# SPARTIN Supplementary material

Nate Osher

# Contents

# S. 1

# Preface

The purpose of this document is to serve as a "living" supplement to the first paper on the SPARTYN pipeline. While the paper itself will be static upon publication, this document will serve as both an ongoing supplement as well as a central location to give updated, detailed information on the pipeline and methods.

If you have a question that is not adequately addressed in this supplement, please email oshern (at) umich.edu.

# S. 2

# Model Fitting

Given a quadrature $\mathbf{u}$ on $A$ with corresponding weights $\mathbf{w}$, the pseduolikelihood function (equation (3) in the main paper) can be approximated by

$$PL(\theta|\mathbf{x}) \approx \prod_{x_i \in \mathbf{x}} \lambda(x_i|\theta, \mathbf{x}) \exp(-\sum_{u_j \in \mathbf{u}} \lambda(u_j|\theta, \mathbf{x}) w_j) \tag{S1}$$

The density of the selected quadrature $\mathbf{u}$ on $A$ is chosen to balance the accuracy of the approximation against the computational load that larger quadratures impose.

Using this approximation of the pseudolikelihood function in place of the more standard likelihood function, Bayesian analysis can proceed in the style of King et al. (2012) by simply assigning priors to the parameters of interest and using techniques to sample from non-closed form posterior likelihoods. We assigned non-informative normal priors with mean 0 and variance $10^6$ to $\log(\gamma_{12})$, $\log(\beta_1)$, and $\log(\beta_2)$. In all analysis presented in the main paper the quadrature and weights used to estimate the integral in the pseudolikelihood function was generated by the spatstat package (Baddeley and Turner (2005)). Samples from the posterior were taken using JAGS via the R2jags package (Su et al. (2015)).

In order to use JAGS to fit the model, the so-called "Poisson zero trick" as described in Kruschke (2014). This allows for the fitting of a model with an arbitrary likelihood (or pseudolikelihood) function. The zero trick works by explicitly specifying the log-likelihood function of the model to be fit for each observation, and treating the resulting log-likelihood as the rate parameter of a Poisson distribution. Note that because the log-likelihood function can be negative, a constant value may have to be added in order to ensure the resulting value is strictly positive, since the rate parameter of a Poisson distribution must be strictly positive. But if the same value is used for all observations in a single model fitting across iterations, this is equivalent to multiplying the likelihood

by a constant, which does not affect the inference. A vector of zeros is passed to JAGS, and each zero is said to be observed from a Poisson distribution with rate parameter specified by the modified log-likelihood function evaluated for that observation. By definition of the probability mass function of the Poisson distribution, the resulting likelihood is the target likelihood for each observation. An analogous trick using the Bernoulli distribution and a vector of ones can be used as well- see Kruschke (2014) for details. Briefly, denoting the number of observed points $n_o$, the number of quadrature points $n_q$, and $n_o + n_q = N$, the underlying MCMC algorithm is as follows

---

**Algorithm 1S**: Bayesian Strauss Model Fitting

---

**Input:** Type vector $t_1, \dots, t_N$, weight vector $w_1, \dots, w_N$, and neighbor counts $c_1, \dots, c_N$

**For** $i = 1, \dots, n\_iter$ **do**:
| **For** $\theta = \beta_1, \beta_2, \gamma_{12}$ **do**:
| | Propose $\theta^*$
| | Evaluate PL( $\hat{}$* | )
| | Set $\theta^{(i)} = \theta^*$ with probability $min(\frac{PL(\theta^*)}{PL(\theta)}, 1)$; otherwise, set $\theta^{(i)} = \theta$
| **End For End For**

**Return:** $\beta_1^{(1)}, \dots, \beta_1^{(n\_iter)}$, $\beta_2^{(1)}, \dots, \beta_2^{(n\_iter)}$, $\gamma_{12}^{(1)}, \dots, \gamma_{12}^{(n\_iter)}$

---

Results were checked against both frequentist model fittings from the spatstat `ppm/hierstrauss` functions as well as STAN model fittings (Carpenter et al. (2017)) and yielded virtually indistinguishable results in both cases. Despite STAN sampling more quickly than JAGS, JAGS still outperformed STAN due to its somewhat shorter time in setting up the model for sampling.

# S. 3

# Biopsy Partition

In order to partition a given biopsy into non-overlapping sub-regions, we began with the smallest bounding rectangular window that contained all cells. We then applied an intensity thresholding algorithm in order to find the smallest possible window that still contained virtually all cells. This was accomplished using the `density.ppp` function from the `spatstat` package (Baddeley and Turner (2005)). The bounding rectangular window was broken up into small ($15\mu m \times 15\mu m$) pixels, the intensity of which was estimated as a function of the number of points per square unit of area of the pixel. These values were also "smoothed" between pixels according to a pre-selected bandwidth of $25\mu m$ chosen through experimentation. The final window was constructed by combining all pixels that were above a certain intensity into a window. The "pixel" size, smoothing bandwidth, and intensity threshold collectively determined the final window, and the same settings ($15\mu m^2$ pixel size and $25\mu m$ smoothing bandwidth) were applied across all biopsies.

# S. 4

# Cell Classification Model

## 4.1 Whole-Slide Image (WSI) Retrieval and Obtaining Training Labels

WSI from 20 different patients were used as our primary training and testing data. The images were imported from The Cancer Genome Atlas (TCGA) Genomic Data Commons Data Portal, which is a repository of validated datasets from various National Cancer Institute Programs. Also known as the gigapixel pathology image, one image slide contained two tissue smears stained with hematoxylin and eosin, imported in a vendor specific format. The imported image for classification is scanned at the 'high' magnification level in a microscope of 40X. The pixel size for images was approximately 0.25 microns per pixel. Manual marking of a minimum of ten samples from each of the classes i.e., Tumors, Immune, Macrophage, and other cell was performed by a pathologist on all slides. Care was taken to ensure that the samples were as diverse as possible, to account for all possible morphologies of the same cell type across all cases. In total, 1250 annotations were made in all the images, with each annotated cell labelled by a pathologist. The reasons for the comparatively smaller labelled dataset available lie in the large size of the slide, and pathologist availability; the large size makes parsing through difficult and explains the diversity of labelled structures in each image.

## 4.2 Whole-Slide Image (WSI) Nucleus Segmentation

Before nuclear segmentation and extraction can be performed on the WSIs, certain pre-processing steps are carried out to ensure staining uniformity across

images. A representative and generalizable stain vector estimation for each of the stains being used in the image is estimated, to normalize the staining intensities across all the images. In our application, vectors from hematoxylin, eosin and residual stains were estimated from a standard image, selected by the clinician. These vectors are then applied across all images to keep the stain detection parameters uniform across the dataset (Bankhead et al. (2018)).

Due to the variations in staining procedures across the dataset, it is difficult to accurately isolate whole cells on the slide, as the cytoplasmic boundaries are not well-defined due to lack of membrane staining. As heterogeneity in nuclear morphology has been shown to be a good discriminator, only the nucleus is segmented out of each cell in every image for classification purposes, using Qupath (Yuan et al. (2012)). Watershed segmentation is used to obtain separated and contoured nuclei from the hematoxylin color channel of the whole slide image, in a patch-based manner (Bankhead et al. (2018)). The separate hematoxylin channel is obtained by performing color deconvolution to separate out the stains used in the slide (Ruifrok et al. (2001)). It was observed that an average of 200,000 cells were segmented out per image, and morphological and intensity features such as spatial location, eccentricity, circularity, and stain intensities were computed. All pertinent image analysis and nuclear segmentation was performed using Qupath, an open-sourced software platform that can be used for a range of pathological image analysis applications (Bankhead et al. (2017)).

| Class | Tumor (1) | Immune (2) | Other Cells (3) | Macrophage (4) | Total Cells |
|---|---|---|---|---|---|
| Number of Labels | 288 | 349 | 252 | 361 | 1250 |

Table 4.2: Cell type composition of training dataset.

A Random Forest model is a type of ensemble learning method, where the weak predictive power of multiple decision trees is aggregated to produce an accurate result Breiman (2001). Our four-class classifier model was developed with a pathologist labelled dataset of 1250 morphologically diverse cells, each belonging to Tumor, Immune, Macrophage, or other cells (including cell types such as stromal cells). 5-fold cross-validation was used to assess model accuracy and the receiver operator characteristic curve AUC for each class, with proportional representation of all 4 classes ensured. After adjusting for multiple parameters, including the number of decision trees, the accuracy was obtained in the range of 87-91%.

This step allowed for a preparation of a dataset for each cell with its spatial location using global coordinates and class of the cell, which is then used for further downstream analysis. The training, testing, and classification of cells from the WSIs was performed on MATLAB version 2017A.

# S. 5

# Genomic Associations

## 5.1 Association with expression of significantly mutated genes

In addition to associations with survival, we investigated the association between our measurement and gene expression. Gene expression data was acquired for all 335 patients in our sample using TCGA Assembler (Zhu et al. (2014)). Additionally, 42 significantly mutated genes (SMGs) of interest were identified using previous work investigating the genomic differences in SKCM (Akbani et al. (2015)).

Of the 335 patients in our sample, 330 had gene expression data for all genes of interest, while 95 were missing data for all genes of interest. We examined the marginal association between the normalized gene expression data for the 330 patients with complete data and average logit CTIP values. Marginal association was assessed via univariate simple linear regression, carried out separately for each gene. The Wald statistic of the coefficient corresponding to gene expression was used to produce a $p$-value. After correcting for multiple testing using a Bonferroni correction with $\alpha = 0.05$, we found that LRRC37A3 was significantly positively associated with average logit CTIP ($p << 0.0001$). See Table 5.2 below for the full list of genes, significance of associations, and directionality of associations.

| Gene | P-value | Direction of Association |
|---|---|---|
| LRRC37A3 | 3.9e-05 * | Positive |
| B2M | 0.0012 | Negative |
| TP53 | 0.0019 | Positive |
| BRAF | 0.024 | Negative |
| RCAN2 | 0.027 | Positive |
| TMEM216 | 0.032 | Positive |
| ACD | 0.033 | Positive |
| PPIAL4G | 0.043 | Positive |
| TBC1D3B | 0.052 | Positive |
| FAM113B | 0.055 | Negative |
| COL9A2 | 0.057 | Positive |
| SIRPB1 | 0.064 | Negative |
| ZFX | 0.073 | Negative |
| RB1 | 0.087 | Negative |
| DDX3X | 0.1 | Negative |
| RQCD1 | 0.13 | Negative |
| CCDC28A | 0.14 | Negative |
| NDUFB9 | 0.15 | Positive |
| EMG1 | 0.15 | Positive |
| ITGA4 | 0.18 | Positive |
| OXA1L | 0.19 | Positive |
| CTNNB1 | 0.2 | Negative |
| STK19 | 0.27 | Positive |
| MAP2K1 | 0.28 | Negative |
| NRAS | 0.28 | Negative |
| GNAI2 | 0.32 | Positive |
| RAPGEF5 | 0.36 | Positive |
| NF1 | 0.37 | Negative |
| IDH1 | 0.37 | Negative |
| PTEN | 0.4 | Negative |
| ARID2 | 0.42 | Negative |
| RAC1 | 0.42 | Positive |
| CDKN2A | 0.56 | Positive |
| MRPS31 | 0.64 | Positive |
| MSR1 | 0.7 | Negative |
| WDR12 | 0.72 | Negative |
| NOTCH2NL | 0.83 | Positive |
| PPP6C | 0.85 | Positive |
| C3orf71 | 0.89 | Negative |
| RPS27 | 0.9 | Negative |
| FAM58A | 0.95 | Positive |
| KNSTRN | 0.98 | Positive |

Table 5.2: Significantly mutated genes, unadjusted p-values, and direction of association with biopsy level average logit CTIP. Asterisk indicates significance under Bonferroni correction, alpha = 0.05.

## 5.2 Association with expression of immune genes

We also assessed the association between CTIP and genes that are associated with immune activity. Bhattacharya et al. (2018) have collected and classified a list of 1,793 unique genes associated with various aspects of human immune activity. Of these, gene expression data was available for 1,305 genes across the same 330 patients used in the previous gene expression analysis. We assessed the univariate association between biopsy level mean logit CTIP and these genes using Spearman Correlation. The advantage of Spearman correlation as opposed to the more standard Pearson correlation is that the former does not assume a linear relationship between the underlying variables of interest. Such assumptions can be problematic, particularly when there is no strong reason a priori to believe the relationship between the two variables is of a particular form. However, like Pearson correlation Spearman correlation is defined to lie in $[-1, 1]$, with each extreme indicating the same directionality and strength of association as Pearson Correlation. Finally, statistical significance of associations was calculated using the cor.test function of the R programming language (R Core Team (2021)). After applying a Bonferroni correction, we found that 28 genes were significantly associated with IP. For the complete list of genes, see Table 5.2 below.

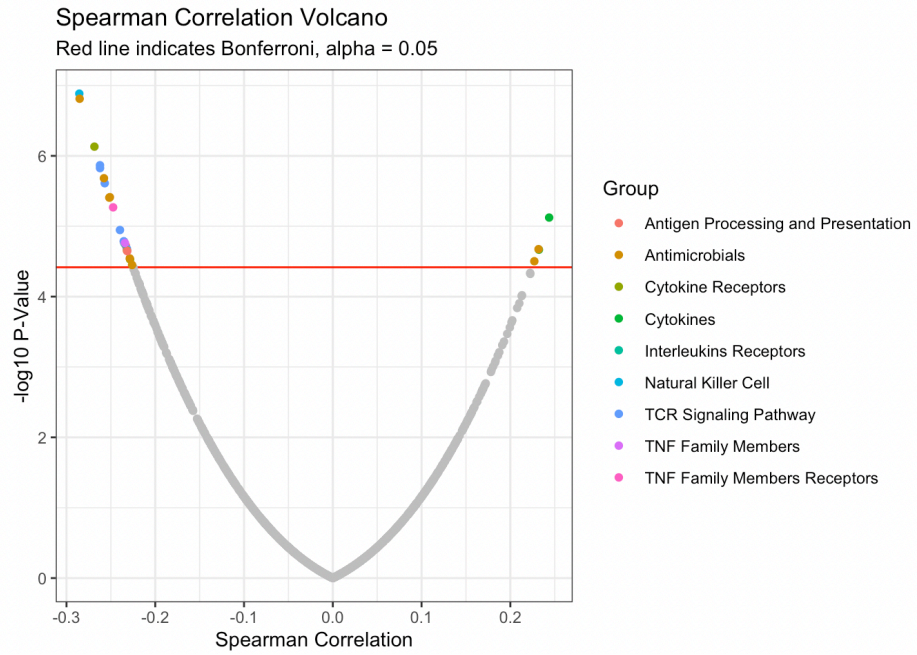



Figure 5.1: $-log_{10}$ p-values plotted against Spearman Correlation. Colors indicate Group as determined by IMMPORT; Red line indicates Bonferroni correction

| Gene | Group | Spearman Correlation | P-Value |
|---|---|---|---|
| CD244 | Natural Killer Cell | -0.29 | « 0.0001 |
| CTSS | Antimicrobials | -0.29 | « 0.0001 |
| CCR8 | Cytokine Receptors | -0.27 | « 0.0001 |
| CD3G | TCR Signaling Pathway | -0.26 | « 0.0001 |
| PTPRC | TCR Signaling Pathway | -0.26 | « 0.0001 |
| TLR8 | Antimicrobials | -0.26 | « 0.0001 |
| ITK | TCR Signaling Pathway | -0.26 | « 0.0001 |
| TLR3 | Antimicrobials | -0.25 | « 0.0001 |
| STAT1 | Antimicrobials | -0.25 | « 0.0001 |
| TNFRSF9 | TNF Family Members Receptors | -0.25 | « 0.0001 |
| EREG | Cytokines | 0.24 | « 0.0001 |
| LCK | TCR Signaling Pathway | -0.24 | $< 0.0001$ |
| PIK3CG | TCR Signaling Pathway | -0.24 | $< 0.0001$ |
| ERAP1 | Antigen Processing and Presentation | -0.24 | $< 0.0001$ |
| TNFSF14 | TNF Family Members | -0.23 | $< 0.0001$ |
| IL6R | Interleukins Receptors | -0.23 | $< 0.0001$ |
| CIITA | Antigen Processing and Presentation | -0.23 | $< 0.0001$ |
| ICOS | TCR Signaling Pathway | -0.23 | $< 0.0001$ |
| DES | Antimicrobials | 0.23 | $< 0.0001$ |
| APLN | Cytokines | 0.23 | $< 0.0001$ |
| IFIH1 | Antimicrobials | -0.23 | $< 0.0001$ |
| RFX5 | Antigen Processing and Presentation | -0.23 | $< 0.0001$ |
| LYZ | Antimicrobials | -0.23 | $< 0.0001$ |
| CXCR6 | Cytokine Receptors | -0.23 | $< 0.0001$ |
| CRABP1 | Antimicrobials | 0.23 | $< 0.0001$ |
| CYBB | Antimicrobials | -0.23 | $< 0.0001$ |

Table 5.2: IMMPORT Genes significantly associated with average logit biopsy level CTIP after Bonferroni correction, alpha = 0.05.

# S. 6

# Association with Deconvolution Data

Using data from TIMER2.0 (Li et al. (2020)), we examined the association between the prevalence of different types of immune cells and biopsy level mean logit CTIP. We ultimately decided to use the MCP-counter algorithm (Becht et al. (2016)) based on the analysis of Sturm et al. (2019), since it was judged to be most effective in detecting the presence and prevalence of the most relevant types of immune cells. We investigated the association of the score of each type of immune cell estimated by MCP-counter with biopsy level mean logit CTIP using Spearman correlation. Significance was assessed using the standard test of statistical significance of Spearman correlation as implemented by the pspearman package.

After applying a Bonferroni correction ($\alpha = 0.05$), we found that six different immune cell scores as computed by MCP-counter were significantly negatively associated with biopsy level mean logit CTIP: CD8+ T cells, B cells, Monocytes, Macrophages, Myeloid Dendritic Cells, and Natural Killer cells. No cell types were significantly positively associated with biopsy level mean logit CTIP after the Bonferroni correction, though the magnitude of the positive association with Cancer Associated Fibroblasts (CAFs) is notable, and while not statistically significant still highly consistent with a truly positive underlying association between biopsy level CTIP and prevalence of CAFs.

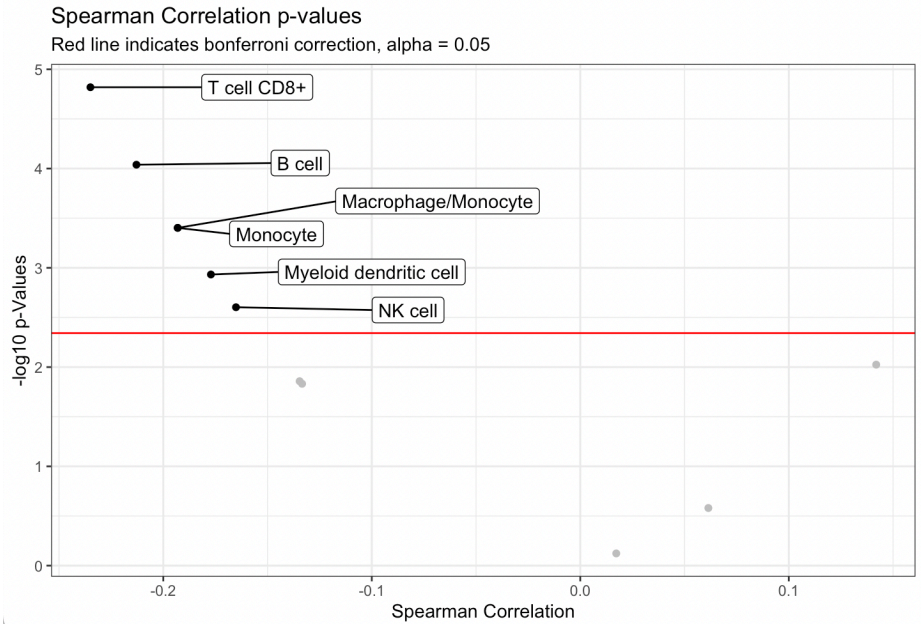



Figure 6.1: Negative log-10 p-values plotted against Spearman Correlation. Colors indicate Group as determined by IMMPORT; Red line indicates Bonferroni correction, alpha = 0.05.

| Cell Type | Spearman Correlation | P-Value |
|---|---|---|
| T cell | -0.1345989 | 0.0138744 |
| T cell CD8+ | -0.2349456 | 0.0000152 |
| cytotoxicity score | -0.1334298 | 0.0147288 |
| NK cell | -0.1651353 | 0.0024940 |
| B cell | -0.2128785 | 0.0000914 |
| Monocyte | -0.1930875 | 0.0003958 |
| Macrophage/Monocyte | -0.1930875 | 0.0003958 |
| Myeloid dendritic cell | -0.1771510 | 0.0011674 |
| Neutrophil | 0.0172481 | 0.7533506 |
| Endothelial cell | 0.0614183 | 0.2628597 |
| Cancer associated fibroblast | 0.1419220 | 0.0094486 |

Table 6.1: IMMPORT Genes significantly associated with average logit biopsy level CTIP after Bonferroni correction, alpha = 0.05.

# S. 7

# SPARTIN R package

This is a brief tutorial on the usage of the SPARTIN R package. It is also available as a vignette in the package itself.

```
library(spatstat)
library(SPARTIN)
library(purrr)
set.seed(10000)
```

The following two functions are purely for visualization. They aren't technically part of the package (yet), but you may find them useful if you decide to use this package. Regardless, we'll be using them for this vignette.

```
CustomHeatmap = CH = function(m, t="", min.v = NA, max.v = NA){
  plot.tib = tibble(r = numeric(0), c = numeric(0), val = numeric(0))
  for(i in 1:nrow(m)){
    plot.tib = bind_rows(plot.tib, tibble(
      r = rep(i, ncol(m)),
      c = 1:ncol(m),
      val = as.numeric(m[i,])))
  }

  plot.tib$c = as.factor(plot.tib$c)

  if(is.na(min.v) || is.na(max.v)){
    min.v = min(plot.tib$val, na.rm = T)
    max.v = max(plot.tib$val, na.rm = T)
  }
  cols = colorRampPalette(c("#0a0722", "#3d0965", "#721a6e", "#a52c60", "#d44842",
                            "#f37819", "#fcb216"))(7) #, "#f1f179"))(8)
```

```
  ggplot(plot.tib) +
    geom_tile(mapping = aes(x = c, y = reorder(r, -r), fill = val)) +
    labs(fill = "") +
    ggtitle(t) +
    scale_fill_gradientn(colors = cols,
                         limits = c(min.v, max.v),
                         # breaks = seq(min.v, max.v, 2),
                         na.value = 'white') +
    theme(axis.title.y=element_blank(),
          axis.text.y=element_blank(),
          axis.ticks.y=element_blank(),
          axis.title.x=element_blank(),
          axis.text.x=element_blank(),
          axis.ticks.x=element_blank(),
          panel.background = element_rect(fill = "white",
                                          colour = "black"),
          panel.border = element_rect(linetype = "solid", fill = NA))
}

# Visualize PPP
vis = function(p, t="", suppWarn = F){
  if(suppWarn){
    suppressWarnings(
      plot(p, cols = c("black", "red"),
           shape = c("circles", "circles"),
           size = 6,
           main = t)
    )
  }else{
    plot(p, cols = c("black", "red"),
         shape = c("circles", "circles"),
         size = 6,
         main = t)
  }
}
```

## 7.1 Simulation and Model Fitting

First, we'll simulate some data.

```
ex_ppp = SimulateData(n1 = 100, n2 = 40, phi = 0.3,
                      winX = 300, winY = 300, r = 30)
```

Breaking down this function call:

- `n1` specifies the number of points of type 1
- `n2` specifies the number of points of type 2
- `phi` specifies the interaction between the points, -1 being the most negative (points of different types tend to "avoid" each other) and 1 being the most positive (points of different types tend to be close together)
- `winX` specifies the horizontal width of the window of simulation
- `winY` specifies the vertical height of the window of simulation
- `r` specifies the radius of interaction

Next, we'll fit a Frequentist version of the model using the `FitHSFreq` function. The "HS" stands for "Hierarchical Strauss," the variant of the model that we'll be using. This is essentially a wrapper around the `spatstat` function `ppm` with certain parameters, since the SPARTIN pipeline uses a special case of the Hierarchical Strauss model. Below, I pass the point process we simulated, and specify that the radius of interaction is 30 units, and that the quadrature used in the fitting should have 3 units of space between each point. As always, there is a tradeoff here: the larger the quadrature (and thus the smaller that `quad.spacing` parameter), the more accurate the fitting will be. However, it will also be more computationally expensive. On the other hand, a small quadrature may lead to quicker fittings, but may also be less accurate. Unfortunately, there isn't a one size fits all solution to this across all use cases. Some situations call for larger quadratures and more precise estimation, while for others a rough approximation will probably be fine.

```
ex_freq = FitHSFreq(ex_ppp, r = 30, quad.spacing = 3)
coef(summary(ex_freq))
```

```
##                 Estimate      S.E.    CI95.lo    CI95.hi Ztest         Zval
## (Intercept) -6.802394763 0.1000000 -6.9983912 -6.6063984   *** -68.02394763
## marks2      -0.936072712 0.4214038 -1.7620089 -0.1101365     *  -2.22132023
## markX1xX2    0.006326801 0.1204110 -0.2296744  0.2423280         0.05254337
```

As expected, the interaction parameter (`markX1xX2`) is fairly close to zero, which makes sense given that we simulated weakly positive interaction.

We can also fit the same model using Bayesian methods. Here's how we would fit a Bayesian version of the same model given above:

```
ex_bayes = FitHSBayes(ex_ppp, r = 30, quad.spacing = 3)
ex_bayes
```

```
## Inference for Bugs model at "/var/folders/jz/pkw0zld53pz3dxvzc5fs4hy00000gn/T//RtmpoGhJGl/mode
```

```
##  1 chains, each with 11000 iterations (first 1000 discarded), n.thin = 5
##  n.sims = 2000 iterations saved
##                mu.vect sd.vect       2.5%        25%        50%        75%
## log.beta.1      -6.809   0.099     -7.008     -6.874     -6.803     -6.742
## log.beta.2      -7.762   0.331     -8.413     -7.986     -7.747     -7.540
## log.gamma        0.006   0.092     -0.178     -0.053      0.006      0.069
## deviance   282260.902   2.368 282258.182 282259.172 282260.279 282261.990
##                 97.5%
## log.beta.1      -6.616
## log.beta.2      -7.124
## log.gamma        0.180
## deviance   282266.853
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 2.8 and DIC = 282263.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

There are a couple of important things to note. Firstly, you might notice that the object returned by the Frequentist fitting function is *very* different from the object returned by the Bayesian fitting function. That's because the Frequentist version returns a `ppm` object defined in the `spatstat` package, whereas the Bayesian version returns an `rjags` object as defined by the `R2jags` package.

Secondly, you might notice that while two of the parameter estimates (`log.gamma`/`markX1xX2` and `log.beta.1`/`(Intercept)`) are virtually identical, the estimate for `marks2` is extremely different from `log.beta.2`. This is because `marks2` is actually the *difference* between the log first order intensity of points of type 1 ($\log(\beta_1)$) and the log first order intensity of points of type 2 ($\log(\beta_2)$), whereas `log.beta.2` is just the value of $\log(\beta_2)$. Sure enough, if you add the estimate of `(Intercept)` to the estimate of `marks2`, you should get something virtually identical to the estimate of `log.beta.2`.

## 7.2 Estimating CTIP

In the paper associated with this package, we define CTIP. For a rigorous definition, I recommend checking the paper. For now, here is how you can use this package to calculate CTIP for a given point process realization:

```
ex_CTIP = CTIP(ex_ppp, r = 30, quad.spacing = 3,
               n.null = 5, n.burn = 1000,
               n.sample = 11000, null.n.burn = 1000, null.n.sample = 11000,
               n.thin = 5)
ex_CTIP
```

```
## [1] "0.503"
```

Again, it's worth breaking down this function call. I won't go into too much detail here, but for more information on how CTIP is actually computed, see the SPARTIN paper.

- `ex_ppp` - This is the point process realization we want to estimate CTIP for.
- `n.null` - In order to compute CTIP, null simulations are performed; this parameter specifies how many should be performed. More simulations yield a more stable and accurate estimate, but naturally take longer.
- `n.burn` - Number of burn-in samples for the model fitting on the true data.
- `n.sample` - Total number of samples to draw when fitting the model on the true data.
- `null.n.burn` - Number of burn-in samples for the model fitting on the simulated data.
- `null.n.sample` - Total number of samples to draw for each null simulation in model fitting.
- `n.thin` - How much to "thin" each of the chains, both in the actual model fitting and the null model fitting

The total number of samples drawn on the true data is given by $\frac{n.sample - n.burn}{n.thin}$, while for the null simulations it's $n.null \cdot \frac{null.n.sample - null.n.burn}{n.thin}$.

## 7.3 Tesselation and Intensity Thresholding

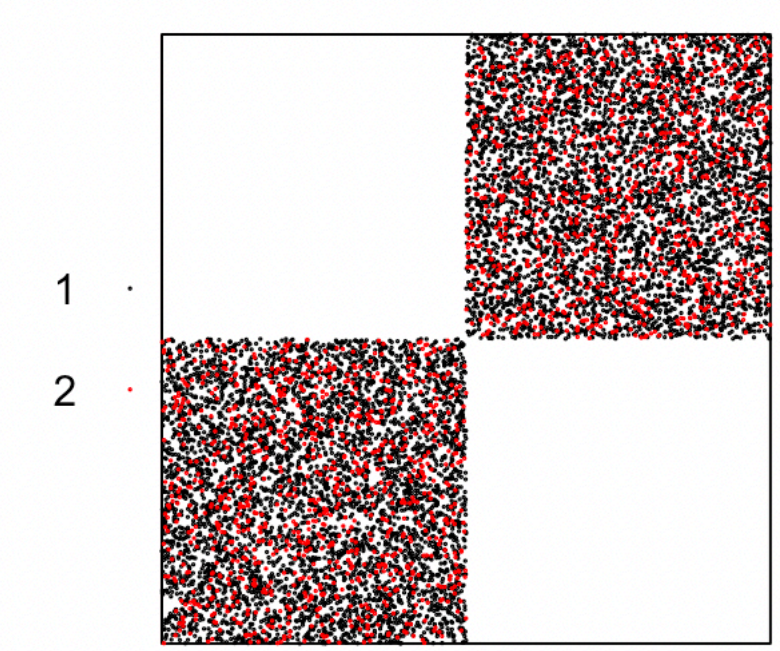
Another important part of the SPARTIN pipeline is the intensity thresholding to remove excess whitespace. First, I'll create a biopsy with a large amount of excess space.

```
# Simulate a biopsy
ex_biopsy_part_1 = SimulateData(n1 = 5000, n2 = 1000, phi = 0.4,
                          winX = 1000, winY = 1000, r = 30)

ex_biopsy_part_2 = SimulateData(n1 = 5000, n2 = 1000, phi = 0.4,
                          winX = 1000, winY = 1000, r = 30)

# Place in a large window to demonstrate intensity thresholding in action
ex_biopsy_excess = ppp(x = c(ex_biopsy_part_1$x, ex_biopsy_part_2$x - 1000),
                       y = c(ex_biopsy_part_1$y, ex_biopsy_part_2$y - 1000),
                       marks = c(ex_biopsy_part_1$marks,
                                 ex_biopsy_part_2$marks),
                       window = owin(xrange = c(-1000, 1000),
                                     yrange = c(-1000, 1000)))
```

```
vis(ex_biopsy_excess)
```

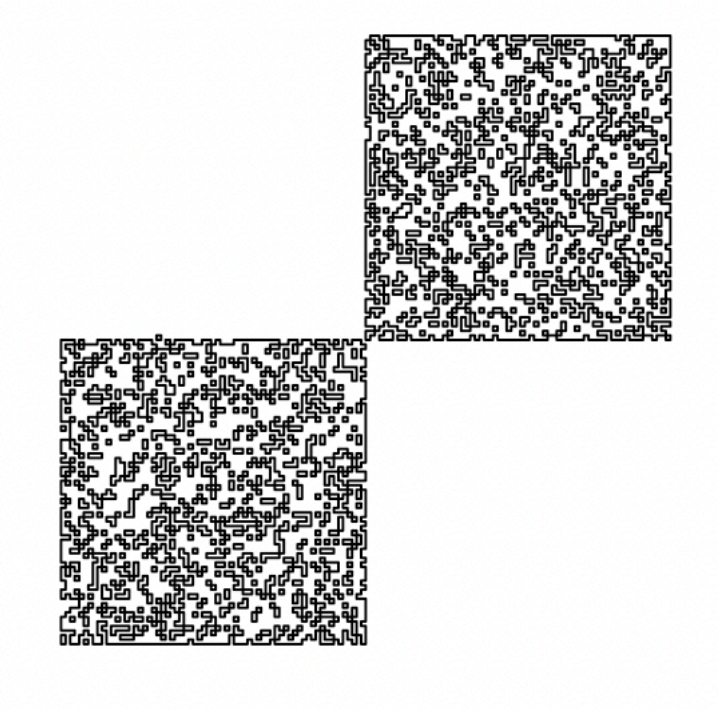



To see what each of the parameters do for this function, it's probably best to directly read the documentation. Here, I'll offer some advice on how to achieve a good tessellation/partition of the subspace. First, I'll start with a bad example: sigma (the bandwidth) is too small and the threshold is too high. Thus, the resulting intensity thresholded window has too much unnecessary white space, and the resulting windows are too"choppy." It also takes longer, and can lead to errors.

```
bad_tessellation_1 = TessellateBiopsy(PPPToTibble(ex_biopsy_excess),
                                      sigma = 1, eps = 15,
                                      threshold = (ex_biopsy_excess$n)/
                                        area.owin(ex_biopsy_excess$window),
                                      clust.size = 75,
                                      max.clust.size = 100)
```

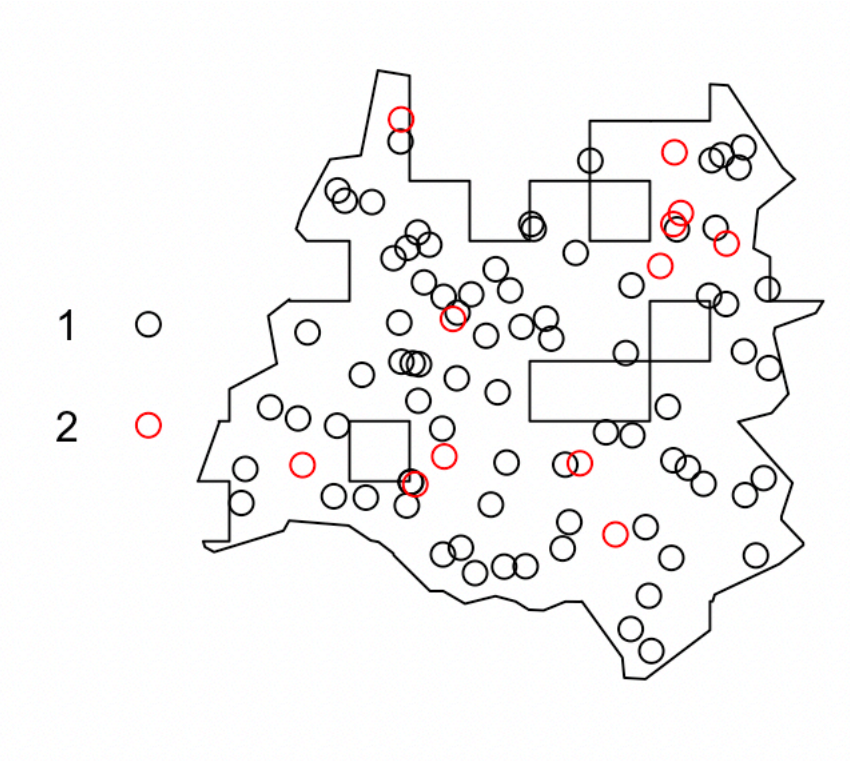
Notice the "checkerboard" pattern of the tessellations:

```
vis(bad_tessellation_1$window)
```



This leads to poorly shaped tiles:

```
vis(bad_tessellation_1$tiles[[1]])
```



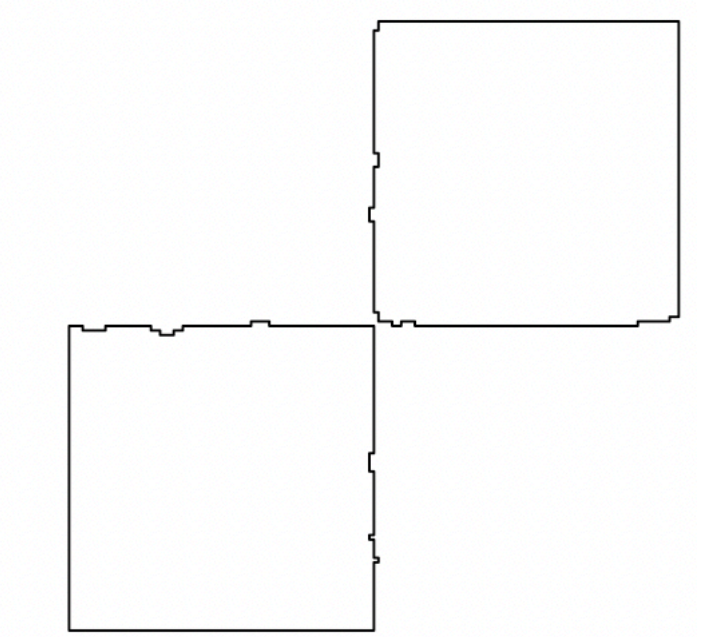

Let's try increasing the value of sigma. Note that all other parameters remain the same.

```
good_tessellation = TessellateBiopsy(PPPToTibble(ex_biopsy_excess),
                                     sigma = 30, eps = 15,
                                     threshold = (ex_biopsy_excess$n)/
                                       area.owin(ex_biopsy_excess$window),
                                     clust.size = 75,
                                     max.clust.size = 100)
```

This looks much better:

```
vis(good_tessellation$window)
```

```
knitr::include_graphics(here("Figures/Package Figs/good_window.png"))
```



Ultimately, these are tuning parameters, and to find good values you may have to experiment a bit. The best advice I can offer is: start with a relatively large value of `sigma` (at least as large as `eps`), a relatively large value of `eps`, and a small `threshold`. Starting with a moderate `eps` is particularly important- if it's too small, the tessellation will not be computationally tractable. Once you have a working tessellation, reduce the values of sigma and eps as necessary, and increase the threshold as necessary. In my experience, the settings that work for one data set of a certain type will work for others; you should only need to go through this process once.

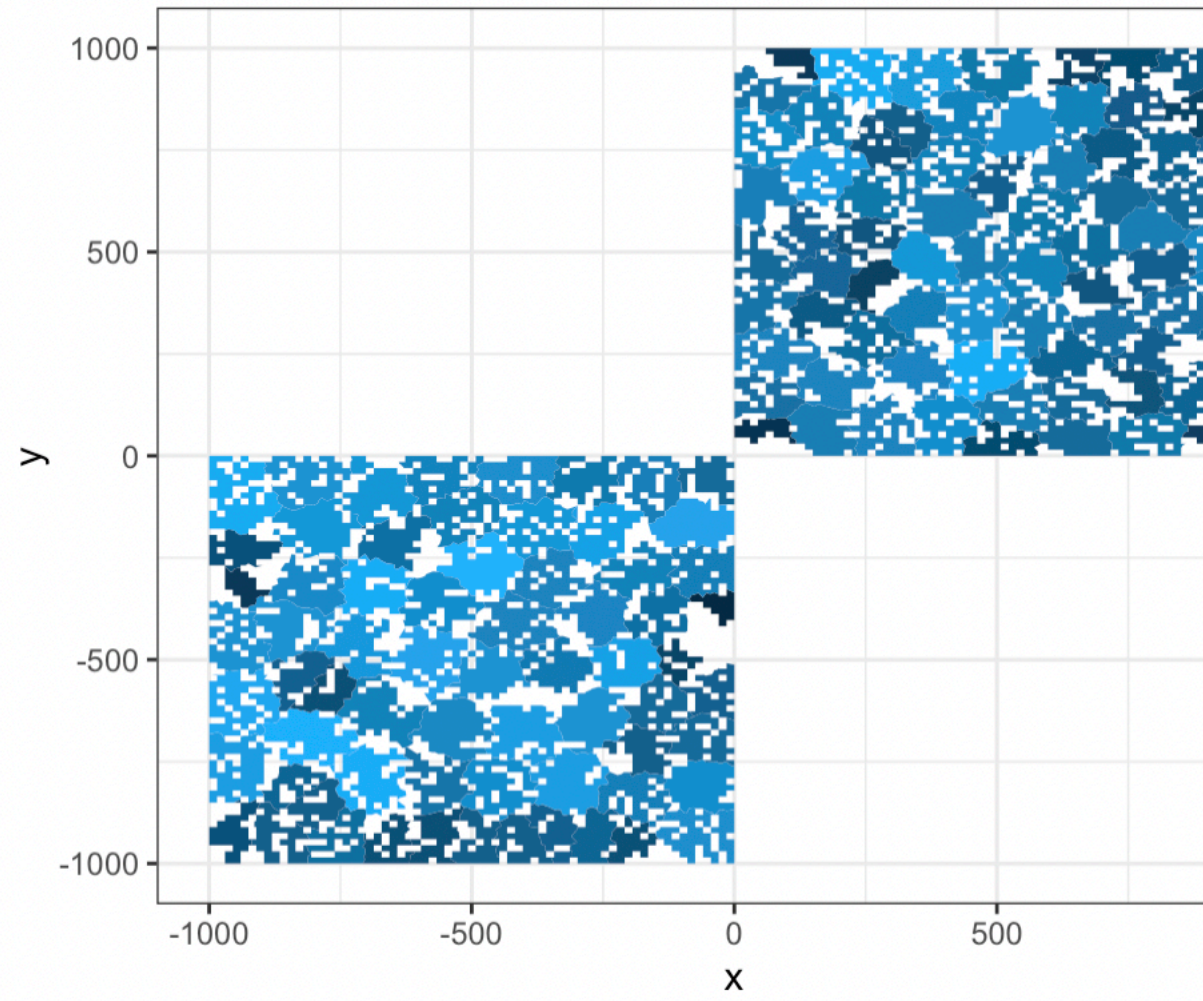
## 7.4 Interactive Visualization

Suppose you want to visualize all of the tiles all together, and a value associated with each tile. To do this, you can simply use the `PlotTessellation` function. First, we need a value to plot for each tile. In this case, I'll use the number of simulated tumor cells:

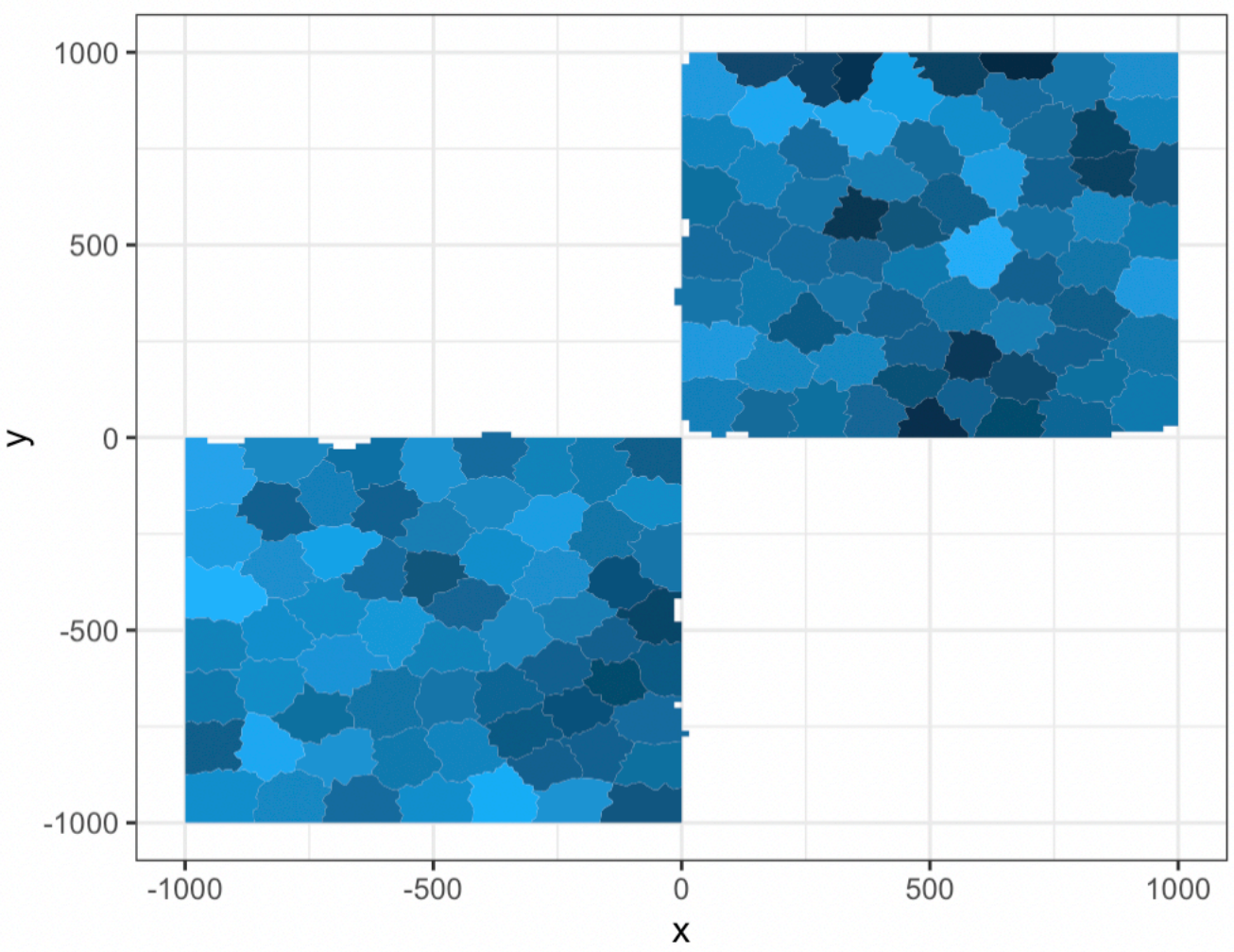
```
# Result is a vector, each entry of which is the number of tumor cells
# in the corresponding tile of the tessellation
bad_tes_n_tum = map_dbl(bad_tessellation_1$tiles, ~ sum(.x$marks == 1))
good_tes_n_tum = map_dbl(good_tessellation$tiles, ~ sum(.x$marks == 1))
```

Then we pass both these things to the `PlotTessellation` function:

```
PlotTessellation(bad_tessellation_1, bad_tes_n_tum, "# Tumor Cells")
```




```
PlotTessellation(good_tessellation, good_tes_n_tum, "# Tumor Cells")
```




However, there are clear limitations to this visualization method. What if you want to zoom in on a particular tile? Unfortunately, there's no easy way to do this in R. However, this can be easily accomplished in the companion web app, and the `ExportToVis` function. This function exports a tessellation as well as associated values so that they can be interactively visualized here: https://nateosher.github.io/SPARTIN.html.

In fact, unlike the static visualization above, it's fairly easy to switch between multiple tile level summaries. First, we create a list of all of the tile level values we want to summarize. Each list entry should have the same set of keys, each of which will be a value we're interested in. I'll keep it simple here and just use three: number of cells, number of tumor cells, and number of immune cells:

```
val_list = map(good_tessellation$tiles, function(t){
  list(
    Cells = t$n,
    Tumor = sum(t$marks == 1),
    Immune = sum(t$marks == 2)
  )
})
```

Then we export it. This command actually writes a `.json` file to the provided path:

```
ExportToVis(good_tessellation, val_list, "path/where/you/want/output")
```

If you upload the resulting `.json` file to the visualization application, it will let you more easily examine multiple measures on the same biopsy, and zoom in on specific tiles to see the configuration of points.

# Bibliography


Akbani, R., Akdemir, K. C., Aksoy, B. A., Albert, M., Ally, A., Amin, S. B., Arachchi, H., Arora, A., Auman, J. T., Ayala, B., et al. (2015). Genomic classification of cutaneous melanoma. *Cell*, 161(7):1681–1696.

Baddeley, A. and Turner, R. (2005). Spatstat: an r package for analyzing spatial point patterns. *Journal of statistical software*, 12:1–42.

Bankhead, P., Fernández, J. A., McArt, D. G., Boyle, D. P., Li, G., Loughrey, M. B., Irwin, G. W., Harkin, D. P., James, J. A., McQuaid, S., et al. (2018). Integrated tumor identification and automated scoring minimizes pathologist involvement and provides new insights to key biomarkers in breast cancer. *Laboratory investigation*, 98(1):15–26.

Bankhead, P., Loughrey, M. B., Fernández, J. A., Dombrowski, Y., McArt, D. G., Dunne, P. D., McQuaid, S., Gray, R. T., Murray, L. J., Coleman, H. G., et al. (2017). Qupath: Open source software for digital pathology image analysis. *Scientific reports*, 7(1):1–7.

Becht, E., Giraldo, N. A., Lacroix, L., Buttard, B., Elarouci, N., Petitprez, F., Selves, J., Laurent-Puig, P., Sautès-Fridman, C., Fridman, W. H., and de Reyniès, A. (2016). Estimating the population abundance of tissue-infiltrating immune and stromal cell populations using gene expression. *Genome Biology*, 17(1):218.

Bhattacharya, S., Dunn, P., Thomas, C. G., Smith, B., Schaefer, H., Chen, J., Hu, Z., Zalocusky, K. A., Shankar, R. D., Shen-Orr, S. S., et al. (2018). Immport, toward repurposing of open access immunological assay data for translational and clinical research. *Scientific data*, 5(1):1–9.

Breiman, L. (2001). Random forests machine learning. 45: 5–32. *View Article PubMed/NCBI Google Scholar*.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).

King, R., Illian, J. B., King, S. E., Nightingale, G. F., and Hendrichsen, D. K. (2012). A bayesian approach to fitting gibbs processes with temporal random effects. *Journal of agricultural, biological, and environmental statistics*, 17(4):601–622.

Kruschke, J. (2014). Doing bayesian data analysis: A tutorial with r, jags, and stan.

Li, T., Fu, J., Zeng, Z., Cohen, D., Li, J., Chen, Q., Li, B., and Liu, X. S. (2020). Timer2.0 for analysis of tumor-infiltrating immune cells. *Nucleic Acids Research*, 48(W1):W509–W514.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ruifrok, A. C., Johnston, D. A., et al. (2001). Quantification of histochemical staining by color deconvolution. *Analytical and quantitative cytology and histology*, 23(4):291–299.

Sturm, G., Finotello, F., Petitprez, F., Zhang, J. D., Baumbach, J., Fridman, W. H., List, M., and Aneichyk, T. (2019). Comprehensive evaluation of transcriptome-based cell-type quantification methods for immuno-oncology. *Bioinformatics*, 35(14):i436–i445.

Su, Y.-S., Yajima, M., Su, M. Y.-S., and SystemRequirements, J. (2015). Package 'r2jags'. *R package version 0.03-08, URL http://CRAN. R-project. org/package= R2jags*.

Yuan, Y., Failmezger, H., Rueda, O. M., Ali, H. R., Gräf, S., Chin, S.-F., Schwarz, R. F., Curtis, C., Dunning, M. J., Bardwell, H., et al. (2012). Quantitative image analysis of cellular heterogeneity in breast tumors complements genomic profiling. *Science translational medicine*, 4(157):157ra143–157ra143.

Zhu, Y., Qiu, P., and Ji, Y. (2014). Tcga-assembler: open-source software for retrieving and processing tcga data. *Nature methods*, 11(6):599–600.