Data Analysis Supplement

February 20, 2019

Contents

Introduction

This document was prepared in support of our paper:

Single cell transcriptomics identifies immunologic priming related to extra corporeal life support survival.

Eric J. Kort MD, Matthew Weiland MS, Emily Eugster BS, Hsiao-yun Milliron PhD, Edgars Grins MD, Marzia Leacche MD, Stephen J Fitch MD, Theodore J Boeve MD, Greg Marco MD, Michael Dickinson MD, Penny Wilton MD, Stefan Jovinge PhD

The following sections document how the data for this paper were processed and how the figures were generated. Those who wish to do so can recreate the figures from the paper from data (that will soon be) posted on GEO, and the code below. For efficient compiling of this document, the "eval=FALSE" option was set globally. Readers desiring to repeat the analysis can either run each code chunk below manually, or change the setting line at the top of the Rmd file from:

```
knitr::opts_chunk$set(echo = TRUE, eval=FALSE)
```

to

```
knitr::opts_chunk$set(echo = TRUE, eval=TRUE)
```

And then knit the entire document, a process which may take several hours, and may also fail if insufficient RAM is available.

The result of running this code is that all processed data and generated figures will be produced and saved to the Final_Data directory. The figures should match exactly what is in the publication, except that in figure 4 panel B the GO ontology term annotations were added manually (based on the GO ontology analysis that can be found below).

Since the processed data elements created below are save as RDS files in the chunks that create them, you can execute just some of the chunks and then pick up where you left off later.

Pre-requisites

To run the following analysis, the file `PBMC_merged_filtered.rds` must be obtained from the GEO series related to this study (GSE127221). The second panel of figure 4b requires the list of genes from Supplmental Table S4 (table_s4.txt). The following code assumes these files are within a subdirectory named "Final_Data"

underneath the working directory. Note that cloning the github repo will create the necessary directory structure, including the file table_s4.txt. However, `PBMC_merged_filtered_recoded.rds` is too large for the gitub repository, and thus must be obtained from GEO and saved in the Final_Data directory.

Running the following analysis requires a computer with R version > 3.5.0 and 128GB of RAM. If not already present, required R packages can be installed as follows:

```
setRepositories(ind=c(1,2))
install.packages(c("devtools",
                   "irlba",
                   "dplyr",
                   "foreach",
                   "Matrix",
                   "reshape2",
                   "ggplot2",
                   "GGally",
                   "ggpubr",
                   "ggplotify",
                   "survminer",
                   "cowplot",
                   "survival",
                   "DOSE",
                   "clusterProfiler",
                   "pheatmap",
                   "org.Hs.eg.db",
                   "stringr"))

devtools::install_github("immunogenomics/harmony")
devtools::install_github("jlmelville/uwot")
devtools::install_github("kassambara/rstatix")
```

Data Pre Processing

The data file `PBMC_merged_filtered_recoded.rds` contaings sequencing data that was aligned and converted to UMI counts with the inDrop pipeline. Barcodes were filtered to retain only cells with at least 500 unique counts.

As shown below, we then further filtered this dataset to select cells with between 750 and 7500 unique counts in an effort to select true, single cells. We then normalized by library size, scaled by a constant (10000), and log transformed, as follows:

```
library(Matrix)
library(irlba)

dat <- readRDS("GEO/PBMC_merged_filtered_recoded.rds")

# filtering, somewhat subjectively, for true, single cells
cellCounts <- rowSums(dat)
dat <- dat[which(cellCounts < 7500 & cellCounts > 750), ]
geneCounts <- colSums(dat)
dat <- dat[ , which(geneCounts >= 10)]

# we want cells as row and genes as columns for ALRA
dat.m <- as(dat, "matrix")
rm(dat)
```

```
# normalize to library size and log transform
sizes <- rowSums(dat.m)
dat.n <- sweep(dat.m, MARGIN = 1, sizes, "/")
dat.n <- dat.n * 10E3
dat.n <- log(dat.n + 1)
saveRDS(dat.n, file="Final_Data/PBMC_merged_filtered_normalized.rds")

# pc loadings prior to imputation for comparison later
dat.raw.pc <- prcomp_irlba(dat.n, n=100)
rownames(dat.raw.pc$x) <- rownames(dat.imp)
saveRDS(dat.raw.pc, file="Final_Data/PBMC_merged_filtered_normalized_pc.rds")

# alra imputation
dat.n <- readRDS("Final_Data/PBMC_merged_filtered_normalized.rds")
source("https://raw.githubusercontent.com/KlugerLab/ALRA/master/alra.R")

# ALRA uses the random SVD for performance reasons.  If we don't set a seed,
# the results of the imputation will be very nearly but not exactly the same
# between runs.
set.seed(1010)
dat.imp <- alra(dat.n, k=50, q = 40)[[3]]
rownames(dat.imp) <- rownames(dat.n)
saveRDS(dat.imp, file="Final_Data/PBMC_merged_filtered_alra.rds")
```

For batch effect removal and UMAP visualization, we first need the principle component loadings for the imputed dataset. Calculating just a partial PC matrix (the first 100 PCs, which is plenty as we shall see) using the `irlba` package makes this task much more tractable in terms of both memory and time requirements.

```
library(harmony)
dat.imp <- readRDS("Final_Data/PBMC_merged_filtered_alra.rds")
dat.pc <- prcomp_irlba(dat.imp, n=100)
rownames(dat.pc$x) <- rownames(dat.imp)
saveRDS(dat.pc, file="Final_Data/PBMC_merged_filtered_alra_pc.rds")
```

Next we regress away donor (batch) effect using the Harmony algorithm.

```
library(harmony)

dat.pc <- readRDS("Final_Data/PBMC_merged_filtered_alra_pc.rds")

# patient id is the library id and each patient was a separate sequencing library
id <- gsub("\\..*", "", rownames(dat.pc$x))

dat.pc.h <- HarmonyMatrix(dat.pc$x, data.frame(id=id), "id", theta = 4)
saveRDS(dat.pc.h, file="Final_Data/PBMC_merged_filtered_alra_pc_harmony.rds")
```

Dimensionality reduction with UMAP

Now we have three sets of PC loadings. One from the raw (library normalized, log transformed) data, one from the data after imputation, and one after removal of batch effects. We can use these PC loadings to further reduce dimensionality with UMAP.

```
library(uwot)
set.seed(1010)
```

```
dat.raw.pc <- readRDS(file="Final_Data/PBMC_merged_filtered_normalized_pc.rds")
umap.r <- umap(dat.raw.pc$x, min_dist = 0.2, n_neighbors = 15, n_threads = 7)
umap.r <- data.frame(id = gsub("\\..*", "", rownames(dat.raw.pc$x)),
                     UMAP1 = umap.r[,1],
                     UMAP2 = umap.r[,2])
saveRDS(umap.r, "Final_Data/PBMC_merged_filtered_normalized_pc_umap.rds")

dat.pc <- readRDS(file="Final_Data/PBMC_merged_filtered_alra_pc.rds")
set.seed(1010)
umap.a <- umap(dat.pc$x, min_dist = 0.2, n_neighbors = 15, n_threads = 7)
umap.a <- data.frame(id = gsub("\\..*", "", rownames(dat.pc$x)),
                     UMAP1 = umap.a[,1],
                     UMAP2 = umap.a[,2])
saveRDS(umap.a, "Final_Data/PBMC_merged_filtered_alra_pc_umap.rds")

set.seed(1010)
dat.pc.h <- readRDS(file="Final_Data/PBMC_merged_filtered_alra_pc_harmony.rds")
umap.h <- umap(dat.pc.h[ , 1:50], min_dist = 0.2, n_neighbors = 15, n_threads = 7)
umap.h <- data.frame(id = gsub("\\..*", "", rownames(dat.pc.h)),
                     UMAP1 = umap.h[,1],
                     UMAP2 = umap.h[,2])
saveRDS(umap.h, "Final_Data/PBMC_merged_filtered_alra_pc_harmony_umap.rds")
```

We can then plot the cells in two dimensions based on the UMAP mappings. We want to color code the cells by either ID, cell type, or outcome, so we need that metadata for each cell. We already have the id (from the rownames of the expression matrix–see above).
Patient outcome (72 hour survival) is available in metadata file also available on GEO (GSE00000000). We just need to match the cell ids in the UMAP data to the patient ids in the metadata object.

Note that we could put all of this data into a `SingleCellExperiment` object or a `Seurat` object. For the operations we are performing, there seemed to be negligible benefit for doing so for the purposes of this analysis, so we prefered to keep the expression matrix and metadata as separate objects and work on them directly. But feel free to select another data structure that best fits your needs.

Cell type assignment

Now we need to define the cell type for each cell based on its gene expression. There are a few ways to do this. Our first thought was to correlate our gene expression values to those for sorted cell populations (Zheng et al.), or cells with known surface marker levels as determined by CITE-Seq. However, the original CITE-Seq showed excellent correspondence between mRNA levels and cell surface protein levels, so we decided to just use mRNA expression as a proxy for cell surface marker protein level. This approach turned out to work very well.

First we define a list structure containig our cell type definitions. The "gate" field indicates whether the corresponding marker should be expressed (`gate = 1`) or not expressed (`gate = 0`). If `gate` is > 1, then `(gate - 1)` is used as the threshold, although this was case only applies to erythrocytes since hemolysis results in some HBA1 RNA is most droplets. A table summarizing the resulting cell definitions is provided in Supplemental Table 1.

```
library(foreach)

cellTypeDefs <- list(
  "B Cells" = list(markers = c("CD19", "CD3E"),
                   gate = c(1, 0)),
```

```
  # CD4 subpopulations can by CD2+/- and FOXP3 +/-, but not double positive
  # Except for CD4 regulatory T which are double positive
  "CD4+ Naive T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "FOXP3", "NCAM1"),
                        gate = c(1, 1, 0, 0, 0, 0)),
  "CD4+ Naive T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "IL2RA", "FOXP3", "NCAM1" ),
                        gate = c(1, 1, 0, 0, 0, 1, 0)),
  "CD4+ Memory T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "B3GAT1", "FOXP3", "NCAM1"),
                        gate = c(1, 1, 0, 1, 0, 0, 0)),
  "CD4+ Memory T" = list(markers = c("CD3E"
                                     , "CD4", "CD8A", "CD2", "B3GAT1", "IL2RA", "FOXP3", "NCAM1"),
                        gate = c(1, 1, 0, 1, 0, 0, 1, 0)),
  "CD4+ Effector T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "B3GAT1", "FOXP3", "NCAM1"),
                        gate = c(1, 1, 0, 1, 1, 0, 0)),
  "CD4+ Effector T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "B3GAT1", "IL2RA", "FOXP3", "NCAM1
                        gate = c(1, 1, 0, 1, 1, 0, 1, 0)),
  "CD4+ Reg T" = list(markers = c("CD3E", "CD4", "CD8A", "IL2RA", "FOXP3", "NCAM1"),
                        gate = c(1, 1, 0, 1, 1, 0)),

  "CD8+ Memory T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "B3GAT1", "NCAM1"),
                        gate = c(1, 0, 1, 1, 0, 0)),
  "CD8+ Naive T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "NCAM1"),
                        gate = c(1, 0, 1, 0, 0)),
  "CD8+ Effector T" = list(markers = c("CD3E", "CD4", "CD8A", "CD2", "B3GAT1", "NCAM1"),
                        gate = c(1, 0, 1, 1, 1, 0)),

  "NKT CD4+" = list(markers = c("CD3E", "CD4", "CD8A", "CD19", "NCAM1"),
                        gate = c(1, 1, 0, 0, 1)),

  "NKT CD8+" = list(markers = c("CD3E", "CD4", "CD8A", "CD19", "NCAM1"),
                        gate = c(1, 0, 1, 0, 1)),

  "NKT CD4- CD8-" = list(markers = c("CD3E", "CD4", "CD8A", "CD19", "NCAM1"),
                        gate = c(1, 0, 0, 0, 1)),

  "NK" = list(markers = c("CD3E", "CD19", "NCAM1"),
                        gate = c(0, 0, 1)),

  "CD16- Monocytes" = list(markers = c("CD14", "FCGR3A"),
                        gate = c(1, 0)),

  "CD16+ Monocytes" = list(markers = c("CD14", "FCGR3A"),
                        gate = c(1, 1)),

  "DC" = list(markers = c("CD3E", "CD14", "CD19", "NCAM1", "HLA-DRA"),
                        gate = c(0, 0, 0, 0, 1)),

  "Erythrocytes" = list(markers = c("HBA1"),
                        gate = c(7))
)


classify <- function(dat, types) {
  class <- rep(NA, nrow(dat))
```

```
  res <- foreach(i = 1:length(types)) %do% {
    m <- types[[i]][["markers"]]
    cl <- foreach(j = 1:length(types[[i]][["markers"]]), .combine = "&") %do% {
      if(types[[i]][["gate"]][j]) {
        return(dat[ , m[j]] > types[[i]][["gate"]][j] - 1)
      } else {
        return(dat[ , m[j]] == 0)
      }
    }
    if(length(which(cl)) > 0) {
      class[which(cl & !is.na(class))] <- "AMBIG"
      class[which(cl & is.na(class))] <- names(types)[i]
    }
  }
  class
}

types <- classify(dat.imp, cellTypeDefs)
types[ which(types == "AMBIG") ] <- NA
types[ which(is.na(types)) ] <- "Unknown"
```

Note that the cell type definitions we have specified successfully assigns $> 81\%$ of cells to a single cell type, with less than 3% of cells assigned to more than one cell type (these abmiguous cell type assignments are removed). Some of the remaining ~17% of cells may be unassigned due to technical drop outs, while the others may be cell types we did not defined such as granulocytes that were not successfully removed by the ficoll gradient isolation of PBMCs, circulating epithelial cells, etc. Some may also represent empty droplets containing just background RNA from lysed cells that made it past our data filtering step.

For figures legends and other plots, we want the cell types to be in a specific order that is consistent and intuitive. So we define a function to set the order of the cell type labels for any given list of cell types. We also want the cell types to have consistent colors accross figures where applicable, so we create a helper function for that too.

```
cellTypes <- c('B Cells',
               'CD4+ Naive T',
               'CD4+ Memory T',
               'CD4+ Effector T',
               'CD4+ Reg T',
               'CD8+ Naive T',
               'CD8+ Memory T',
               'CD8+ Effector T',
               "NKT CD4+",
               "NKT CD8+",
               "NKT CD4- CD8-",
               'NK',
               'CD16- Monocytes',
               'CD16+ Monocytes',
               'DC',
               'Erythrocytes',
               'Unknown')

fixOrder <- function(x) {
  order <- cellTypes[which(cellTypes %in% unique(x))]
  factor(x, levels=order)
}
```

```
getCellPallette <- function(x) {
  x <- factor(x)
  cols<-c('forestgreen', 'chartreuse', 'orange', 'cornflowerblue',
                'magenta', 'darkolivegreen4',
                'indianred1', 'tan4', 'darkblue',
                'mediumorchid1','firebrick4',  'yellowgreen', 'lightsalmon', 'tan3', '#DDAD4B',  'red2'
  ix <- match(levels(x), cellTypes)
  return(cols[ix])
}

  # cols<-c('forestgreen', 'red2', 'orange', 'cornflowerblue',
  #         'magenta', 'darkolivegreen4',
  #         'indianred1', 'tan4', 'darkblue',
  #         'mediumorchid1','firebrick4',  'yellowgreen', 'lightsalmon', 'tan3',
  #         "tan1",'darkgray', 'wheat4', '#DDAD4B', 'chartreuse', 'seagreen1',
  #         'moccasin', 'mediumvioletred', 'seagreen','cadetblue1',
  #         "darkolivegreen1" ,"tan2" ,   "tomato3" , "#7CE3D8","gainsboro")

types <- fixOrder(types)
saveRDS(types, "Final_Data/cell_types.rds")
```

Figure 1

Figure 1 in the paper is a schematic of the analysis prepared in Microsoft PowerPoint and InkScape. No data analysis was performed for this figure.

Figure 2

With the data processed as above, we are ready to generate the 9 panels of figure 2. First we define a theme to tweak plotting formatting (I prefer bold axis labels, etc.) and keep things consistent. We also create a helper function for the UMAP plots.

```
library(ggplot2)

my_theme <- theme_bw() + theme(
  strip.background = element_blank(),
  strip.text = element_text(face = "bold", margin = margin(0,0,5,0), size=10),
  panel.spacing = unit(1, "lines"),
  plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"),
  plot.title = element_text(size = 10,
                            margin=margin(0,0,5,0),
                            face="bold"),
  axis.text = element_text(size=10),
  axis.title.y = element_text(face = "bold",
                              margin = margin(t = 0, r = 0, b = 0, l = 0, unit="pt")),
  axis.title.x = element_text(face = "bold",
                              margin = margin(t = 2, r = 0, b = 0, l = 0))
)

my_theme_wide_lab <- theme(
  axis.title.y = element_text(face = "bold", margin = margin(t = 0, r = 10, b = 0, l = 0))
```

```
)

my_theme_no_labs <- my_theme + theme(axis.title = element_text(size=0))

my_theme_no_space <- my_theme + theme(plot.margin=unit(c(0,0,0,0),"cm"))

my_theme_wider_lab <- theme(
  axis.title.y = element_text(face = "bold", margin = margin(t = 0, r = 20, b = 0, l = 0))
)
plotUmap <- function(p1, p2, color, label, palette, legend.position = "none") {
  dat <- melt(data.frame(UMAP1 = p1, UMAP2 = p2, color=color),
              id.vars = c("UMAP1", "UMAP2", "color"))
  ggplot(dat, aes(x=UMAP2, y=UMAP1, color = color)) +
    geom_point(alpha=0.5, pch=19, size=0.1, stroke=0) +
    annotate("text", x = -10, y = -15, size=3, hjust=0, label = label) +
    theme_bw()  +
    my_theme +
   ylim(c(-15,15)) +
   xlim(c(-15,15)) +
    scale_color_manual(values=palette) +
    theme(legend.position = legend.position)

}
```

Here we used the UMAP coordinates calculated above together with the assigned cell types and publically available metadata to generate the nine panels of figure 2. Note that we keep the x and y axis ranges fixed, and a few cells fall outside of this range resulting in warnings about missing values.

```
library(reshape2)
library(cowplot)

types <- readRDS("Final_Data/cell_types.rds")
md <- readRDS("Final_Data/metadata.rds")

# types is the type of each cell, defined above
pal_cell <- getCellPallette(types)
colors = grDevices::colors()[grep('gr(a|e)y', grDevices::colors(), invert = T)]
set.seed(1000)
pal_id <- sample(colors, 38)
pal_surv <- c("#1F70B0", "#BE1D2A")

umap <- readRDS("Final_Data/PBMC_merged_filtered_normalized_pc_umap.rds")
ix.cell <- match(umap$id, md$id)
a1 <- plotUmap(umap$UMAP1, umap$UMAP2, umap$id, "Normalized, color = id", pal_id)
a2 <- plotUmap(umap$UMAP1, umap$UMAP2, types, "Normalized, color = cell type", pal_cell, legend.positio
a3 <- plotUmap(umap$UMAP1, umap$UMAP2, md$mort_72h[ix.cell], "Normalized, color = survival", pal_surv,
leg1 <- get_legend(a2 +
        guides(color=guide_legend(override.aes = list(size=5, alpha=1), title="Cell Type")))
leg2 <- get_legend(a3 +
        guides(color=guide_legend(override.aes = list(size=5, alpha=1), title="Survival")))
a2 <- a2 +  theme(legend.position = "none")
a3 <- a3 + theme(legend.position = "none")

umap <- readRDS("Final_Data/PBMC_merged_filtered_alra_pc_umap.rds")
```

```
b1 <- plotUmap(umap$UMAP1, umap$UMAP2, umap$id, "Imputed, color = id", pal_id)
b2 <- plotUmap(umap$UMAP1, umap$UMAP2, types, "Imputed, color = cell type", pal_cell)
b3 <- plotUmap(umap$UMAP1, umap$UMAP2, md$mort_72h[ix.cell], "Imputed, color = survival", pal_surv)

umap <- readRDS("Final_Data/PBMC_merged_filtered_alra_pc_harmony_umap.rds")
c1 <- plotUmap(umap$UMAP1, umap$UMAP2, umap$id, "Donor corrected, color = id", pal_id)
c2 <- plotUmap(umap$UMAP1, umap$UMAP2, types, "Donor corrected, color = cell type", pal_cell)
c3 <- plotUmap(umap$UMAP1, umap$UMAP2, md$mort_72h[ix.cell], "Donor corrected, color = survival", pal_su

png("Final_Data/fig2.png", height=2200, width=2000, type = "cairo", res = 200)
p <- plot_grid(a1, a2, a3, b1, b2, b3, c1, c2, c3,
               label_size = 18,
               nrow = 3,
               ncol = 3,
               labels = c("A", "B", "C", "D", "E", "F", "G", "H", "I"))
leg <- plot_grid(leg1, leg2, ncol=2, rel_widths=c(0.75, 0.25))
p <- plot_grid(p, leg,
               nrow=2,
               ncol=1,
               rel_heights = c(0.9, .1))
print(p)
dev.off()
```

Figure 3

Figure 3 provided further detail about the scRNAseq classifiation of cells. Panel A breaks out each individual cell type to better visualize their clustering (based on the harmony corrected PC loadings).

```
umap <- readRDS("Final_Data/PBMC_merged_filtered_alra_pc_harmony_umap.rds")
types <- readRDS("Final_Data/cell_types.rds")

fig3a <- function() {
  data <- data.frame(UMAP1 = umap$UMAP1, UMAP2 = umap$UMAP2, id = umap$id, type = types)
  data_m <- melt(data, id.vars = c("UMAP1", "UMAP2", "id", "type"))
  # make sure our cell labels and colors match figure 2
  data_m$type <- fixOrder(data_m$type)
  ix.un <- which(data_m$type == "Unknown")
  data_m <- data_m[-ix.un,]
  ggplot(data_m, aes(x=UMAP2, y=UMAP1, color = type)) +
    geom_point(alpha=0.2, pch=19, size=1, stroke=0) +
    theme_bw()  +
    scale_color_manual(values=getCellPallette(data_m$type)) +
    facet_wrap(~type) +
    my_theme +
    theme(legend.position = 'none')
}

F3A <- fig3a()
```

Panel B shows the expression of key surface markers in the cell clusters.

```
types <- readRDS("Final_Data/cell_types.rds")

fig3b <- function() {
```

```
  data <- readRDS("Final_Data/PBMC_merged_filtered_alra_pc_harmony_umap.rds")
  data$type <- types
  data$CD19 <- dat.imp[ , 'CD19']
  data$'CD3' <- dat.imp[ , 'CD3E']
  data$'CD4' <- dat.imp[ , 'CD34']
  data$'CD8' <- dat.imp[ , 'CD8A']
  data$'CD2' <- dat.imp[ , 'CD2']
  data$'CD25' <- dat.imp[ , 'IL2RA']
  data$'FOXP3' <- dat.imp[ , 'FOXP3']
  data$'CD56' <- dat.imp[ , 'NCAM1']
  data$'CD14' <- dat.imp[ , 'CD14']
  data$'CD16' <- dat.imp[ , 'FCGR3A']
  data_m <- melt(data, id.vars = c("UMAP2", "UMAP1", "id", "type"))

  ggplot(data_m, aes(x=UMAP2, y=UMAP1, color=value)) +
    geom_point(size=0.5, stroke=0) +
    scale_color_gradientn(colors = c("#00348E00", "cyan", "yellow", "orange", "red"), name = "Log Counts
    facet_wrap(~variable, nrow = 2) +
    theme_bw() +
    my_theme
}

F3B <- fig3b()
```

To validate our scRNASeq analysis, we also performed FACS analysis on the same samples to directly measure the surface marker expression of major lymphocyte population markers. The percentages of the lymphocyte gate comprised of each subtype as determined by FACS are included in the metadata file. Proportion of lymphocytes in each population as assigned by FACS vs. scRNASeq is compared in Panel C.

```
library(ggpubr)

fig3c <- function() {
  # accumulate all our cell counts per sample, dropping non-lymphocytes
  ids <- gsub("\\..*", "", rownames(dat.imp))
  sc.counts <- as.data.frame.matrix(table(ids, types))[ , -c(16:17)]

  # aggregate some sub-populations to match flow populations
  sc.counts$`T Cells` <- sc.counts$`CD4+ Naive T` +
                         sc.counts$`CD4+ Memory T` +
                         sc.counts$`CD4+ Effector T` +
                         sc.counts$`CD4+ Reg T` +
                         sc.counts$`CD8+ Naive T` +
                         sc.counts$`CD8+ Memory T` +
                         sc.counts$`CD8+ Effector T` +
                         sc.counts$`NKT CD4+` +
                         sc.counts$`NKT CD8+` +
                         sc.counts$`NKT CD4- CD8-`
  sc.counts$`CD4+ T Cells` <- sc.counts$`CD4+ Naive T` +
                         sc.counts$`CD4+ Memory T` +
                         sc.counts$`CD4+ Effector T` +
                         sc.counts$`CD4+ Reg T` +
                         sc.counts$`NKT CD4+`

  sc.counts$`CD8+ T Cells` <- sc.counts$`CD8+ Naive T` +
```

```
                        sc.counts$`CD8+ Memory T` +
                        sc.counts$`CD8+ Effector T` +
                        sc.counts$`NKT CD8+`

# and convert to % of lymphocytes
sc.lymphs <- table(grepl("CD4", types) |
                   grepl("CD8", types) |
                   grepl("B Cell", types) |
                   grepl("NK", types), ids)[2,]

sc.props <- sweep(sc.counts, 1, sc.lymphs, "/")
sc.props <- as.data.frame.matrix(sc.props)
sc.props <- sc.props[match(md$id, rownames(sc.props)),]

# sanity check
all.equal(as.character(md$id), rownames(sc.props))

# extract the populations we are interested in
sc <- data.frame("B Cells" = sc.props$`B Cells`)
flow <- data.frame("B Cells" = md$`B Cells`)

# put space back in column names
colnames(sc)[1] <- "B Cells"
colnames(flow)[1] <- "B Cells"

sc$`T Cells` <- sc.props$`T Cells`
flow$`T Cells` <- md$`T Cells`

sc$`CD4+ T Cells` <- sc.props$`CD4+ T Cells`
flow$`CD4+ T Cells` <- md$`CD4+ T Cells`

sc$`CD8+ T Cells` <- sc.props$`CD8+ T Cells`
flow$`CD8+ T Cells` <- md$`CD8+ T Cells`

sc$`NK Cells` <- sc.props$NK
flow$`NK Cells` <- md$`NK Cells`

# assemble, melt, and plot
flow$ID <- md$id
sc$ID <- md$id

flow_m <- melt(flow, id.vars = c("ID"))
sc_m <- melt(sc, id.vars = c("ID"))

dat <- data.frame(ID = flow_m$ID, Population = flow_m$variable, Flow=flow_m$value, scRNASeq = sc_m$va
ggplot(dat, aes(x = Flow, y = scRNASeq)) +
  geom_smooth(method='lm',formula=y~x, color="#cccccc", se = FALSE) +
  geom_point() +
  stat_cor(method = "pearson", size=3, label.x = 0, label.y = 0.95, color="black") +
  facet_wrap( ~ Population, scales = "free") +
  scale_x_continuous(limits=c(0,1)) + scale_y_continuous(limits=c(0,1)) +
  xlab(label = "Flow Cytometry") +
  theme_bw(base_size=10) +
```

```
    my_theme +
    my_theme_wide_lab +
    theme(
      axis.text.x = element_text(angle = 315, hjust = 0),
      legend.position = "none"
    )
}

F3C <- fig3c()
```

And figure 3 can then be assembled

```
png("Final_Data/fig3.png", height=3400, width=1500, type = "cairo", res = 220)
p <- plot_grid(F3A + theme(legend.position="none",
                           plot.margin=unit(c(1,2,1,.5),"cm")),
               F3B + theme(plot.margin=unit(c(1,0,1,.5),"cm")),
               F3C + theme(plot.margin=unit(c(1,2,1,.5),"cm")) + my_theme_wide_lab,
               #rel_widths = c(4,1.2,4,1.2),
               label_size = 18,
               nrow = 3,
               ncol = 1,
               rel_heights = c(1.1, .6, .8),
               labels = c("A", "B", "C"))
print(p)
dev.off()
```

Figure 4

Panel A of figure 4 displays the proportion of cells in each subtype for each patient, stratified by surviving vs. non-surviving patients.

```
library(reshape2)
library(rstatix)
library(ggpubr)
dat.imp <- readRDS("Final_Data/PBMC_merged_filtered_alra.rds")
md <- readRDS("Final_Data/metadata.rds")
fig4a <- function() {
  ids <- gsub("\\..*", "", rownames(dat.imp))

  # cell counts per patient, dropping unknown cells
  # and erythrocytes
  sc.counts <- as.data.frame.matrix(table(ids, types))[ , -c(16:17)]

  # now convert to proportion of total cells
  totals <- apply(sc.counts, 1, sum)
  sc.counts.p <- sweep(sc.counts, 1, totals, "/")


  ix <- match(rownames(sc.counts.p), as.character(md$id))
  # sanity check
  all.equal(rownames(sc.counts.p), as.character(md$id)[ix])

  sc.counts.p$Survival <- md$mort_30d[ix]
  sc.counts.p$ID <- md$id[ix]
```

```
  dat_m <- melt(sc.counts.p, id.vars = c("Survival", "ID"))
  dat_m$variable <- fixOrder(dat_m$variable)

  # now calculated p-value for t-test between survival groups
  # for each cell type, adjusting for multiple comparisons
  # and also format resulting labels and locations for plotting
  stat.test <-  dat_m %>%
    group_by(variable) %>%
    t_test(value ~ Survival)
  stat.test$p <- p.adjust(method = "holm", stat.test$p)
  stat.test$yloc <- 0.9 * unlist(tapply(dat_m$value, INDEX = dat_m$variable, max))
  stat.test$x <- rep("Survived", nrow(stat.test))
  stat.test$p <- paste0("  adj. p=", format(round(stat.test$p,3), nsmall=3))
  median2 <- function(x) { t <- median(x, na.rm=TRUE); return(data.frame(ymin=t, ymax=t, y=t))}

  # And plot.  Colors will match the colors in Figure 2, and Figure 3A
  ggplot(dat_m, aes(x=Survival, y=value, color=variable)) +
    stat_summary(fun.data = median2, size = 0.5, geom="crossbar", width=0.6) +
    stat_pvalue_manual(stat.test, label = "p", remove.bracket = TRUE,
                       x="x", hjust=0,
                       y.position = "yloc",
                       tip.length = 0,
                       size = 3.5,
                       color="#444444") +
    geom_jitter(width=0.07) +
    scale_color_manual( values = getCellPallette(dat_m$variable)) +
    ylab("Proportion of all Cells") +
    facet_wrap( ~ variable, scales = "free", nrow = 4) +
    my_theme +
    my_theme_wide_lab +
    theme(legend.position = "none")
}

F4A <- fig4a()
```

As none of the major cell types seemed predictive of survival, we wanted to drill into each cell type and find biological processes and surface markers that might distinguish surviving vs. non-surviving patients. Panel B displays the results of this analysis. For identification of differentially expressed genes and associated biological processes, we limited our analysis to those genes that had variable expression defined as having an expression level normalized robust z-score greater than 2. This approach is the same as that described in Zheng et al. (2017) and Macosko, et al. (2015).

Here is the function we used to calculate the dispersion for each gene:

```
dispersion <- function(x, dim=2, verbose=FALSE) {

  if(verbose)  message("Calculating means (1 of 4)")
  means <- apply(x, dim, mean, na.rm=TRUE)

  if(verbose)  message("Calculating dispersion (2 of 4)")
  disp <- apply(x, dim, function(x) { var(x, na.rm=TRUE) / mean(x, na.rm=TRUE)})

  if(verbose)  message("Binning (3 of 4)")
  bins <- cut(means, quantile(means, seq(0, 1, len = 20)), include.lowest = TRUE)
```

```
  if(verbose)  message("Normalizing (4 of 4)")
  rv <- tapply(disp, bins, function(y) { abs(y - median(y)) / mad(y)})
  rv <- unlist(rv)
  names(rv) <- gsub("\\[.*\\]\\.", "", names(rv))
  rv
}
```

We then used this function to identify the variable gene set based on cells of known type (as defined above) that were not erythrocytes.

```
types <- readRDS("Final_Data/cell_types.rds")

ix <- which(!types %in% c("Erythrocytes", "Unknown"))
dat <- dat.imp[ix,]
disp <- dispersion(dat)
ix <- which(disp > 2)
gg <- gsub(".*\\]\\.", "", names(disp))
gg <- gg[ix]

# remove pseudogenes and non-coding genes from a list of genes
expGenes <- function(x) {
  x <- x[-which(grepl("A\\w\\d*\\.\\d", x))]
  x <- x[-which(grepl("C\\d{1,2}orf", x))]
  x <- x[-which(grepl("^LINC\\d+$", x))]
  x
}
gg <- expGenes(gg)
ix <- which(colnames(dat) %in% gg)
dat <- dat[,ix]
saveRDS(dat, file = "Final_Data/PBMC_merged_filtered_alra_variable_genes.rds")
```

Now we need helper functions to calculate the proportion of cells within each subtype that express each gene for each patient.

```
library(doParallel)

applyById <- function(x, ids, FUN=function(x) { sum(x>0) }) {
  ids.x <- gsub("\\..*", "", names(x))
  rv <- foreach(i=ids, .combine = c) %do% {
    ix <- which(ids.x == i)
    if(length(ix)) {
      FUN(x[ix])
    } else {
      0
    }
  }
  names(rv) <- ids
  rv
}

# convenience wrapper
countsById <- function(x, ids) {
  applyById(x, ids, length)
}
```

```
# convenience alias
positiveById <- function(x, ids) {
  applyById(x,ids)
}

R_na_zero <- function(x) {
  ix <- which(is.na(x))
  if(length(ix)) x[ix] <- 0
  x
}


genesProp <- function(types, dat, cluster=TRUE) {
  calcRatio <- function(x, ix.s, ix.d, ids.s, ids.d) {
    pos.s <- positiveById(x[ix.s], ids.s)
    tot.s <- countsById(x[ix.s], ids.s)
    # if there are no cells for an id, assum proportion is zero
    prop.s <- R_na_zero(pos.s/tot.s)

    pos.d <- positiveById(x[ix.d], ids.d)
    tot.d <- countsById(x[ix.d], ids.d)
    prop.d <- R_na_zero(pos.d/tot.d)

    prop.tot <- sum(pos.s, pos.d) / sum(tot.s, tot.d)

    ratio=log2(median(prop.d)/median(prop.s))
    if(is.nan(ratio)) ratio <- 0
    data.frame(logratio=ratio,
               Survived=median(prop.s),
               Died=median(prop.d),
               prop.positive = prop.tot,
               p=suppressWarnings(wilcox.test(prop.s, prop.d)$p.value))
  }

  md <- readRDS("Final_Data/metadata.rds")
  ids <- gsub("\\..*", "", rownames(dat))
  ix <- match(ids, md$id)
  survival <- md$mort_72h[ix]

  if(cluster) {
    cl <- makeCluster(detectCores() - 1)
    registerDoParallel(cl)
  }

  ratios <- foreach(t = levels(types), .packages = c("dplyr", "foreach"), .export = c("applyById",
                                                    "positiveById",
                                                    "countsById",
                                                    "R_na_zero")) %dopar% {
    ix.type <- which(types == t)
    dat.type <- dat[ix.type, ]
    ix.s <- which(survival[ix.type] == "Survived")
    ix.d <- which(survival[ix.type] == "Died")
    ids.s <- unique(ids[which(survival == "Survived")])
```

```
    ids.d <- unique(ids[which(survival == "Died")])
    rv <- apply(dat.type, 2, calcRatio, ix.s, ix.d, ids.s, ids.d)
    rv <- do.call(rbind, rv)
    rv$p.adj <- p.adjust(rv$p, "BH")
    rv$gene <- colnames(dat)
    rv
  }
  if(cluster) {
    stopCluster(cl)
  }
  names(ratios) <- levels(types)
  ratios
}

types <- readRDS("Final_Data/cell_types.rds")
types.p <- factor(types[which(!types %in% c("Erythrocytes", "Unknown"))])
dat <- readRDS("Final_Data/PBMC_merged_filtered_alra_variable_genes.rds")
props <- genesProp(types.p, dat)
names(props) <- levels(types.p)
saveRDS(props, file="Final_Data/gene_props.rds")
```

We can then plot a heatmap of the gene proportions. Red genes are those expressed in a higher proportion of cells from deceased patients, while blue genes are those expressed in a higher proportion of cells from surviving patients. The intensity of the color is proportional to the adjusted (FDR) p-value (t-test) for each gene (within each subtype).

```
library(pheatmap)
library(ggplotify)
library(cowplot)
fig4b_1 <- function() {
  props <- readRDS("Final_Data/gene_props.rds")
  # take 1 - adjusted pvalue (so the most significant genes are the
  # extremes of the scale), and set sign to the direction of the
  # fold change.
  pv <- foreach(d=props, .combine=cbind) %do% {
    (1 - d$p.adj) * sign(d$logratio)
  }

  pv[which(is.na(pv))] <- 0
  colnames(pv) <- levels(types.p)
  rownames(pv) <- props$`B Cells`$gene

  # remove completely uninformative genes
  ix.rm <- which(apply(pv,1,sum)==0)
  pv <- pv[-ix.rm,]

  hm <- pheatmap(pv, silent=TRUE, legend=FALSE,
                 cluster_cols = FALSE,
                 border_color = NA,
                 cluster_rows = TRUE,
                 show_rownames = FALSE,
                 color = colorRampPalette(c("#005AC8", "white", "#FA2800"))(11),
                 treeheight_col = 10,
                 treeheight_row = 30)
```

```
  # save the dendrogram for further analysis below
  dg <- rev(as.dendrogram(hm$tree_row))
  saveRDS(dg, file="Final_Data/gene_prop_dendrogram.rds")

  # grab the legend
  hm.leg <- pheatmap(pv,
                     silent=TRUE, breaks = c(seq(-1,1,0.2)),
                     legend_breaks = seq(-.8,.8,.2),
                     legend=TRUE,
                     color = colorRampPalette(c("#005AC8", "white", "#FA2800"))(11))$gtable$grobs[[6]]

  hm.leg <- as.ggplot(hm.leg) + theme(plot.margin = unit(c(1,0,0,0), "cm"))
  hm.g <- as.ggplot(hm)


  plot_grid(hm.leg,
            hm.g + theme(plot.margin = unit(c(0.5,-0.2,0,0), "cm")),
            ncol=2,
            labels=c(" ", "Variable Genes"),
            label_size = 10,
            rel_widths=c(0.4, 1.8))
}

F4B1_pre <- fig4b_1()
```

In an effort to functionalize the gene clusters, we performed Gene Ontology term enrichment analysis on the genes in each of the major subclusters. We create a helper function that accepts a dendrogram, and the index of a subtree, and then runs GO enrichment analysis on the genes in that subtree. The function also optionally plots the subtree so you can visually verify that you know which subtree is being analyzed by cross referencing to the whole dendrogram.

```
library(dendextend)
library(clusterProfiler)
library(org.Hs.eg.db)
library(stringr)

# get 1:1 mapping of gene symbol to entrez gene, dropping NAs
symToEG <- function(x) {
  x <- x[which(x %in% ls(org.Hs.egSYMBOL2EG))]
  x <- mget(x, org.Hs.egSYMBOL2EG)
  unlist(lapply(x, function(i) { i[1]}))
}

# k is the number of subtrees to cut the dendrogram into, i is the
# subtree of interest. If heatmap=TRUE, exp must be supplied (the
# full data matrix originally used for clustering)
dendGo <- function(dend, k, i, heatmap=FALSE, go=TRUE, exp=NULL) {
  gt <- NULL
  clusters <- dendextend::cutree(dend, k)
  genes <- labels(dend)
  labels(dend) <- clusters[order.dendrogram(dend)]
  ix <- which(clusters[order.dendrogram(dend)]==i)
  labels(dend) <- genes
  gg <- labels(dend)[ix]
```

```
  if(heatmap) {
    if(is.null(exp)) {
      stop("If heatmap=TRUE, must supply data matrix exp")
    }
    p <- pheatmap(exp[which(rownames(exp) %in% gg),],
             cluster_cols = FALSE,
             border_color = NA,
             cluster_rows = TRUE,
             show_rownames = FALSE,
             color = colorRampPalette(c("#005AC8", "white", "#FA2800"))(11),
             treeheight_col = 10,
             treeheight_row = 50)
    print(p)

  }
  if(go) {
    go_test <- enrichGO(symToEG(gg), OrgDb='org.Hs.eg.db', pvalueCutoff = 0.05)
    go_test@result$Description <- go_test@result$Description
    gt <- go_test
    #return(dotplot(go_test, font.size=10) + scale_size_continuous(range = c(1, 6)))
  }
  return(gt)
}
```

We then step through each of 12 (mutually exclusive, but exhaustive) subtrees and test for enriched GO terms. Although the number of subtrees is arbitrary, the identification of subtrees once the number is chosen is systematic and automated, performed using Tal Galili's `dendextend` package.

```
library(dendextend)
library(doParallel)

dg <- readRDS(file="Final_Data/gene_prop_dendrogram.rds")
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)

go_enrich <- foreach(i = 1:12, .packages = c("dendextend", "clusterProfiler", "stringr")) %dopar% {
#go_enrich <- foreach(i = 1:20) %do% {
  dendGo(dg, 12, i)
}

stopCluster(cl)
saveRDS(go_enrich, "Final_Data/go_enrichment.rds")
```

It would be helpful to create an annotation bar that will indicate where exactly the sub trees are, and to label these bars with any significantly enriched GO terms. Because space is limited, a maximum of 2 terms are annotated on the plot, and terms are truncated for length as needed.

```
topTerms <- function(enr, n=3, strlen = 40) {
  termlist <- foreach(i = 1:length(enr)) %do% {
    r <- enr[[i]]@result

    # truncate and remove duplicate terms that are identical after truncation
    r$Description <- str_trunc(r$Description, strlen)
    ix.rm <- which(duplicated(r$Description))
    r <- r[-ix.rm,]
```

```
    ix <- which(r$p.adjust < 0.05)
    if(length(ix)) {
      if(length(ix) > n) {
        terms <- r$Description[ix][1:n]
      } else {
        terms <- r$Description[ix]
      }
    } else {
      terms <- ""
    }
    terms
  }
  termlist
}

dendnote <- function(dend,
                     k,
                     offset = 0.125,
                     cols=c("gray", "black"),
                     reverse=TRUE,
                     sort.labels=FALSE,
                     enr = NULL) {
  if(reverse) {
    dend <- rev(dend)
  }
  order <- order.dendrogram(dend)
  clusters <- cutree(dend, k)
  levels <- unique(clusters[order])
  dn <- data.frame(labels=unique(clusters[order]))
  if(length(enr)) {
    dn$label = sapply(topTerms(enr, 2)[levels], paste, collapse="\n")
  } else {
    dn$label = ""
  }
  if(sort.labels) {
    dn$id <- sort(levels, decreasing=TRUE)
  } else {
    dn$id <- levels
  }
  dn$at <- which(!duplicated(clusters[order]))
  dn$length <- as.vector(table(clusters[order])[levels])
  dn$at_ann <- dn$at + dn$length
  dn$at <- dn$at + 0.5 * dn$length
  dn$col <- cols
  dn$offset <- c(0,offset)
  ggplot(dn, aes(x=1+offset, y=at, height=length, fill=col, label=id)) +
    geom_tile(width=0.25) +
    geom_text(aes(x=2, col=col), size=3.5) +
    geom_text(aes(x=3, y = at, label=label, col=col), lineheight=0.8, size=2.5, hjust=0, vjust=0.5) +
    scale_fill_manual(values = cols) +
    scale_color_manual(values = cols) +
    coord_cartesian(xlim = c(0, 8),
                    clip = 'off') +
```

```
    theme(axis.line=element_blank(),
          axis.text.x=element_blank(),
          axis.text.y=element_blank(),axis.ticks=element_blank(),
          axis.title.x=element_blank(),
          axis.title.y=element_blank(),legend.position="none",
          panel.background=element_blank(),panel.border=element_blank(),panel.grid.major=element_blank()
          panel.grid.minor=element_blank(),plot.background=element_blank())

}
```

We can then add the annotation bar to our heatmap.

```
dg <- readRDS(file="Final_Data/gene_prop_dendrogram.rds")
go_enrich <- readRDS("Final_Data/go_enrichment.rds")

# fig4b panel 1 with GO annotation
fig4b_1_ann <- function() {
  plot_grid(F4B1_pre,
            dendnote(dg,
                     12,
                     reverse=FALSE,
                     sort.labels = TRUE,
                     enr = go_enrich,
                     cols = c("black", "#006666")) +
            theme(plot.margin=margin(-0.1,2,2.25,-0.4,"cm")),
            rel_widths = c(1.2, 1))
}
F4B1 <- fig4b_1_ann()
```

We did not have room in the figure to display every significant GO term. But of course, we want to report them, and we do so in supplemental table S4. For figure 4 in the manuscript, we wanted the node ids to be in numerical order, top to bottom. But this is not the way `cutree` numbers the subtrees (it numbers them hierachically as it encounters them). Below we create a table of significant GO terms with both the original node id as assigned by `cutree` and the node id as presented in the figure.

```
go_enrich <- readRDS("Final_Data/go_enrichment.rds")
dg <- readRDS(file="Final_Data/gene_prop_dendrogram.rds")
clusters <- cutree(dg, 12)
order <- order.dendrogram(dg)
pub_id <- match(c(1:12), rev(unique(clusters[order])))

res <- NULL
for(i in 1:length(go_enrich)) {
  r <- go_enrich[[i]]@result
  ix <- which(r$p.adjust < 0.05)
  if(length(ix)) {
    res <- rbind(res, cbind(pub_id[i], i, r$Description[ix], r$p.adjust[ix]))
  }
}
colnames(res) <- c("Node_ID_pub", "Node_ID_orig", "GO_Term", "adj_p")
write.table(res, quote=FALSE, row.names=FALSE, sep="\t", file="Final_Data/table_s5.tab")
```

In order to facilitate future isolation of interesting cell subpopulations by FACS, we would also like to focus in on surface markers specifically. We obtained a list of known clusters of differentiations, and their alternative gene symbols (if any) from the Cell Surface Protein Atlas and used this to filter our gene list. The resulting

list of genes is available as table_s4.txt from our manuscript (and is also included in the github repository).

```
types <- readRDS("Final_Data/cell_types.rds")
types.p <- factor(types[which(!types %in% c("Erythrocytes", "Unknown"))])
dat <- readRDS("Final_Data/PBMC_merged_filtered_alra_variable_genes.rds")
cds <- read.delim("Final_Data/table_s4.txt", as.is=TRUE, header=TRUE)

ix.cd <- which(colnames(dat) %in% cds$Gene_Symbol | grepl("^CD\\d", colnames(dat)))
props_cd <- genesProp(types.p, dat[, ix.cd])
names(props_cd) <- levels(types.p)
saveRDS(props_cd, file="Final_Data/cd_props.rds")
```

We can then plot this heatmap as well

```
types <- readRDS("Final_Data/cell_types.rds")
types.p <- factor(types[which(!types %in% c("Erythrocytes", "Unknown"))])
props_cd <- readRDS(file="Final_Data/cd_props.rds")

fig4b_2 <- function()  {
  pv <- foreach(d=props_cd, .combine=cbind) %do% {
    (1 - d$p.adj) * sign(d$logratio)
  }
  colnames(pv) <- levels(types.p)

  pv[which(is.na(pv))] <- 0
  pv <- as.data.frame(pv)
  hm <- pheatmap(pv, silent=TRUE, legend=FALSE,
           cluster_cols = FALSE, breaks = c(seq(-1,1,0.2)),
           cluster_rows = TRUE, border_color = NA,
           show_rownames = FALSE,
           color = colorRampPalette(c("#005AC8", "white", "#FA2800"))(11),
           treeheight_col = 10,
           treeheight_row = 20)
  hm <- as.ggplot(hm)
  plot_grid(hm + theme(plot.margin = unit(c(0.5,-0.2,0,0), "cm")),
            ncol=1,
            labels=c("Surface markers"),
            label_size = 10)
}
F4B2 <- fig4b_2()
F4B <- plot_grid(F4B1, F4B2, labels=c("B", ""), ncol=2, rel_widths = c(1, 0.4))
```

We are most interested in those surface markers that exhibited a strong fold in proportion of positive cells between surviving and non-surviving patients and that have a p-value corresponding to a relatively low false discovery rate. For our purposes, we will screen for surface markers with a fold change of at least 1.5 (in either direction) and an FDR $< 20\%$. Surface markers meeting these requirements were identified for the CD8+ Naive T Cells and CD8+ NKT cells. The corresponding candidate markers are provided in supplemental table S6 in the manuscript.

Finally, we would like to perform survival analysis based on the novel markers we have selected. Here we create a helper function to perform the analysis and plot the results.

```
library(survival)
survAnalysis <- function(cellp,
                         cutoff = NULL,
                         days = 3,
```

```
                          pop.name = "Positive cells / total lymphs:") {
  md <- readRDS("Final_Data/metadata.rds")
  md <- md[which(md$id %in% names(cellp)), ]
  ix <- match(md$id, names(cellp))
  md$cellp <- cellp[ix]
  if(days == 3) {
    md$surv_time[which(md$surv_time > 3)] <- 3
    surv <- Surv(time = md$surv_time, event = md$mort_72h == "Died")
  } else {
    md$surv_time[which(md$surv_time > 30)] <- 30
    surv <- Surv(time = md$surv_time, event = md$mort_30d == "Died")
  }
  if(is.null(cutoff)) {
    cutoff <- median(cellp)
  }
  md$surv <- surv
  md$high_count <- factor(md$cellp > cutoff, labels=c("High", "Low"))
  fit<-coxph(surv ~ high_count, data=md)
  ggsurvplot(survfit(surv ~ high_count, data=md),
             color = "black",
             ggtheme = my_theme_no_labs +
                       my_theme_no_space +
                       theme(legend.title = element_text(size=10, face="bold"),
                             legend.text = element_text(size=10)),
             data=md,
             pval=TRUE,
             pval.coord = c(0.6*days, 0.1),
             pval.size = 3.5,
             linetype = "strata",
             risk.table = FALSE,
             legend.title = pop.name,
             legend.labs = c("Low", "High") )
}
```

```
library("rstatix")  # https://github.com/kassambara/rstatix
fig4cd <- function() {
  dat <- readRDS("Final_Data/PBMC_merged_filtered_alra_variable_genes.rds")
  types <- readRDS("Final_Data/cell_types.rds")
  types.p <- factor(types[which(!types %in% c("Erythrocytes", "Unknown"))])
  ids <- gsub("\\..*", "", rownames(dat))
  pd <- data.frame(ids = ids, type = types.p)

  md <- readRDS("Final_Data/metadata.rds")
  ix <- match(pd$ids, md$id)
  pd$Survival <- factor(md$mort_72h[ix], labels=c("Survived", "Died"))
  pd$CD8NKTCD52 <- dat[ , 'CD52'] > 0
  pd$CD8NKTCD3G <- dat[ , 'CD3G'] > 0
  pd$CD8NKTDP <- pd$CD8NKTCD3G & pd$CD8NKTCD52

  cd8_nkt <- table(pd$id, pd$type)[,"NKT CD8+"]
  cd8_nkt_cd52 <- table(pd$id,  pd$CD8NKTCD52, pd$type)[, 2, "NKT CD8+"]
  cd8_nkt_cd3g <- table(pd$id,  pd$CD8NKTCD3G, pd$type)[, 2, "NKT CD8+"]
  cd8_nkt_dp <- table(pd$id,  pd$CD8NKTDP, pd$type)[, 2, "NKT CD8+"]
```

```
  cellp <- cd8_nkt_cd52 / cd8_nkt
  cellp <- cellp[-which(is.nan(cellp))]
  p1 <- survAnalysis(cellp, pop.name = "CD8+ NKT/CD52+")[[1]] +
    my_theme_wide_lab

  cellp <- cd8_nkt_cd3g / cd8_nkt
  cellp <- cellp[-which(is.nan(cellp))]
  p2 <- survAnalysis(cellp, pop.name = "CD8+ NKT/CD3G")[[1]] +
    my_theme_wide_lab

  p3 <- survAnalysis(cellp, days = 30, pop.name = "CD8+ NKT/CD3G")[[1]] +
    my_theme_wide_lab

  list(p1,p2,p3)

}
F4CDE <- plot_grid(plotlist = fig4cd(), ncol=3, labels=c("C", "D", "E"))
```

And we can now assemble figure 4

```
fig4<- function() {
  F4AB <- plot_grid(F4A, F4B, labels=c("A", ""), rel_widths = c(1, 1),  ncol=2)
  p <- plot_grid(F4AB, F4CDE, nrow=2, rel_heights = c(2,1))

  png("Final_Data/fig4.png", height=2000, width=3000, type = "cairo", res = 200)
  print(p)
  dev.off()
}
fig4()
```