Introduction to Robust Spatial Extent Inference using the pbj Package

Simon N. Vandekar

2019-07-17

Acknowledgements

I am grateful to Shawn Garbett for his help in the preparation and debugging of the `pbj` package and the code and theory for this vignette, as well as other troubling issues I have not yet experienced that he is likely to help me with.

The PBJ hypothesis testing procedures

Spatial extent inference (SEI) is a widely used tool in neuroimaging to perform group level analysis, where neuroimaging data are modeled as a function of subject level covariates such as age, sex, diagnosis, and/or other covariates. The parametric bootstrap joint (PBJ) testing procedure and the semi-PBJ (sPBJ) are methods that we recently proposed to perform SEI that relax the assumptions of gaussian random field (GRF) and permutation based methods. For a detailed description of these procedures please see (Vandekar, Satterthwaite, Xia, et al. 2018).

The PBJ and sPBJ procedures conceptualize a group level analysis as the second level in a multilevel model; both allow the user to specify subject specific weights that can be defined as images (e.g. varcopes from fsl) or as a scalar value for each subject. An optimal decision for the weight for subject i at voxel v is the inverse of subject i's variance at location v, call it $\sigma_i^{-2}(v)$. The PBJ procedure assumes that the weights the user passes are exactly equal to $\sigma_i^{-2}(v)$. However, most of the time, $\sigma_i^{-2}(v)$ is unknown. As a solution, the sPBJ procedure uses a robust "sandwich" covariance estimator that provides consistent estimates of the standard errors, even if the estimates of $\sigma_i^{-2}(v)$ are misspecified.

The sPBJ SEI procedure is appropriate for any type of group level analysis and we demonstrated that it is robust at cluster forming thresholds (CFTs) where GRF-based methods fail to control the FWER.

Overview of this vignette

To demonstrate how to use the pbj package, we use data from Maumet et al. (2016) to perform a meta-analysis of statistical maps from 21 pain studies (Maumet et al. 2016; Gorgolewski et al. 2015). This vignette will take you through the process of computing a robust statistical meta-analytic map from the results of the 21 pain studies, performing inference on the maps using the PBJ and sPBJ procedures, and saving and visualizing the results. Note, the `pbj` analysis package was designed for group level models where the input data can be subject or study level data. The tools in this package assume you have fully processed study (or subject) level imaging data. Preprocessing can be done using other tools in `R` (Muschelli et al. 2018).

Computing statMaps

loading in the data

The first step is to gather the files we need to perform the analysis. This is the hardest part; passing the necessary files to the `lmPBJ` function produces a `statMap` object which retains all of the objects we need to

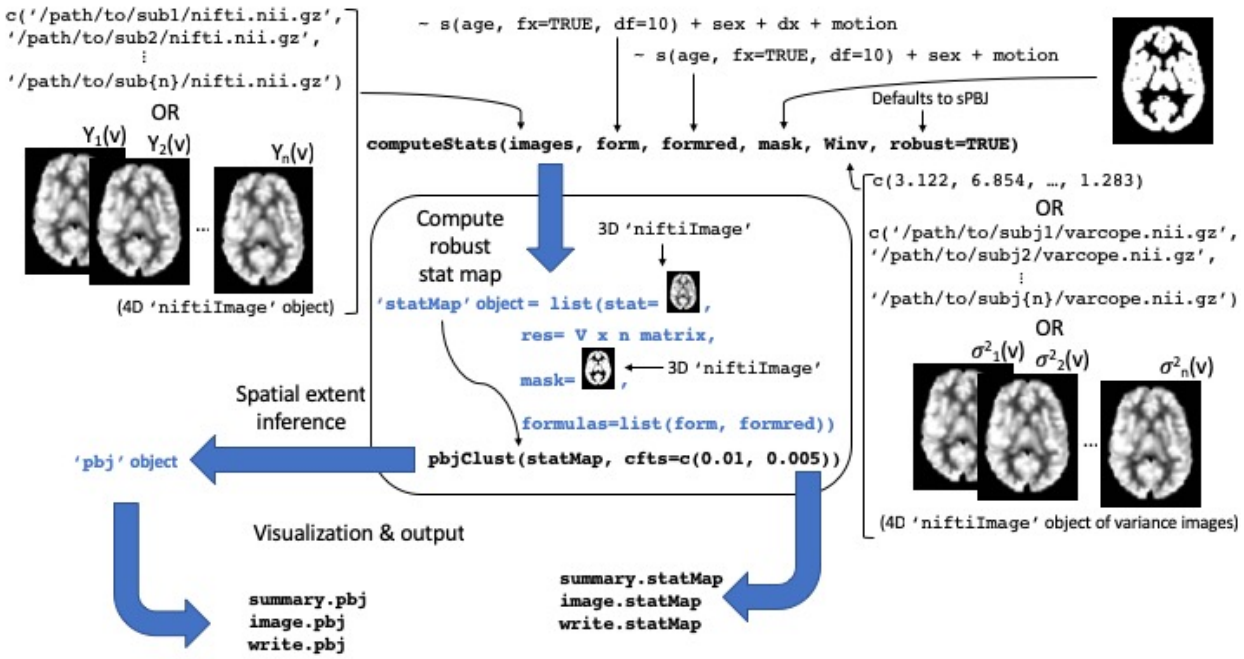

Figure 1: Schematic of pbj analysis functions.

perform SEI and visualization. First, we install the packages we need. The pain21 package includes curated data from the "21 Pain Studies" Neurovault repository (Maumet et al. 2016).

INSTALLATION

Use these commands:

```
install.packages('devtools')
devtools::install_github('simonvandekar/pain21')
devtools::install_github('simonvandekar/pbj')
```

The `pain21` function from the `pain21` package creates a data frame with file paths for the imaging data that we need to perform the analysis. The following images are required to perform the analysis:

- The `mask` image is a binary image indicating which voxels should be included in the analysis.
- The character vector of `images` is the contrast image from each study included in the meta-analysis.
- The character vector of `varimages` contains the voxel level estimates of variance from each study.

The `template` is the MNI 152 template and is not necessary to compute the statistical map or perform SEI, but it is handy to include here for visualization in later steps.

```
library(pain21)
pain = pain21()
pain$mask
#> [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/pain21/pain21/mask.nii.gz"
dirname(pain$data$images[1])
#> [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/pain21/pain21"
head(basename(pain$data$images))
#> [1] "contrast_pain_01.nii.gz" "contrast_pain_02.nii.gz"
#> [3] "contrast_pain_03.nii.gz" "contrast_pain_04.nii.gz"
```

```
#> [5] "contrast_pain_05.nii.gz" "contrast_pain_06.nii.gz"
head(basename(pain$data$varimages))
#> [1] "var_pain_01.nii.gz" "var_pain_02.nii.gz" "var_pain_03.nii.gz"
#> [4] "var_pain_04.nii.gz" "var_pain_05.nii.gz" "var_pain_06.nii.gz"
basename(pain$template)
#> [1] "MNI152_T1_2mm_brain.nii.gz"
```

Computing the statistical map using voxel-wise weights

These data can be passed as arguments to the `lmPBJ` function. Since this is a meta-analysis we are performing a one sample t-test, so the full model includes only the intercept and the reduced model is `NULL`. There are many options with how we pass arguments and analyze the data. As a first pass, we'll use the inverse of the variance images as weights in the regression. Here, we will specify the outdir argument, which will save the output as nifti images so that the large objects stored in the statMap object are not retained in memory. Using voxel level weights can take a little while because the computation must be performed separately at each voxel. `lmPBJ` defaults to using a robust "sandwich" covariance estimate (utilized by the sPBJ procedure); this can be changed by setting `robust=FALSE`. Because we specified the output directory, nifti images of the results are stored in `outdir` and `statmap` includes only character vectors and formulas that point to the files in `outdir`. The output directory and all parent directories are created automatically by `lmPBJ`.

```
library(pbj)
# setup model equations
form = ~ 1
formred = NULL
outdir = tempfile()
dir.create(outdir)
```

```
comptime = system.time(
  statmap <- lmPBJ(pain$data$images, form = form,
                           formred, pain$mask,
                           data = pain$data,
                           Winv = pain$data$varimages, outdir=outdir)
  )
#> Weights are voxel-wise.
#> Running voxel-wise weighted linear models.
#> Getting voxel-wise hat values.
#> Getting voxel-wise residuals for covariate and outcome vectors.
#> Computing robust stat image.
#> Writing output images.
#> Writing sqrtSigma 4d image.

statmap
#>
#> Formula: ~1
#>
#> Contents:
#>   Stat:       '/var/folders/h7/d7np_ycn1zs6d9msn4y3r3vr0000gn/T//RtmpV5QeI5/file184059043649/stat.ni
#>   Coef:       matrix[1 x 259353]
#>   Sqrt Sigma: '/var/folders/h7/d7np_ycn1zs6d9msn4y3r3vr0000gn/T//RtmpV5QeI5/file184059043649/sqrtSig
#>   Mask:       '/Library/Frameworks/R.framework/Versions/3.5/Resources/library/pain21/pain21/mask.nii
#>   Robust:     TRUE
```

The summary output gives us an idea of the data we are working with. The `lmPBJ` function returns a `statMap` object, which contains the following fields:

- The `stat` object is a character pointing to the statistical nifti image
- The `coef` object is a character pointing to the 4d coefficient image
- The `sqrtSigma` object is a character pointing to a 4d covariance nifti image
- The `mask` object gives the directory to the mask file passed as input
- Because we did not specify a template file it does not exist in the output

Computing the statistical map with study-wise (subject-wise) weights

Often, public data sets only include standardized statistical maps; this makes meta-analysis using conventional methods (e.g. mixed effects models) difficult or impossible because the study level variances aren't known. However, the sPBJ procedure is still valid in large samples because it uses robust standard errors, so any weights can be used and the standard errors are still consistent, as long there is an adequate number of studies (or subjects) used in the analysis. The best study level weight is one that is a good estimate of the variance for the statistical map for that study. However, let's see how inference changes if we don't have the variance images, but assume instead that the variance of the estimate from each study is proportional to the square root of the sample size. The details are not critical to understanding how to use the package, but feel free to see below for more information.

Let's run this analysis again, instead on scaled versions of the statistical maps. Instead of voxel-wise weights we'll use weights are that proportional to the sample size for each study. Because we are now going to pass statistical maps to `lmPBJ`, we will compute these maps in `R` and pass `niftiImage` objects as arguments to `lmPBJ`, instead of character vectors. Note that we are passing the weights using the `W=` argument this time because the weights are assumed to be approximately inversely proportional to the variance. We include the `template` argument here for downstream aesthetics.

```
library(pbj)
# formulas can be passed as strings
form = '~ 1'
formred = NULL

library(RNifti) # NIfTI IO
library(abind)  # to concatenate 4d arrays
# read in contrast images
images = readNifti(pain$data$images)
# read in variance images
varimages = readNifti(pain$data$varimages)

# images gets the statistical images scaled by sample size
images = lapply(1:length(images), function(ind){
  # get stat image
  out = images[[ind]];
  # scale out variance and sample size
  out[ varimages[[ind]]!=0] =  out[ varimages[[ind]]!=0 ] / sqrt(varimages[[ind]][ varimages[[ind]]!=0]
  out})

# don't need these anymore
rm(varimages)

# create 4d nifti image of study level statistical maps
images = updateNifti(do.call(abind, c(images, along=4)), readNifti(pain$template))

# third level analysis of statistical maps
comptimen = system.time(statmapn <- lmPBJ(images, form, formred, pain$mask,
                                          template=pain$template, data = pain$data, W = pain$data
```

```
#> Performing voxel regression.
#> Computing hat values.
#> Getting residuals
#> Computing robust stat image.

statmapn
#>
#> Formula: ~ 1
#>
#> Contents:
#>   Stat:       matrix[1 x 259353]
#>   Coef:       matrix[1 x 259353]
#>   Sqrt Sigma: matrix[259353 x 21]
#>   Mask:       nifti[91 x 109 x 91] Pixel dimensions: 2mm x 2mm x 2s
#>   Template:   '/Library/Frameworks/R.framework/Versions/3.5/Resources/library/pain21/pain21/MNI152_T
#>   Robust:     TRUE
```

The statistical map with study level weights has a much faster compute time than the voxel-wise weights because the voxel-wise weights require inverting a matrix at each location in the mask, whereas the study level weights only need to invert one matrix. In our paper, for group level analyses of subject level data, motion was a very effective estimate of subject level variance (Vandekar, Satterthwaite, Xia, et al. 2018). We will see here that our sample size approximation leads to reductions in power.

The `statMap` object has a slightly different output here because we did not specify an output directory.

- The `stat` object is V-vector, where V is the number of nonzero voxels in the mask, indicating the statistical values within the mask
- The `coef` object is a $m_1 \times V$ matrix, where m_1 is the number of extra parameters in the full model, giving the parameter estimates
- The `sqrtSigma` object is a $V \times n$ matrix that is a square root of the covariance matrix of the test-statistics
- The `mask` is a niftiImage object of the mask file
- The `template` object is a character of the template file

`statMap` includes other things that are important to keep track of, such as the formulas used in the call to `lmPBJ`, whether the robust covariance was used, the degrees of freedom, and residual degrees of freedom. By default, for 1-dimensional parameter images (i.e. t-tests) a Z-statistic image is return for the `stat` object. This is encoded within the `pbj` package by setting `statmap$df=0`.

Although, the `pbj` procedures do not make assumptions about the distribution of the errors we still transform the T-statistics to Z-statistics and the F-statistics to Chi-square statistics (for details see Vandekar, Satterthwaite, Rosen, et al. 2018). We have found that this improves small sample performance in simulations.

Visualizing and saving `statMap` objects

At this point we may want to view the statistical images to see what the results look like at a few different CFTs. We can do this using `statMap` method for the `image` function. The default threshold for visualization is 2.32, which is a one-sided p-value of approximately 0.01.

```
#image(statmap)
image(statmapn)
```

We can see from the maps that using voxel-level weights appears to be more powerful: there are more extreme values. Because we did not specify a template image for the first analysis, image defaults to plotting on the mask image. If no mask image is passed then `pbjSEI` will error.

It is also possible to display the lightbox for other orientations:

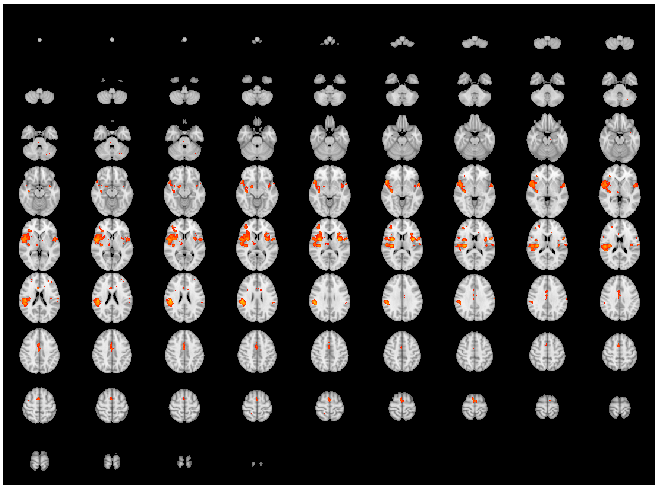

Figure 2: A comparison of using voxel-wise level standard error estimates for parameter estimates versus using study-level standard error estimates based on sample size.

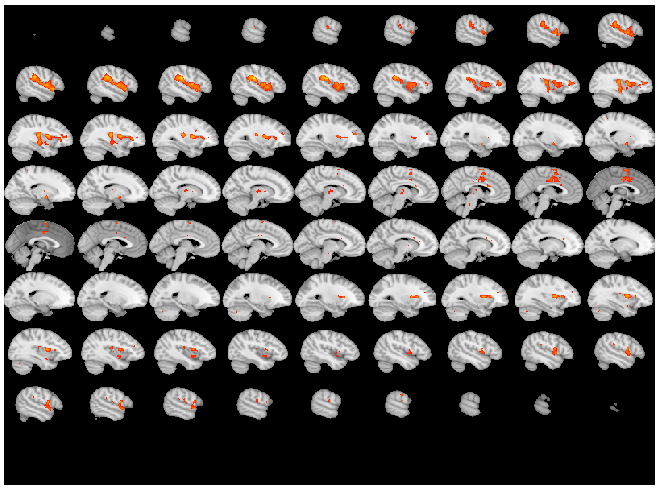

Figure 3: Other image views.

```
#image(statmap, plane='coronal')
image(statmapn, plane='sagittal')
```

the function `write.statMap` can be used to write the output to a directory for later use (e.g. separate visualization or to upload to a public access database).

```
resdir = tempfile()
dir.create(resdir)
write.statMap(statmapn, outdir=resdir)
#> Writing output images.
#> Writing sqrtSigma 4d image.
#> $stat
#> [1] "/var/folders/h7/d7np_ycn1zs6d9msn4y3r3vr0000gn/T//RtmpV5QeI5/file1840554ef020/stat.nii.gz"
#>
#> $coef
#> [1] "/var/folders/h7/d7np_ycn1zs6d9msn4y3r3vr0000gn/T//RtmpV5QeI5/file1840554ef020/coef.nii.gz"
```

```
#>
#> $sqrtSigma
#> [1] "/var/folders/h7/d7np_ycn1zs6d9msn4y3r3vr0000gn/T//RtmpV5QeI5/file1840554ef020/sqrtSigma.nii.gz"
```

Performing SEI on the `statMap` objects

The next step is to perform spatial extent inference to see how likely our results are to have occurred under the global null hypothesis that the image is not associated with the covariate. In this case, the global null corresponds to the hypothesis that the mean of the study-level images is equal to zero.

Performing SEI using `pbjSEI`

The `statMap` object includes the statistical map (`statmap$stat`) and a matrix that is proportional to an estimate of a square root of the covariance matrix of the statistical map (`statmap$sqrtSigma`). The `sqrtSigma` object can be use to estimate the distribution of the largest cluster size under the null using a parametric bootstrap procedure (Vandekar, Satterthwaite, Xia, et al. 2018). This distribution is then used to compute p-values for each cluster.

This step is easy to evaluate because the `statMap` object contains all of the objects necessary to perform SEI on the statistical map. The only thing we need to do is specify the CFTs. We have the voxel level map, which we might choose to use a more stringent threshold on to get finer anatomical details, so we might lean towards a CFT of 0.005. For the study level map we've lost some power with our approximation, so we get anatomically distinct clusters at a lower threshold, so we might lean towards a CFT of 0.01. The default is to use multiple CFTs, `cfts=c(0.01, 0.005)`, which will run the sPBJ procedure at both thresholds.

Under the hood, the clustering procedure operates using a large sample Chi-square approximation, even for t-tests. It does this by squaring the To be consistent with other neuroimaging software, the CFT for t-tests assumes a 1-tailed test. We do this by multiplying the CFT by 2 in the case that a t-test was performed, which is encoded in the `statMap` by `statMap$df=0`.

Because the PBJ and sPBJ procedures use a bootstrap, the `nboot` argument can be used to control the number of bootstrap samples used. For this example we use 200 samples, but for a publication a few thousand is more appropriate to reduce the error in the computational approximation. `pbjSEI` keeps track of progress while it's running (not shown here). We again use `system.time` here to track computing times.

```
pbjcomptime  = system.time(sPBJ  <- pbjSEI(statmap,  nboot=200))
pbjcomptimen = system.time(sPBJn <- pbjSEI(statmapn, nboot=200))
```

The bootstrap time doesn't differ between voxel-wise or study/subject level weights.

```
print(c('voxel weights'=pbjcomptime[3], 'study weights'=pbjcomptimen[3]))
#> voxel weights.elapsed study weights.elapsed
#>                51.448                45.294
```

The `pbjSEI` function returns a `pbj` object which contains information about which clusters are large enough to be unlikely to have occurred by chance under the global null. This object is a list containing the following things:

- `stat` This is the statistical map from the `statMap` object without changes
- `template` This is the template you passed to the `statMap` object
- `mask` This is the mask image from the `statMap` object without changes
- `cft*` This is a list containing the following:
  - `pvalues` This is a vector of the SEI p-values, ordered relative to space, not value
  - `clustmap` This is a `niftiImage` object of cluster indices where each index in `clustmap` corresponds to that particular index of the vector of SEI p-values

– `pmap` This is a `niftiImage` object of `-log10(p)` where `p` is the SEI p-value

Visualizing and saving pbj objects

`pbj` objects can be visualized in the same way as the `statMap` object. The image function takes a desired rejection threshold `alpha` and only plots clusters with cluster-wise p-values below that threshold.

If multiple CFTs were used it produces the images for each of those thresholds.

```
image(sPBJn, alpha = 0.2)
```

The `write.pbj` function can be used to export the the images contained in the `pbj` object and save summary results in a .csv file. The cluster and p-value maps are written for every CFT in a standardized format `pbj_sei_clust_cft*.nii.gz` and `pbj_sei_log10p_cft*.nii.gz`, respectively.

```
write.pbj(sPBJn, resdir)
```

```
tab = read.table(file.path(resdir, 'sei_table_cft0.01.csv'), check.names = FALSE, sep = ',', header=TRU
head(tab)
#>   Index Adjusted p-value Signed log10(p-value) Volume (mm)   Centroid
#> 1     6        0.1641791             0.7846821       23112 44, -4, 11
#> 2    22        0.5024876             0.2988747        2304  -55, 4, 6
#> 3    24        0.5174129             0.2861627        2144 -35, 9, 12
#> 4     4        0.5671642             0.2462912        1712  2, -1, 42
#> 5     1        0.6517413             0.1859248        1104   1, 3, 64
#> 6    31        0.7810945             0.1072964         448 -42, 4, -8
```

Simulation tools

Tools to run simulations as in (Vandekar, Satterthwaite, Xia, et al. 2018) are in progress.

Appendix

Glossary of acronyms

- CFT - cluster forming threshold. The threshold chosen to binarize the statistical map for spatial extent hypothesis testing.
- PBJ - parametric bootstrap joint (testing procedure).
- sPBJ - semiparametric bootstrap joint (testing procedure). Robust to variance misspecification.
- SEI - spatial extent inference. Using thresholded contiguous cluster extents to perform inference on a statistical map.

Technical details for study level weights

Because fMRI data are in arbitrary units, one parameter of interest may be a standardized parameter estimate. For each study $i = 1, \ldots, 21$

$$\mathbb{E}Z_i = \mathbb{E}\left\{\left(\frac{\hat{\beta}_i}{\hat{\sigma}_{yi}}\right) \times (X_{ri}^T X_{ri})^{1/2}\right\} \approx \left(\frac{\beta}{\sigma_{yi}}\right) \times \mathbb{E}\left\{(X_{ri}^T X_{ri})^{1/2}\right\} \approx \left(\frac{\beta}{\sigma_{yi}}\right) \times \sqrt{n_i},$$

where σ_{yi}^2 is the variance for the subjects y from study i, β is the unknown parameter, and X_{ri} is the vector for the covariate of interest (pain scale for study i). The first approximation is because the function in the

expectation is nonlinear; also, because scaling X_{ri} differently for each study proportionally changes the scale of $\hat{\beta}_i$, so we assume that, across data sets, the scaling of X_{ri} is constant so that the expectation $\mathbb{E}\{\hat{\beta}_i\}$ does not depend on i, and the scale of $\mathbb{E}\left\{(X_{ri}^T X_{ri})^{1/2}\right\} = O(\sqrt{n_i})$ only depends on the sample size. Because we are assuming σ_{yi} are not provided we further assume $\sigma_{yi} = \sigma_y$.

We can obtain estimates of $\frac{\beta}{\sigma_y}$ from

$$\mathbb{E} n_i^{-1/2} Z_i \approx \left(\frac{\beta}{\sigma_{yi}}\right),$$

and then weight with n_i since

$$\mathrm{Var}(n_i^{-1/2} Z_i) = n_i^{-1}.$$

Back to "Computing the statistical map with study-wise (subject-wise) weights"

References

Gorgolewski, Krzysztof J., Gael Varoquaux, Gabriel Rivera, Yannick Schwarz, Satrajit S. Ghosh, Camille Maumet, Vanessa V. Sochat, et al. 2015. "NeuroVault.Org: A Web-Based Repository for Collecting and Sharing Unthresholded Statistical Maps of the Human Brain." *Frontiers in Neuroinformatics* 9. https://doi.org/10.3389/fninf.2015.00008.

Maumet, Camille, Tibor Auer, Alexander Bowring, Gang Chen, Samir Das, Guillaume Flandin, Satrajit Ghosh, et al. 2016. "Sharing Brain Mapping Statistical Results with the Neuroimaging Data Model." *Scientific Data* 3 (December). https://doi.org/10.1038/sdata.2016.102.

Muschelli, John, Adrian Gherman, Jean-Philippe Fortin, Brian Avants, Brandon Whitcher, Jonathan D. Clayden, Brian S. Caffo, and Ciprian M. Crainiceanu. 2018. "Neuroconductor: An R Platform for Medical Imaging Analysis." *Biostatistics*. https://doi.org/10.1093/biostatistics/kxx068.

Vandekar, Simon N., Theodore D. Satterthwaite, Adon Rosen, Rastko Ciric, David R. Roalf, Kosha Ruparel, Ruben C. Gur, Raquel E. Gur, and Russell T. Shinohara. 2018. "Faster Family-Wise Error Control for Neuroimaging with a Parametric Bootstrap." *Biostatistics* 19 (4): 497–513. https://doi.org/10.1093/biostatistics/kxx051.

Vandekar, Simon N., Theodore D. Satterthwaite, Cedric H. Xia, Kosha Ruparel, Ruben C. Gur, Raquel E. Gur, and Russell T. Shinohara. 2018. "Robust Spatial Extent Inference with a Semiparametric Bootstrap Joint Testing Procedure." *arXiv:1808.07449 [Stat]*, August. http://arxiv.org/abs/1808.07449.