signal intensity, we follow the method proposed by Finck *et al.*[8]. In essence, bead intensities are smoothed using a median sliding-window with width *k* (default 500 bead events). At each timepoint, the slope of a line with intercept zero is computed by minimizing the squared error between smoothed bead and mean bead intensities (= baseline). Alternatively, a reference set of beads from which to compute the baseline can be provided. Slopes for non-bead timepoints are obtained via constant interpolation of these slopes. Here, large slopes correspond to significant deviation from the baseline, and small slopes indicate that the signal is already similar to the baseline. Thus, raw bead counts are normalized by multiplication with the fitted slopes at each timepoint.

Debarcoding

Preliminary barcode assignment. The debarcoding process commences by assigning each event a preliminary barcode ID. This requires specification of a binary barcoding scheme (or debarcoding key)

$$B = (b_{ij}) \in \{0,1\}^{n \times m}$$

where $i = 1, ..., n$ is the barcode index, $j = 1, ..., m$ a barcode channel, and n, m denote the number of unique barcodes and barcoding channels, respectively. Further, let k_i denote the number of positive barcoding channels for barcode i: $k_i = \sum_{j=1}^{m} b_{ij}$.

If $k_i = k \; \forall \; i = 1, ..., n$ (i.e., every barcode has the same number of positive barcoding channels), the k channels with the highest signal in a given event are considered to be positive, the remaining $m - k$ to be negative. The *separation* δ of positive and negative events is then defined as the difference between the kth highest and $(m - k)$th lowest scaled intensity for that event.

Seperation cutoff estimation. When the separation between positive and negative barcoding channels is low, we cannot be confident in the barcode assignment.

For the estimation of cutoff parameters, we consider yields upon debarcoding as a function of the applied cutoffs. Commonly, this function will be characterized by an initial weak decline, where doublets are excluded, and subsequent rapid decline in yields to zero. In between, low numbers of counts with intermediate barcode separation give rise to a plateau. To facilitate robust estimation, we fit a linear and a three-parameter log-logistic function[32] to the yields function with *drc*'s `LL.R` function[33] (Figure 19). As an adequate cutoff estimate, we target a point that marks the end of the plateau regime and on-set of yield decline to appropriately balance confidence in barcode assignment and cell yield.

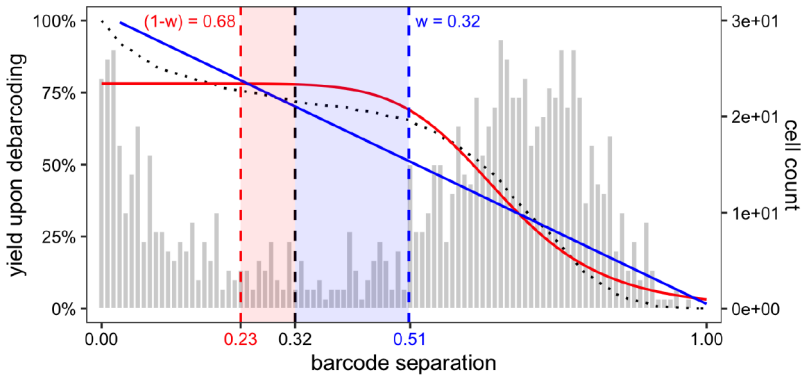

Figure 19. Schematic description of automated separation cutoff estimation. Bar graphs represent the distribution of cells relative to the barcode distance, dotted line scorresponds to yield upon debarcoding as a function of the applied separation cutoff. The yield curve is fitted with a linear regression (blue) and a three parameter log-logistic function (red). The final cutoff estimate (black dashed line) is defined as the mean of estimates derived from both fits, weighted with the goodness of the respective fit.

We define the linear model cutoff estimate c_{LM} as the value for which the cell yield Y has declined to half of the initial Yield β_0:

$$Y = \beta_0 + c_{linear} \cdot \beta_1 = \beta_0 / 2 \Leftrightarrow c_{linear} = -\beta_0 /(2 \cdot \beta_1)$$

where β_0, β_1 are the intercept and slope of the linear model fit, respectively.

We define the log-logistic model cutoff estimate c_{LLM} as the value for which the log-logistic function's decline is minimized relative to its value:

$$c_{\text{log-logistic}} = \arg\min_x \frac{\left|f'(x)\right|}{f(x)} > 0.1$$

The final cutoff estimate c is defined as the weighted mean between these estimates:

$$c = w \cdot c_{linear} + (1 - w) \cdot c_{log-logistic}$$

where w is the goodness of the linear fit relative to the log-logistic fit:

$$w = \frac{\mathrm{RSS}_{log-logistic}}{\mathrm{RSS}_{log\text{-}logistic} + \mathrm{RSS}_{linear}}$$

Compensation

Retrieval of real signal. As in conventional flow cytometry, we model spillover linearly, with the channel stained for as predictor, and spill-effected channels as response. Thus, the intensity observed in a given channel j are a linear combination of its real signal and contributions of other channels that spill into it. Let I denote the (unknown) real and J the observed signal. Further, let s_{ij} be the proportion of channel j signal that is due to channel i, and w_j the set of channels that spill into channel j. Then

$$J_j = I_j + \sum_{i \in w_j} s_{ij}$$

In matrix notation, measurement intensities may be viewed as the convolution of real intensities and a spillover matrix $SM = (s_{ij}) \in \mathbb{R}_+^{n \times p}$, where n denotes the number of samples (cells) and p the number of features (channels): $J = I \cdot SM$. The real signal I can then be retrieved via:

$$I = J \cdot SM^{-1} = J \cdot CM$$

where SM^{-1} is termed compensation matrix (CM).

While mathematically exact, the solution to this equation will yield negative values, and does not account for the fact that ion counts are strictly non-negative. A computationally efficient way to adress this is to instead use non-negative linear least squares (NNLS), which optimizes the least squares criterion under the constraint of non-negativity:

$$\min\{(J - SM \cdot I)^T \cdot (J - SM \cdot I)\} \,|\, I \geq 0$$

To solve for I, we apply the Lawson-Hanson algorithm[34,35] for NNLS implemented in the *nnls* package.

Spillover estimation. Because any signal not in a single stain experiment's primary channel j results from channel crosstalk, each spill entry s_{ij} can be approximated by the slope of a linear regression with channel j signal as the response, and channel i signals as the predictors, where $i \in w_j$. `computeSpillmat()` offers two alternative ways for spillover estimation (20).

The `default` method approximates this slope with the following single-cell derived estimate: Let i^+ denote the set of cells that are positive in channel i, and s_{ij}^c be the channel i to j spill computed for a cell c that has been assigned to this population. We approximate s_{ij}^c as the ratio between the signal in unstained spillover receiving and stained spillover emitting channel, I_j and I_i, respectively. The expected background in these channels, m_j^- and m_i^-, is computed as the median signal of events that are i) negative in the channels for which spill is estimated (i and j); ii) not assigned to potentionally interacting channels; and, iii) not unassigned, and subtracted from all measurements:

$$s_{ij}^c = \frac{I_j - m_j^{i-}}{I_i - m_i^{i-}}$$

Each entry s_{ij} in SM is then computed as the median spillover across all cells $c \in i^+$:

$$s_{ij} = \text{med}(s_{ij}^c \mid c \in i^+)$$

In a population-based fashion, as done in conventional flow cytometry, s_{ij} is computed as the slope of a line through the medians (or trimmed means) of stained and unstained populations, m_j^+ and m_i^+, respectively. Background signal is computed as above and subtracted, according to:

$$s_{ij} = \frac{m_j^+ - m_j^-}{m_i^+ - m_i^-}$$

On the basis of their additive nature, spill values are estimated independently for every pair of interacting channels. Hereby, we take into account only interactions that are sensible from a chemical and physical point of view: $M \pm 1$ channels (abundance sensitivity), $M + 16$ channels (oxide formation), and channels measuring isotopes (impurities; Figure 21).

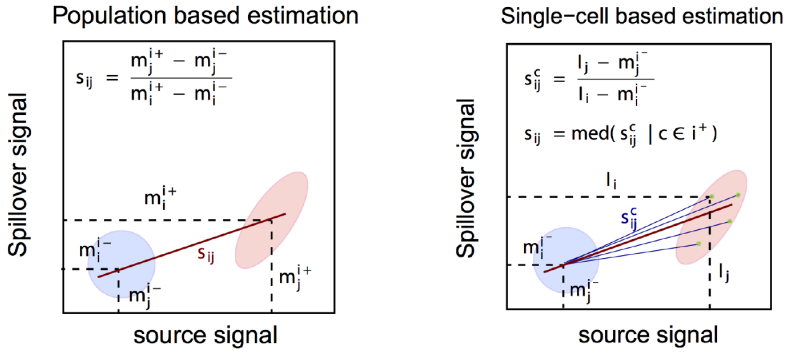

Figure 20. Population versus single-cell based spillover estimation. In a population-based setting (left), spillover is estimated as the slope of a line through the centers of positive (red) and negative (blue) populations. In a single-cell based setting (right), slopes are computed independently for each cell in the positive population, and spillover is estimated as their median.

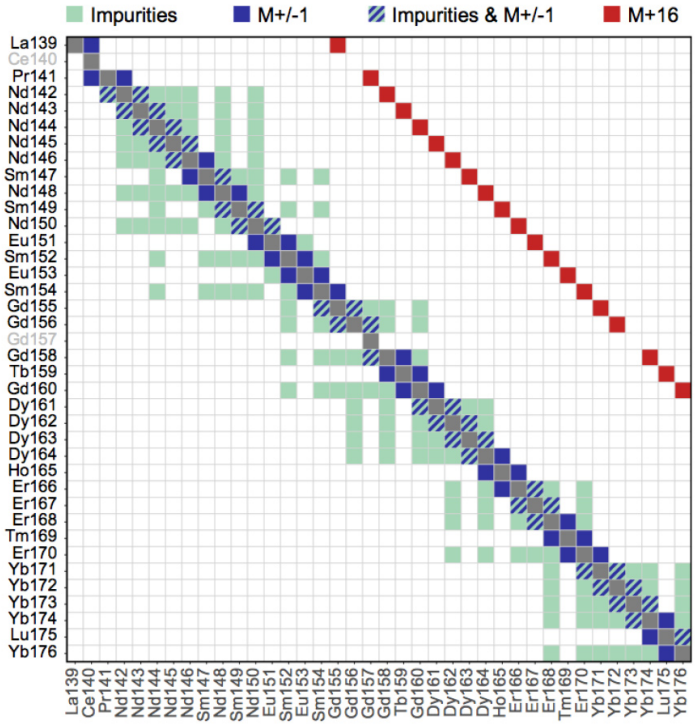

Figure 21. Heatmap of channel interactions expected to exhibit spillover. Included are only interactions that are sensible from a chemical and physical point of view: adjacent mass channels (abundance sensitivity), +16 mass channels (oxidation), and channels measuring isotopes (impurities).

Alternatively, `interactions = "all"` will compute a spill estimate for all $n \cdot (n - 1)$ possible interactions, where n denotes the number of measurement parameters. Estimates falling below the threshold specified by `th` will be set to zero. Lastly, note that diagonal entries $s_{ii} = 1$ for all $i \in 1, ..., n$, so that spill is relative to the total signal measured in a given channel.

Data availability

Underlying data

The CyTOF data as well as all metadata required to run the full pipeline presented herein are available from Figshare as well as the Tumor Profiler website at https://tu-pro.ch/download/catalyst.

Figshare: An R-based reproducible and user-friendly preprocessing pipeline for CyTOF data. https://doi.org/10.6084/m9.figshare.c.5063984.v1

This project contains the following underlying data:

- CyTOF_acquisition_1-3.fcs (40-Ab panel CyTOF data of 2 blood and 2 tumor samples, and 9 reference samples selected to contain positive and negative populations for each marker included in the study's Ab- panel. Samples were multiplexed with a 20-well barcoding plate, and obtained from a single experiment provided as 3 FCS files.)

- normalization_beads.fsc (Beads identified using 'CATALYST' during the normalization step of a previous CyTOF experiment. – Used as reference beads to correct for changes in signal sensitivity over time across multiple CyTOF experiments.)

- ref_bead_counts.csv (A table of mean dual counts for the six different bead channels (columns) obtained from 7 previous experiments (rows). – Used as a reference to assess the measurement sensitivity for the current experiment.)

- debarcoding_scheme.csv (A binary barcoding scheme of 6-choose-3 = 20 barcodes with columns cor- responding to barcode channel masses (101, 104, 105, 106, 108, 110) and rows corresponding to

barcodes (7 empty, 9 references, 2 PBMC and 2 tumor samples) – Used for single-cell deconvolution of multiplexed of samples.)

- spillover_matrix.csv (A spillover matrix calculated with 'CATALYST' from beads single-stained with each of the 40 antibodies included in the panel used in this study. The matrix contains, for each measurement channel (rows), the percentage of signal received by all other channels (columns). – Used for correction of spillover.)

- ref_cell_counts.csv (A table of the number of cells measured in 7 previous experiments, each including 4 cell line, 3 PBMC and 2 tumor references samples (63 samples in total). – Used to assess reference sample cell yields in the current in comparison to previous experiments.)

- sample_cell_counts.csv (A table of the number of cells measured in 7 previous experiments, each including 2 PBMC and 2 tumor samples (28 samples in total). – Used to assess sample cell yields in the current in comparison to previous experiments.)

- ref_marker_levels.csv (A table of the 98th expression percentiles for each target (columns) across 7 previous experiments (rows). – Used to assess the staining efficiency of the current experiment.)

Software availability

Analyses were run in R v4.2.0[36], with Bioconductor v3.15[37], and all software packages used throughout this workflow are publicly available through the Comprehensive R Archive Network (https://cran.r-project.org) or the Bioconductor project (http://bioconductor.org). Specific package versions are captured in the following session information:

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Monterey 12.2
##
## Matrix products: default
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    stats     graphics
## [4] grDevices utils     datasets
## [7] methods   base
##
## other attached packages:
##  [1] reshape2_1.4.4
##  [2] patchwork_1.1.1
##  [3] openCyto_2.8.0
##  [4] mvtnorm_1.1-3
##  [5] ggcyto_1.24.0
##  [6] ncdfFlow_2.42.0
##  [7] BH_1.78.0-0
##  [8] RcppArmadillo_0.11.1.1.0
##  [9] ggplot2_3.3.6
## [10] flowWorkspace_4.8.0
## [11] flowCore_2.8.0
## [12] dplyr_1.0.9
## [13] BiocStyle_2.24.0
## [14] vespa_0.99.0
## [15] CATALYST_1.21.1
```

```
## [16] SingleCellExperiment_1.18.0
## [17] SummarizedExperiment_1.26.1
## [18] Biobase_2.56.0
## [19] GenomicRanges_1.48.0
## [20] GenomeInfoDb_1.32.2
## [21] IRanges_2.30.0
## [22] S4Vectors_0.34.0
## [23] BiocGenerics_0.42.0
## [24] MatrixGenerics_1.8.0
## [25] matrixStats_0.62.0
## [26] RColorBrewer_1.1-3
## [27] testthat_3.1.4
##
## loaded via a namespace (and not attached):
##   [1] scattermore_0.8
##   [2] SpatialExperiment_1.6.0
##   [3] R.methodsS3_1.8.2
##   [4] tidyr_1.2.0
##   [5] knitr_1.39
##   [6] irlba_2.3.5
##   [7] multcomp_1.4-19
##   [8] DelayedArray_0.22.0
##   [9] R.utils_2.11.0
##  [10] data.table_1.14.2
##  [11] rpart_4.1.16
##  [12] RCurl_1.98-1.7
##  [13] doParallel_1.0.17
##  [14] generics_0.1.2
##  [15] ScaledMatrix_1.4.0
##  [16] callr_3.7.0
##  [17] cowplot_1.1.1
##  [18] TH.data_1.1-1
##  [19] usethis_2.1.6
##  [20] ggpointdensity_0.1.0
##  [21] spatstat.data_2.2-0
##  [22] xml2_1.3.3
##  [23] assertthat_0.2.1
##  [24] viridis_0.6.2
##  [25] xfun_0.31
##  [26] evaluate_0.15
##  [27] DEoptimR_1.0-11
##  [28] fansi_1.0.3
##  [29] tmvnsim_1.0-2
##  [30] Rgraphviz_2.40.0
##  [31] igraph_1.3.1
##  [32] DBI_1.1.2
##  [33] spatstat.geom_2.4-0
##  [34] purrr_0.3.4
##  [35] ellipsis_0.3.2
##  [36] ks_1.13.5
##  [37] ggnewscale_0.4.7
##  [38] ggpubr_0.4.0
##  [39] backports_1.4.1
##  [40] bookdown_0.26
##  [41] cytolib_2.8.0
##  [42] BiocWorkflowTools_1.22.0
##  [43] RcppParallel_5.1.5
##  [44] deldir_1.0-6
##  [45] sparseMatrixStats_1.8.0
##  [46] vctrs_0.4.1
##  [47] remotes_2.4.2
##  [48] abind_1.4-5
##  [49] cachem_1.0.6
```

```
##  [50] withr_2.5.0
##  [51] ggforce_0.3.3
##  [52] aws.signature_0.6.0
##  [53] robustbase_0.95-0
##  [54] prettyunits_1.1.1
##  [55] mnormt_2.0.2
##  [56] mclust_5.4.10
##  [57] goftest_1.2-3
##  [58] cluster_2.1.3
##  [59] crayon_1.5.1
##  [60] drc_3.0-1
##  [61] edgeR_3.38.1
##  [62] pkgconfig_2.0.3
##  [63] labeling_0.4.2
##  [64] tweenr_1.0.2
##  [65] vipor_0.4.5
##  [66] nlme_3.1-157
##  [67] pkgload_1.2.4
##  [68] devtools_2.4.3
##  [69] rlang_1.0.2
##  [70] lifecycle_1.0.1
##  [71] sandwich_3.0-1
##  [72] rsvd_1.0.5
##  [73] rprojroot_2.0.3
##  [74] polyclip_1.10-0
##  [75] flowClust_3.34.0
##  [76] graph_1.74.0
##  [77] Matrix_1.4-1
##  [78] carData_3.0-5
##  [79] Rhdf5lib_1.18.2
##  [80] zoo_1.8-10
##  [81] beeswarm_0.4.0
##  [82] base64enc_0.1-3
##  [83] ggridges_0.5.3
##  [84] GlobalOptions_0.1.2
##  [85] processx_3.5.3
##  [86] pheatmap_1.0.12
##  [87] viridisLite_0.4.0
##  [88] png_0.1-7
##  [89] rjson_0.2.21
##  [90] bitops_1.0-7
##  [91] R.oo_1.25.0
##  [92] ConsensusClusterPlus_1.60.0
##  [93] KernSmooth_2.23-20
##  [94] rhdf5filters_1.8.0
##  [95] DelayedMatrixStats_1.18.0
##  [96] shape_1.4.6
##  [97] stringr_1.4.0
##  [98] brew_1.0-7
##  [99] spatstat.random_2.2-0
## [100] jpeg_0.1-9
## [101] rstatix_0.7.0
## [102] ggsignif_0.6.3
## [103] aws.s3_0.3.21
## [104] beachmat_2.12.0
## [105] scales_1.2.0
## [106] memoise_2.0.1
## [107] magrittr_2.0.3
## [108] plyr_1.8.7
## [109] hexbin_1.28.2
## [110] zlibbioc_1.42.0
## [111] hdrcde_3.4
## [112] compiler_4.2.0
```

```
## [113] dqrng_0.3.0
## [114] plotrix_3.8-2
## [115] clue_0.3-61
## [116] rrcov_1.7-0
## [117] cli_3.3.0
## [118] XVector_0.36.0
## [119] ps_1.7.0
## [120] FlowSOM_2.4.0
## [121] MASS_7.3-57
## [122] mgcv_1.8-40
## [123] tidyselect_1.1.2
## [124] stringi_1.7.6
## [125] RProtoBufLib_2.8.0
## [126] yaml_2.3.5
## [127] BiocSingular_1.12.0
## [128] locfit_1.5-9.5
## [129] latticeExtra_0.6-29
## [130] ggrepel_0.9.1
## [131] grid_4.2.0
## [132] tools_4.2.0
## [133] parallel_4.2.0
## [134] CytoML_2.8.0
## [135] circlize_0.4.15
## [136] rstudioapi_0.13
## [137] git2r_0.30.1
## [138] foreach_1.5.2
## [139] gridExtra_2.3
## [140] farver_2.1.0
## [141] Rtsne_0.16
## [142] DropletUtils_1.16.0
## [143] BiocManager_1.30.18
## [144] digest_0.6.29
## [145] pracma_2.3.8
## [146] Rcpp_1.0.8.3
## [147] car_3.0-13
## [148] broom_0.8.0
## [149] scuttle_1.6.2
## [150] fda_6.0.3
## [151] IDPmisc_1.1.20
## [152] httr_1.4.3
## [153] ComplexHeatmap_2.12.0
## [154] flowStats_4.8.0
## [155] colorspace_2.0-3
## [156] rainbow_3.6
## [157] brio_1.1.3
## [158] XML_3.99-0.9
## [159] fs_1.5.2
## [160] tensor_1.5
## [161] splines_4.2.0
## [162] RBGL_1.72.0
## [163] spatstat.utils_2.3-1
## [164] scater_1.24.0
## [165] sessioninfo_1.2.2
## [166] fds_1.8
## [167] jsonlite_1.8.0
## [168] corpcor_1.6.10
## [169] R6_2.5.1
## [170] pillar_1.7.0
## [171] htmltools_0.5.2
## [172] nnls_1.4
## [173] glue_1.6.2
## [174] fastmap_1.1.0
## [175] BiocParallel_1.30.3
```

```
## [176] deSolve_1.32
## [177] BiocNeighbors_1.14.0
## [178] codetools_0.2-18
## [179] pcaPP_2.0-1
## [180] pkgbuild_1.3.1
## [181] utf8_1.2.2
## [182] lattice_0.20-45
## [183] spatstat.sparse_2.1-1
## [184] tibble_3.1.7
## [185] flowViz_1.60.0
## [186] ggbeeswarm_0.6.0
## [187] curl_4.3.2
## [188] colorRamps_2.3.1
## [189] gtools_3.9.2.1
## [190] magick_2.7.3
## [191] survival_3.3-1
## [192] limma_3.52.1
## [193] roxygen2_7.2.0
## [194] rmarkdown_2.14
## [195] desc_1.4.1
## [196] munsell_0.5.0
## [197] GetoptLong_1.0.5
## [198] rhdf5_2.40.0
## [199] GenomeInfoDbData_1.2.8
## [200] iterators_1.0.14
## [201] HDF5Array_1.24.1
## [202] gtable_0.3.0
## [203] spatstat.core_2.4-4
```

Consent

Written informed consent for publication of the tumor and blood samples was obtained from the patients (BASEC-Nr.2018-02050, approved by the Kantonal Ethics Commisions of Zurich and Basel).

Author contributions

HLC and SC developed methodology, ran analyses, and drafted the manuscript. HLC implemented code in `CATALYST`. SC tested and gave input on the package. AJ and SS performed experiments. MDR and BB gave feedback on the manuscript. All authors read and approved the final manuscript and agreed to its content.

Acknowledgments

The authors would like to acknowledge support from the Biobank team of the Department of Dermatology at University Hospital Zurich, who provided us with the samples used in this study.

Tumor Profiler Consortium members and affiliations

Rudolf Aebersold[2], Faisal S Al-Quaddoomi[7,14], Jonas Albinus[6], Ilaria Alborelli[22], Sonali Andani[2,5,14,24,27], Per-Olof Attinger[9], Marina Bacac[13], Daniel Baumhoer[22], Beatrice Beck-Schimmer[34], Niko Beerenwinkel[3], Christian Beisel[3], Lara Bernasconi[25], Anne Bertolini[7,14], Bernd Bodenmiller[31], Ximena Bonilla[2,5,14,24], Ruben Casanova[31], Stéphane Chevrier[31], Natalia Chicherova[7,14], Maya D'Costa[8], Esther Danenberg[32], Natalie Davidson[2,5,14,24], Monica-Andreea Dra˘gan[3], Reinhard Dummer[26], Stefanie Engler[31], Martin Erkens[11], Katja Eschbach[3], Cinzia Esposito[32], André Fedier[15], Pedro Ferreira[3], Joanna Ficek[2,5,14,24], Anja L Frei[27], Bruno Frey[10], Sandra Goetze[6], Linda Grob[7,14], Gabriele Gut[32], Detlef Günther[4], Martina Haberecker[27], Pirmin Haeuptle[1], Viola Heinzelmann-Schwarz[15,21], Sylvia Herter[13], Rene Holtackers[32], Tamara Huesser[13], Anja Irmisch[26], Francis Jacob[15], Andrea Jacobs[31], Tim M Jaeger[9], Katharina Jahn[3], Alva R James[2,5,14,24], Philip M Jermann[22], André Kahles[2,5,14,24], Abdullah Kahraman[14,27], Viktor H Koelzer[27], Werner Kuebler[23], Jack Kuipers[3], Christian P Kunze[20], Christian Kurzeder[18], Kjong-Van Lehmann[2,5,14,24], Mitchell Levesque[26], Sebastian Lugert[8], Gerd Maass[10], Philipp Markolin[2,5,14,24], Julien Mena[2], Ulrike Menzel[3], Nicola Miglino[1], Emanuela S Milani[6], Holger Moch[27], Simone Muenst[22], Riccardo Murri[33], Charlotte KY Ng[22,30], Stefan Nicolet[22], Marta Nowak[27], Patrick GA Pedrioli[2], Lucas Pelkmans[32], Salvatore Piscuoglio[15,22], Michael Prummer[7,14], Mathilde Ritter[15], Christian Rommel[11], María L Rosano-González[7,14], Gunnar Rätsch[2,5,14,24], Natascha Santacroce[3], Jacobo Sarabia del Castillo[32], Ramona Schlenker[12], Petra C Schwalie[11], Severin Schwan[9], Tobias Schär[3], Gabriela Senti[25], Franziska Singer[7,14], Sujana Sivapatham[31],

Berend Snijder[2], Bettina Sobottka[27], Vipin T Sreedharan[7,14], Stefan Stark[2,5,14,24], Daniel J Stekhoven[7,14], Tinu M Thomas[2,5,14,24], Markus Tolnay[22], Vinko Tosevski[13], Nora C Toussaint[7,14], Mustafa A Tuncel[3], Audrey Van Drogen[6], Marcus Vetter[17], Tatjana Vlajnic[22], Sandra Weber[25], Walter P Weber[16], Rebekka Wegmann[2], Michael Weller[29], Fabian Wendt[6], Norbert Wey[27], Andreas Wicki[1,15,19], Bernd Wollscheid[6], Shuqing Yu[7,14], Johanna Ziegler[26], Marc Zimmermann[2,5,14,24], Martin Zoche[27], Gregor Zuend[28]

[1]Cantonal Hospital Baselland, Medical University Clinic, Rheinstrasse 26, 4410 Liestal, Switzerland

[2]ETH Zurich, Department of Biology, Wolfgang-Pauli-Strasse 27, 8093 Zurich, Switzerland

[3]ETH Zurich, Department of Biosystems Science and Engineering, Mattenstrasse 26, 4058 Basel, Switzerland

[4]ETH Zurich, Department of Chemistry and Applied Biosciences, Vladimir-Prelog-Weg 1-5/10, 8093 Zurich, Switzerland

[5]ETH Zurich, Department of Computer Science, Institute of Machine Learning, Universitätstrasse 6, 8092 Zurich, Switzerland

[6]ETH Zurich, Department of Health Sciences and Technology, Otto-Stern-Weg 3, 8093 Zurich, Switzerland

[7]ETH Zurich, NEXUS Personalized Health Technologies, John-von-Neumann-Weg 9, 8093 Zurich, Switzerland

[8]F. Hoffmann-La Roche Ltd, Grenzacherstrasse 124, 4070 Basel, Switzerland

[9]F. Hoffmann-La Roche Ltd, Grenzacherstrasse 124, 4070 Basel, Switzerland

[10] Roche Diagnostics GmbH, Nonnenwald 2, 82377 Penzberg, Germany

[11]Roche Pharmaceutical Research and Early Development, Roche Innovation Center Basel, Grenzacherstrasse 124, 4070 Basel, Switzerland

[12]Roche Pharmaceutical Research and Early Development, Roche Innovation Center Munich, Roche Diagnostics GmbH, Nonnenwald 2, 82377 Penzberg, Germany

[13]Roche Pharmaceutical Research and Early Development, Roche Innovation Center Zurich, Wagistrasse 10, 8952 Schlieren, Switzerland

[14]Swiss Institute of Bioinformatics, Zurich, Switzerland

[15]University Hospital Basel and University of Basel, Department of Biomedicine, Hebelstrasse 20, 4031 Basel, Switzerland

[16]University Hospital Basel and University of Basel, Department of Surgery, Brustzentrum, Spitalstrasse 21, 4031 Basel, Switzerland

[17]University Hospital Basel, Brustzentrum & Tumorzentrum, Petersgraben 4, 4031 Basel, Switzerland

[18]University Hospital Basel, Brustzentrum, Spitalstrasse 21, 4031 Basel, Switzerland

[19]University Hospital Basel, Centre for Neuroendocrine & Endocrine Tumours, Spitalstrasse 21/Petersgraben 4, 4031 Basel, Switzerland

[20]University Hospital Basel, Department of Information- and Communication Technology, Spitalstrasse 26, 4031 Basel, Switzerland

[21]University Hospital Basel, Gynecological Cancer Center, Spitalstrasse 21, 4031 Basel, Switzerland

[22]University Hospital Basel, Institute of Medical Genetics and Pathology, Schönbeinstrasse 40, 4031 Basel, Switzerland

[23]University Hospital Basel, Spitalstrasse 21/Petersgraben 4, 4031 Basel, Switzerland

[24]University Hospital Zurich, Biomedical Informatics, Schmelzbergstrasse 26, 8006 Zurich, Switzerland

[25]University Hospital Zurich, Clinical Trials Center, Rämistrasse 100, 8091 Zurich, Switzerland

[26]University Hospital Zurich, Department of Dermatology, Gloriastrasse 31, 8091 Zurich, Switzerland

[27]University Hospital Zurich, Department of Pathology and Molecular Pathology, Schmelzbergstrasse 12, 8091 Zurich, Switzerland

[28]University Hospital Zurich, Rämistrasse 100, 8091 Zurich, Switzerland

[29]University Hospital and University of Zurich, Department of Neurology, Frauenklinikstrasse 26, 8091 Zurich, Switzerland

[30]University of Bern, Department of BioMedical Research, Murtenstrasse 35, 3008 Bern, Switzerland

[31]University of Zurich, Department of Quantitative Biomedicine, Winterthurerstrasse 190, 8057 Zurich, Switzerland

[32]University of Zurich, Institute of Molecular Life Sciences, Winterthurerstrasse 190, 8057 Zurich, Switzerland

[33]University of Zurich, Services and Support for Science IT, Winterthurerstrasse 190, 8057 Zurich, Switzerland

[34]University of Zurich, VP Medicine, Künstlergasse 15, 8001 Zurich, Switzerland

References

1. Simoni Y, Chng MHY, Li S, *et al.*: **Mass cytometry: a powerful tool for dissecting the immune landscape.** *Curr Opin Immunol.* 2018; **51**: 187–196.
PubMed Abstract | Publisher Full Text

2. Spitzer MH, Nolan GP: **Mass Cytometry: Single Cells, Many Features.** *Cell.* 2016; **165**(4): 780–791.
PubMed Abstract | Publisher Full Text | Free Full Text

3. Behbehani GK: **Applications of Mass Cytometry in Clinical Medicine: The Promise and Perils of Clinical CyTOF.** *Clin Lab Med.* 2017; **37**(4): 945–964.
PubMed Abstract | Publisher Full Text

4. Schulz AR, Baumgart S, Schulze J, *et al.*: **Stabilizing Antibody Cocktails for Mass Cytometry.** *Cytometry A.* 2019; **95**(8): 910–916.
PubMed Abstract | Publisher Full Text

5. Hartmann FJ, Babdor J, Gherardini PF, *et al.*: **Comprehensive Immune Monitoring of Clinical Trials to Advance Human Immunotherapy.** *Cell Rep.* 2019; **28**(3): 819–831.e4.
PubMed Abstract | Publisher Full Text | Free Full Text

6. Palit S, Heuser C, de Almeida GP, *et al.*: **Meeting the Challenges of High-Dimensional Single-Cell Data Analysis in Immunology.** *Front Immunol.* 2019; **10**: 1515.
PubMed Abstract | Publisher Full Text | Free Full Text

7. Olsen LR, Leipold MD, Pedersen CB, *et al.*: **The anatomy of single cell mass cytometry data.** *Cytometry A.* 2019; **95**(2): 156–172.
PubMed Abstract | Publisher Full Text

8. Finck R, Simonds EF, Jager A, *et al.*: **Normalization of mass cytometry data with bead standards.** *Cytometry A.* 2013; **83**(5): 483–494.
PubMed Abstract | Publisher Full Text | Free Full Text

9. Chevrier S, Crowell HL, Zanotelli VRT, *et al.*: **Compensation of Signal Spillover in Suspension and Imaging Mass Cytometry.** *Cell Syst.* 2018; **6**(5): 612–620.e5.
PubMed Abstract | Publisher Full Text | Free Full Text

10. Zunder ER, Finck R, Behbehani GK, *et al.*: **Palladium-based mass tag cell barcoding with a doublet-filtering scheme and single-cell deconvolution algorithm.** *Nat Protoc.* 2015; **10**(2): 316–333.
PubMed Abstract | Publisher Full Text | Free Full Text

11. Schuyler RP, Jackson C, Garcia-Perez JE, *et al.*: **Minimizing Batch Effects in Mass Cytometry Data.** *Front Immunol.* 2019; **10**: 2367.
PubMed Abstract | Publisher Full Text | Free Full Text

12. Van Gassen S, Gaudilliere B, Angst MS, *et al.*: **CytoNorm: A Normalization Algorithm for Cytometry Data.** *Cytometry A.* 2020; **97**(3): 268–278.
PubMed Abstract | Publisher Full Text | Free Full Text

13. Kotecha N, Krutzik PO, Irish JM: **Web-based analysis and publication of flow cytometry experiments.** *Curr Protoc Cytom.* 2010; **53**: 10–17.
PubMed Abstract | Publisher Full Text | Free Full Text

14. Nowicka M, Krieg C, Crowell HL, *et al.*: **CyTOF workflow: differential discovery in high-throughput high-dimensional cytometry datasets [version 3; peer review: 2 approved].** *F1000Res.* 2019; **6**: 748.
PubMed Abstract | Publisher Full Text | Free Full Text

15. Irmisch A, Bonilla X, Chevrier S, *et al.*: **The Tumor Profiler Study: Integrated, multi-omic, functional tumor profiling for clinical decision support.** *Cancer Cell.* 2020; **39**(3): 288–293.
PubMed Abstract | Publisher Full Text

16. Chevrier S, Zurbuchen Y, Cervia C, *et al.*: **A distinct innate immune signature marks progression from mild to severe COVID-19.** *bioRxiv.* 2020: 2020.08.04.236315.
Publisher Full Text

17. Chevrier S, Levine JH, Zanotelli VRT, *et al.*: **An Immune Atlas of Clear Cell Renal Cell Carcinoma.** *Cell.* 2017; **169**(4): 736–749.e18.
PubMed Abstract | Publisher Full Text | Free Full Text

18. Lun A, Risso D, Korthauer K: **SingleCellExperiment: S4 classes for single cell data.** *R package version, 1.14.1.* 2021.
Reference Source

19. Finak G, Frelinger J, Jiang W, *et al.*: **OpenCyto: an open source infrastructure for scalable, robust, reproducible, and automated, end-to-end flow cytometry data analysis.** *PLoS Comput Biol.* 2014; **10**(8): e1003806.
PubMed Abstract | Publisher Full Text | Free Full Text

20. Finak G, Jiang M: **FlowWorkspace: Infrastructure for representing and interacting with gated and ungated cytometry data sets.** *R package version.* 2018.
Publisher Full Text

21. Wickham H: **ggplot2: Elegant Graphics for Data Analysis**. Springer, 2016.
Reference Source

22. Van P, Jiang W, Gottardo R, *et al.*: **ggCyto: next generation open-source visualization software for cytometry.** *Bioinformatics.* 2018; **34**(22): 3951–3953.
PubMed Abstract | Publisher Full Text | Free Full Text

23. Hahne F, LeMeur N, Brinkman RR, *et al.*: **flowCore: a Bioconductor package for high throughput flow cytometry.** *BMC Bioinformatics.* 2009; **10**(1): 106.
PubMed Abstract | Publisher Full Text | Free Full Text

24. Wickham H: **reshape2: Flexibly reshape data: a reboot of the reshape package**. *R package*, 2012.
Reference Source

25. Wickham H, Francois R, Henry L, *et al.*: **dplyr: A grammar of data manipulation.** *R package.* 2015.
Reference Source

26. Bodenmiller B, Zunder ER, Finck R, *et al.*: **Multiplexed mass cytometry profiling of cellular states perturbed by small-molecule regulators.** *Nat Biotechnol.* 2012; **30**(9): 858–867.
PubMed Abstract | Publisher Full Text | Free Full Text

27. McCarthy DJ, Campbell KR, Lun ATL, *et al.*: **Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R.** *Bioinformatics.* 2017; **33**(8): 1179–1186.
PubMed Abstract | Publisher Full Text | Free Full Text

28. Weber LM, Nowicka M, Soneson C, *et al.*: **diffcyt: Differential discovery in high-dimensional cytometry via high-resolution clustering.** *Commun Biol.* 2019; **2**: 183.
PubMed Abstract | Publisher Full Text | Free Full Text

29. Fletez-Brant K, Špidlen J, Brinkman RR, *et al.*: **flowClean: Automated identification and removal of fluorescence anomalies in flow cytometry data.** *Cytometry.* 2016; **89**(5): 461–471.
PubMed Abstract | Publisher Full Text | Free Full Text

30. Trussart M, Teh CE, Tan T, *et al.*: **Removing unwanted variation with CytofRUV to integrate multiple CyTOF datasets.** *eLife.* 2020; **9**: e59630.
PubMed Abstract | Publisher Full Text | Free Full Text

31. Van Gassen S, Callebaut B, Van Helden MJ, *et al.*: **FlowSOM: Using self-organizing maps for visualization and interpretation of cytometry data.** *Cytometry A.* 2015; **87**(7): 636–645.
PubMed Abstract | Publisher Full Text

32. Finney DJ: **Probit analysis.** *J Pharm Sci.* 1971; **60**(9): 1432.

33. Ritz C, Baty F, Streibig JC, *et al.*: **Dose-Response Analysis Using R.** *PLoS One.* 2015; **10**(12): e0146021.
PubMed Abstract | Publisher Full Text | Free Full Text

34. Lawson CL, Hanson RJ: **Solving least squares problems prentice-hall**. Prentice Hall, Englewood Cliffs, NJ. 1974.
Reference Source

35. Lawson CL, Hanson RJ: **Solving Least Squares Problems**. SIAM, Philadelphia, PA. 1995.
Reference Source

36. R Core Team: **R: A Language and Environment for Statistical Computing**. R Foundation for Statistical Computing, Vienna, Austria, 2019.
Reference Source

37. Huber W, Carey VJ, Gentleman R, *et al.*: **Orchestrating high-throughput genomic analysis with Bioconductor.** *Nat Methods.* 2015; **12**(2): 115–121.
PubMed Abstract | Publisher Full Text | Free Full Text

Open Peer Review

Current Peer Review Status: ✔ ✔

Version 2

Reviewer Report 25 August 2022

https://doi.org/10.5256/f1000research.135465.r146797

✔ **Marie Trussart**
Bioinformatics Division, Walter and Eliza Hall Institute of Medical Research, Parkville, Vic, Australia

Thanks for the updated version and for assessing the suggestions and issues mentioned in the first report.
I think this revised version is easier to follow and clearer with the updates provided.
One last comment would be on the batch alignment section, the Figure 17 selected is not showing the 7 different markers distribution to compare the expression distributions before and after batch correction and I think it would be useful in the assessment of this correction.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: I work in Bioinformatics and especially in the normalization and batch correction of CyTOF datasets.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Reviewer Report 01 December 2020

https://doi.org/10.5256/f1000research.28774.r73588

✔ **Felix Hartmann**

School of Medicine, Stanford University, Palo Alto, CA, USA

Sean C. Bendall

Immunology Graduate Program, Stanford University, Stanford, CA, USA

In their manuscript, Crowell & Chevrier et al. present a novel workflow to preprocess mass cytometry (CyTOF) datasets in R. The presented pipeline is a useful update on earlier publications and packages, including the authors' CATALYST package which is clearly stated. Overall, I find this to be a valuable tool that brings together different functionalities into a unified workflow that enables reproducible and comprehensive preprocessing of this data type. The different steps and approaches are well described and illustrated. Especially, the inclusion of a functionality to perform live cell gating without having to switch platforms is much appreciated, although its current implementation could be improved:

1. Gating on cells: Cells are first identified using an elliptical gate to exclude the two lowest density percentiles. Firstly, this plot relies on two DNA channels (whose information is likely redundant) and wasn't directly applicable to alternative DNA stains (e.g. rhodium). Furthermore, I am wondering whether this approach might exclude for example a fraction of cycling cells or preferentially exclude cell types or states with increased chromatin accessibility and therefore higher DNA signal?

2. Gating on live cells: The approach suggested by the authors worked well on my test data, however, it takes a while to manually adjust values for every file to fit the gates closely to the data. While I see the value of automating this step, I also think that some manual gating could simplify the process and further increase downstream data quality. Potentially, the authors could adopt an approach like the gate_draw function from the CytoRSuite library.

3. Compensation: The workflow includes compensation as a preprocessing step which the authors have shown in separate publications to improve data quality, but which is currently not routinely performed by many researchers working using mass cytometry. I, therefore, assume that most users of this pipeline would be relying on published spillover matrices that reflect estimates of isotope purity and oxidation. While I agree with the usefulness of this function, I believe that adding additional quality control functions could improve acceptance of and trust in this approach. For example, in flow cytometry, overcompensation is often easily spotted by the occurrence of overly negative values, however, using their NNLS approach this is not readily apparent in compensated mass cytometry data. It would be very helpful to have a quality metric that would alert users to such potential issues introduced by the compensation step.

In addition, testing this pipeline on some in-house generated data, a few minor issues occurred which should be addressed:

1. While this might only be needed in rare cases, a function to rename channels and potentially match these names across multiple fcs files could enhance the adaptability of this package. In my test case, conflicting channel names prevented the import of the files into the workflow. In other cases, it might help to match channel names between batches. The authors could look to the premessa package for inspiration.

2. The authors have incorporated various options for DNA channels which is much appreciated. My test data had been stained with a rhodium intercalator. Specifying this

worked well, only the res$scatter function seems to ignore this choice and instead seems to default to iridium DNA intercalators.

3. Sample specific debarcoding is appreciated. Figure 6 and the plotYields function return a debarcoding percentage. I believe this percentage refers to percent of initial assignments, but it is not specifically stated. It might be helpful to get a feeling of the percentage of cells (out of total cells) that are assigned after refining the initial assignments.

Is the rationale for developing the new method (or application) clearly explained?
Yes

Is the description of the method technically sound?
Yes

Are sufficient details provided to allow replication of the method development and its use by others?
Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Single cell proteomics, Mass Cytometry, Immunology, Stem Cell biology

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 23 Jun 2022

Helena Crowell, University of Zurich, Zurich, Switzerland

1. *Gating on cells: Cells are first identified using an elliptical gate to exclude the two lowest density percentiles. Firstly, this plot relies on two DNA channels (whose information is likely redundant) and wasn't directly applicable to alternative DNA stains (e.g. rhodium). Furthermore, I am wondering whether this approach might exclude for example a fraction of cycling cells or preferentially exclude cell types or states with increased chromatin accessibility and therefore higher DNA signal?*

   The first gating step is indeed performed on two DNA channels which contains redundant information. However, this approach is commonly used in the mass cytometry field to exclude debris and cell doublets. By modifying the quantile and the

target value defining the center of the ellipse, the user can control how many cells are excluded from the gate and ensure that most cycling cells are kept in the analysis.

To gate on alternative DNA stains, a different pair of channels could be assigned to the "dna" variable in the corresponding code chunk. In the case of a single DNA channel, a one-dimensional gating (i.e., thresholding) could be applied (as opposed to the currently used elliptical gate). We have added a comment mentioning this to the text under "Gating on cells".

2. *Gating on live cells: The approach suggested by the authors worked well on my test data, however, it takes a while to manually adjust values for every file to fit the gates closely to the data. While I see the value of automating this step, I also think that some manual gating could simplify the process and further increase downstream data quality. Potentially, the authors could adopt an approach like the gate_draw function from the CytoRSuite library.*

   Indeed, the approach depicted in this paper works well in cases where a limited number of samples are included in a run and when the live/dead cell profile is well defined and consistent between samples. The process can indeed be tedious when hundreds of samples are included in a run or when the live/dead cell profile is more complex. In the latter case, including a function similar to gate_draw function from CytoRSuite could be helpful. However, we here aimed at proposing an automated pipeline; manual gating would defeat this purpose.

   As a side note: We have attempted applying CytoRSuite, however, encountered several confusing issues that we've been unable to resolve: The CytoRSuite site (https://dillonhammill.github.io/CytoRSuite) lists a GH repository that no-longer exists; we could find an installable version at https://github.com/gfinak/cytoRSuite (is this the same?) but 'drawGate()' gave an error that we have not been able to resolve; meanwhile, any of CytoRSuite, cytoRSuite and cytoSuite (from which the latter has been forked) have not been changed in 4 years. Taken together, this gave us the feeling that the tool is no longer maintained and likely to be inapplicable with current versions of R and Bioconductor.

   Of course, there may be other tools available at this point for manual gating, and we leave it to the user to incorporate these into their workflow should that be of interest. A possible strategy then might be to i) perform manual gating and export the resulting gates into a table (gating scheme); ii) apply that scheme in an automated fashion (e.g., using the code we presented); and, iii) do manual adjustments to refine gates according to the current experiment.

3. *Compensation: The workflow includes compensation as a preprocessing step which the authors have shown in separate publications to improve data quality, but which is currently not routinely performed by many researchers working using mass cytometry. I, therefore, assume that most users of this pipeline would be relying on published spillover matrices that reflect estimates of isotope purity and oxidation. While I agree with the usefulness of this function, I believe that adding additional quality control functions could*

improve acceptance of and trust in this approach. For example, in flow cytometry, overcompensation is often easily spotted by the occurrence of overly negative values, however, using their NNLS approach this is not readily apparent in compensated mass cytometry data. It would be very helpful to have a quality metric that would alert users to such potential issues introduced by the compensation step.

These are all very good points and legitimate concerns. As indicated in the original paper, the spillover matrix used to compensate mass cytometry data should be calculated based on the antibodies included in the panel. We should stress here that, based on the single-stained bead acquisition approach presented in the original paper, the experimental procedure required to generate a compensation matrix is fairly straightforward and can be achieved rapidly. Using a previously published spillover matrix is a risky strategy, which can indeed lead to inaccurate compensation. The user should instead first run the compensation in classic mode and perform a visual inspection to ensure no overcompensation can be detected before using the NNLS method. This is a valuable option to avoid this specific type of artefact. Automating this step is a good suggestion, but is out of the scope of this publication and comes with some disadvantages. The risk we see is that this process could be imperfect and potentially misleading for the user. Indeed, such an approach would only identify overcompensation in channels where a single positive population is present but not in the case of a double positive population. In other words, it will highly depend on the user's data type. Furthermore, it would not identify under-compensated signals. As a consequence, providing an approach to alert users of potential issues would likely be imperfect and could give a wrong impression that the data are correctly compensated if no alert is raised, which is not necessarily the case. Moreover, to the best of our knowledge, such an approach also doesn't exist in fluorescent flow cytometry, most likely due to the fact that ensuring accurate compensation on a fully stained data set is a challenging task. We should also mention that the spillover coefficients in mass cytometry rarely exceed 4% and therefore the consequences of a slight over or under-compensation are less important in mass cytometry than in flow cytometry.

Minor comments:

1. *While this might only be needed in rare cases, a function to rename channels and potentially match these names across multiple fcs files could enhance the adaptability of this package. In my test case, conflicting channel names prevented the import of the files into the workflow. In other cases, it might help to match channel names between batches. The authors could look to the premessa package for inspiration.*

   We very much appreciate this comment as we have encountered various discrepancies between panels, especially in long-term projects. To date, we have used a custom R script to i) read in files separately; ii) fix panels according to a reference file (i.e., removing/adding additional/missing channels); and, iii) write out a new set of FCS files with concordant panels.

   However, this solution is suboptimal as it leads to a duplication of files (or, in case the original files were overwritten, the process would no longer be reproducible). Similarly, a GUI solution (as 'premessa') would defeat the purpose of providing an

automated, reproducible preprocessing solution. Thus, taken together, we propose (and have now implemented) the following strategy:

> 'prepData' now exposes additional arguments to be passed to 'flowCore::read.FCS' via '...'
> 'read.FCS' provides an argument 'channel_alias': "an optional 'data.frame' used to provide the alias of the channels to standardize and solve the discrepancy across FCS files. [...]"
> independent of whether or not this option is used, 'prepData' will check whether panels (FCS channel names) match between files:
- in case of any discrepancy, the newly added 'fix_chs' argument will be used to determine how to resolve discrepancies
- "all" will keep all channels (i.e., the union across files); any missing channels will be added to the respective samples, and a channels x samples matrix is stored in the object to track which channels were present in which samples originally
- "common" will keep shared channels (i.e., the intersection across files); any other channels will be dropped from the respective files
- 'prepData' will, in any case, return a 'SingleCellExperiment', i.e., no altered FCS files or 'flowFrame's will be written out / returned

2. *The authors have incorporated various options for DNA channels which is much appreciated. My test data had been stained with a rhodium intercalator. Specifying this worked well, only the res$scatter function seems to ignore this choice and instead seems to default to iridium DNA intercalators.*

   We thank the reviewer for noticing this. Indeed, while the workflow allows for specification of the DNA channels used (via variable 'dna'), these were fixed internally in CATALYST's 'normCytof()' function. We have added an additional argument to allow passing custom DNA channel masses (default 'dna = c(191, 193)' for Ir191/3; for Rh103, the argument would be 'dna = 103' instead); the output scatter plot of DNA vs. bead intensities ('res$scatter') is now generated based on the first matching DNA channel (see updated '?normCytof' documentation).

3. *Sample specific debarcoding is appreciated. Figure 6 and the plotYields function return a debarcoding percentage. I believe this percentage refers to percent of initial assignments, but it is not specifically stated. It might be helpful to get a feeling of the percentage of cells (out of total cells) that are assigned after refining the initial assignments.*

   That is correct: As in the original Finck et al. outputs (a MATLAB application), yields (left-hand y-axis) correspond to the proportion of cells that would be retained upon applying a given cutoff (x-axis). In Figure 8, we compare the absolute barcode population sizes before vs. after debarcoding. Analogously, it would be straightforward for users to generate such a barplot from cell counts obtained when applying various thresholding schemes (e.g., no filtering compared to global vs. sample-specific separation cutoffs).

Competing Interests: No competing interests were disclosed.

Reviewer Report 05 November 2020

https://doi.org/10.5256/f1000research.28774.r73591

?

Marie Trussart
Bioinformatics Division, Walter and Eliza Hall Institute of Medical Research, Parkville, Vic, Australia

The manuscript is presenting an updated version of the CATALYST package for preprocessing Cytof data. It is well detailed with several examples and has been updated based on the Bioconductor SingleCellExperiment class. Every step of preprocessing is clearly stated and illustrated to guide the user on the different steps to process their data. Also, new quality checks are being reviewed to explore the quality of the data.
I provided below some feedback to make the manuscript clearer and some suggestions to address some issues I encountered:

1. It will be useful to define clearly what are the differences between successive acquisitions, single CyTOF run and batch.

2. The different samples and runs listed through the different examples could be better presented with a table containing all runs and samples. In the data description, it explains that "The dataset used in this study was obtained from a single CyTOF run containing nine references, three blood samples and three tumor samples barcoded with a 20-well barcoding plate". However in the quality checks section; additional data is being analyzed which makes it confusing, coming from additional runs, sometimes from 7 runs or other times from 8 runs.

3. Batch alignment:
   Could you provide an additional plot showing the effect of applying this correction factor?
   How are you assessing the performance of your batch alignment method?

4. argument norm_to in the normCytof() function: give explanations on how it being computed when giving reference beads, especially how does it compute the new baseline, does it takes into account both the beads from the reference and current by averaging both?
   Can it be used to normalize data from different batches? If so how does it deal with distinguishing times and ordering the beads and time which would be similar in separated batches?

5. Figure 4: Could you please give more explanations on how to assess run sensitivity and how does the user decide what is acceptable and what is not?

Also, you need to load the library(reshape2) to run this part.

6. The wrap_plots function is missing here.

7. I got an error when running the QC on reference cell counts. "Error: Can't combine `1$CellLine_R1` and `2$CellLine_R1` ."

Minor comments:

1. When running the code using the data provided, the directory name should be modified to "CyTOF_acquisition_1-3.FCS/" instead of data
fcs <- list.files("CyTOF_acquisition_1-3.FCS/", "acquisition", full.names = TRUE)
Also, it should be specified that the directory name containing all the data should be called "data" and it refers to the directory name, or an alternative is to have the local directory "." instead of data like in here:
# specify path to reference beads
ref_beads <- file.path(".", "normalization_beads.fcs")

2. Introduction:
"an important step is to correct for batch effects, which can be achieved by including a shared control sample in each independent batch[11,12] " Add CytofRUV reference mentioned in the discussion.

3. "In our example, barcode identifiers include each sample's type (CellLine, PBMC or Tumor), group (R for reference or S for sample of interest), and replicate number; and follow a consistent naming scheme: We can easily extract these components and store them in the SCE's cell metadata (colData)". The example selected is not the best one, as it not showing any differences between the 6 first row.

4. There is a typo in the legend of some figures like figure 17: "previously" acquired runs.

Is the rationale for developing the new method (or application) clearly explained?

Yes

Is the description of the method technically sound?

Yes

Are sufficient details provided to allow replication of the method development and its use by others?

Partly

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?

Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: I work in Bioinformatics and especially in the normalization and batch correction of CyTOF datasets.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 23 Jun 2022

Helena Crowell, University of Zurich, Zurich, Switzerland

1. *It will be useful to define clearly what are the differences between successive acquisitions, single CyTOF run and batch.*

   Indeed, the meaning behind the concepts of successive acquisitions, single CyTOF run and batch was not fully clear and these terms were not used in a consistent way. A “CyTOF run” corresponds to an independent experiment where samples are stained and acquired simultaneously on the CyTOF. We replaced the term run with experiment to clarify the meaning. Each CyTOF experiment corresponds to one “batch” and this term is used to refer to the batch correction which is performed on the different CyTOF experiments. The data from a single CyTOF experiment are usually acquired over multiple “successive acquisitions”, each leading to the generation of a single FCS file. We also made the use of these terms consistent throughout the paper.

2. *The different samples and runs listed through the different examples could be better presented with a table containing all runs and samples. In the data description, it explains that "The dataset used in this study was obtained from a single CyTOF run containing nine references, three blood samples and three tumor samples barcoded with a 20-well barcoding plate". However in the quality checks section; additional data is being analyzed which makes it confusing, coming from additional runs, sometimes from 7 runs or other times from 8 runs.*

   The pipeline described in this paper was designed to preprocess CyTOF data acquired over a long period of time with a focus on ensuring data consistency over time. The aim of the workflow is to guide the readers through the preprocessing steps required to convert FCS files obtained in a given CyTOF experiment to a format suitable for downstream analysis, while presenting key quality checks to ensure the reliability of the data generated in the experiment of interest. Therefore, the whole analysis is based on a dataset obtained from a single CyTOF experiment, which is benchmarked against data acquired during a preparatory phase. For consistency reasons, we included now systematically the data from seven previous CyTOF experiments to benchmark the data of the CyTOF experiment preprocessed in this paper.

3. *Batch alignment: Could you provide an additional plot showing the effect of applying this correction factor? How are you assessing the performance of your batch alignment method?*

   The batch alignment presented in this paper is based on a linear scaling based on a percentile, using references as anchoring points, similar to a previously published method (Schuyler et al, 2019). To assess the performance of our batch alignment method, we have now included a figure to compare the expression distributions before and after batch correction (including their 98th percentiles and those of the references). As intended, 98th percentiles align with the references' upon correction, while expression distributions remain virtually unchanged.

4. *argument norm_to in the normCytof() function: give explanations on how it being computed when giving reference beads, especially how does it compute the new baseline, does it take into account both the beads from the reference and current by averaging both?*

   Normalization using reference beads follows the methodology originally introduced in Finck et al. (2013): The baseline is computed as the mean intensity of the reference beads only, not including the current experiment. Would the average be taken over both, intensities would not be aligned between current and reference experiment. While the statement "[...] We provide the path to a set of reference beads (argument `norm_to`) that are used to compute baseline intensities for normalization" explains this only briefly, we believe that the method is well established and readers should refer to the original publication for more detail.

   Can it be used to normalize data from different batches? If so how does it deal with distinguishing times and ordering the beads and time which would be similar in separate batches?

   Yes, certainly. The normalization aims at correcting the signal time-drift due to progressive loss of sensitivity during acquisition. This is a technical effect that is independent of batch effects, and should be accounted for regardless of whether or not batch effects are present: these should be corrected for downstream analysis.

   Events from different FCS files (independent of whether these are different acquisitions of the same experiment or batches) are concatenated. How event times are dealt with depends on prepData()'s input arguments. When by_time = TRUE, files are ordered according to their acquisition time (stored under each flowFrame's $BTIM description field). Otherwise, they are kept in the order provided by the input metadata table (argument md).

5. *Figure 4: Could you please give more explanations on how to assess run sensitivity and how does the user decide what is acceptable and what is not?*

   Instrument sensitivity is an important parameter that should be closely monitored. This parameter is assessed during the tuning but those data cannot be easily

exported and compared between experiments. The aim was to take advantage of the beads, which are run together with the samples to report on instrument sensitivity. Figure 3 provides key information regarding how the sensitivity evolves during the run, while the point of Figure 4 is to show how the average sensitivity evolves from one experiment to another. Instrument sensitivity varies from machine to machine and deciding what is acceptable will depend on the requirements of the users. The point of this plot was to offer an option for the user to easily identify in case the sensitivity is getting low compared to previous experiments, and to make a link between the quality of the data generated in a specific experiment with the sensitivity of the instrument.

Also, you need to load the library(reshape2) to run this part.

Yes, thank you for catching this; we've added reshape2 to the list of dependencies, and it is now loaded along the other required libraries.

6. *The wrap_plots function is missing here.*

   Yes, thank you for catching this; we've added patchwork to the list of dependencies, and it is now loaded along the other required libraries.

7. *I got an error when running the QC on reference cell counts. "Error: Can't combine `1$CellLine_R1` and `2$CellLine_R1`."*

   True, thank you; I could reproduce this with the current R and package versions. It has been fixed by converting the 'run' object of class 'table' to call 'integer' using c().

Minor comments:

1. *When running the code using the data provided, the directory name should be modified to "CyTOF_acquisition_1-3.FCS/" instead of data:*
   fcs <- list.files("CyTOF_acquisition_1-3.FCS/", "acquisition", full.names = TRUE)

   We are not sure we understand this comment. 'list.files(path, pattern, ...)' expects the first argument to be a directory (where the FCS files are located), not the file names themselves ("xxx.FCS").

   Also, it should be specified that the directory name containing all the data should be called "data" and it refers to the directory name, or an alternative is to have the local directory "." instead of data like in here:
   # specify path to reference beads
   ref_beads <- file.path(".", "normalization_beads.fcs")

   Thank you, yes, we forgot to mention that in the presented code all data used throughout the workflow is expected to sit inside a "data" subdirectory relative to where the .Rmd file is being run. We have now added a note explaining this in the 2nd paragraph under "Results".

2. *Introduction: "an important step is to correct for batch effects, which can be achieved by*

including a shared control sample in each independent batch" Add CytofRUV reference mentioned in the discussion.

We updated the reference to CytofRUV to the new version of the manuscript published in eLife and added it to the introduction.

3. *"In our example, barcode identifiers include each sample's type (CellLine, PBMC or Tumor), group (R for reference or S for sample of interest), and replicate number; and follow a consistent naming scheme: We can easily extract these components and store them in the SCE's cell metadata (colData)". The example selected is not the best one, as it not showing any differences between the 6 first row.*

   True. We have modified the example to sample 10 unique 'sample' entries (= type_group) for which to display the 'colData'.

4. *There is a typo in the legend of some figures like figure 17: "previously" acquired runs.*

   Thanks for pointing out this typo, which was corrected in the corresponding figures.

Competing Interests: No competing interests were disclosed.