User Guide: Science-Wise False Discovery Rate

Nathan (Nat) Goodman

June 1, 2017

Contents

This user guide explains how to install and run the scripts in my SWFDR GitHub repository and briefly describes the rest of the distribution. The base script is `swfdr_base.R`, a simple implementation that uses base R capabilities only. Other scripts extend the base implementation by providing solutions to some exercises for the reader.

`swfdr_base.R`

This script reimplements the core idea in David Colquhoun's fascinating paper, "An investigation of the false discovery rate and the misinterpretation of p-values" and further discussed in Felix Schönbrodt's blog post, "What's the probability that a significant p-value indicates a true effect?" and related ShinyApp. The term *science-wise false discovery rate* is from Jager and Leek's paper, "An estimate of the science-wise false discovery rate and application to the top medical literature". John Ioannidis's landmark paper, "Why most published research findings are false", is the origin of it all.

The *false discovery rate* (*FDR*) is the probability that a significant p-value indicates a false positive, or equivalently, the proportion of significant p-values that correspond to results without a real effect. The complement, *positive predictive value (PPV=1-FDR)* is the probability that a significant p-value indicates a true positive, or equivalently, the proportion of significant p-values that correspond to results with real effects.

The script produces graphs of FDR for a range of parameter values. The user can easily change parameters and rerun the program.

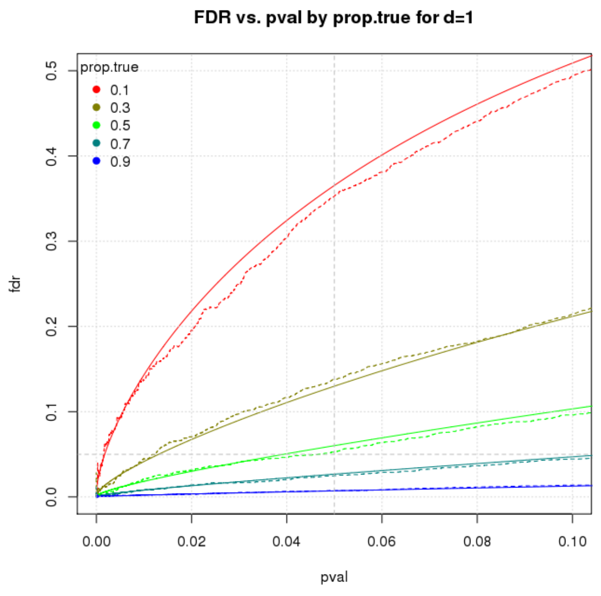
Installation and Usage

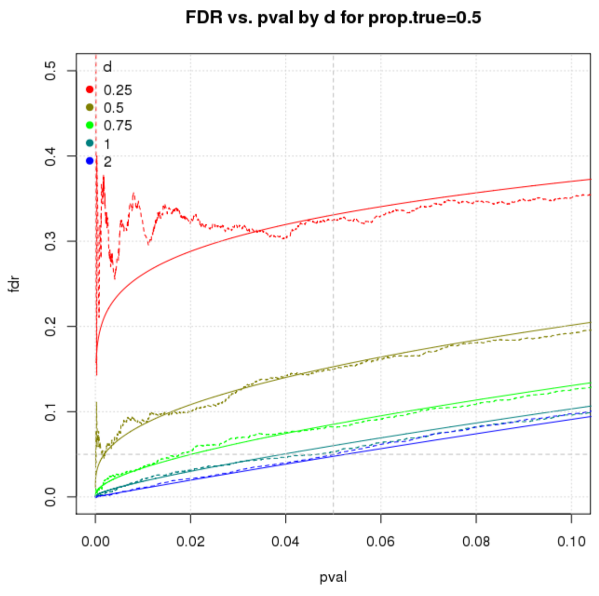
The simplest way to get the software is to download the entire repository. The software is in the `script` subdirectory. At present, the only available script is `swfdr_base.R`. This program uses base R capabilities only and will run “out of the box” on any (reasonably modern) R installation.

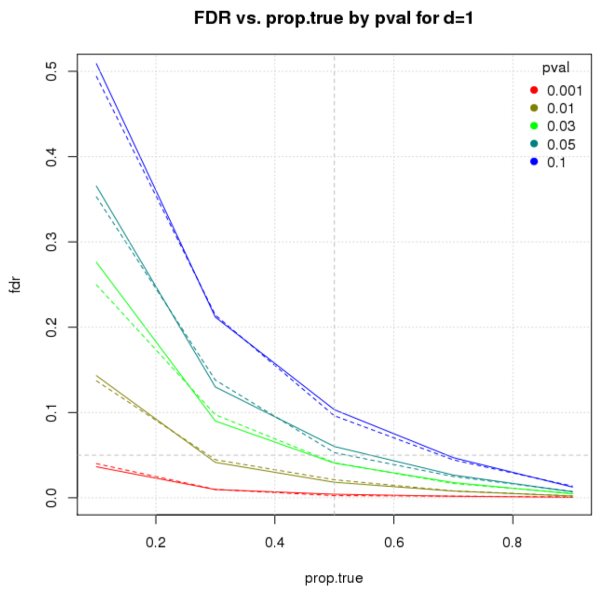
The recommended way to run `swfdr_base.R` is to source the program into your R session and run the statement `run();` The code below illustrates the default process and some variants.

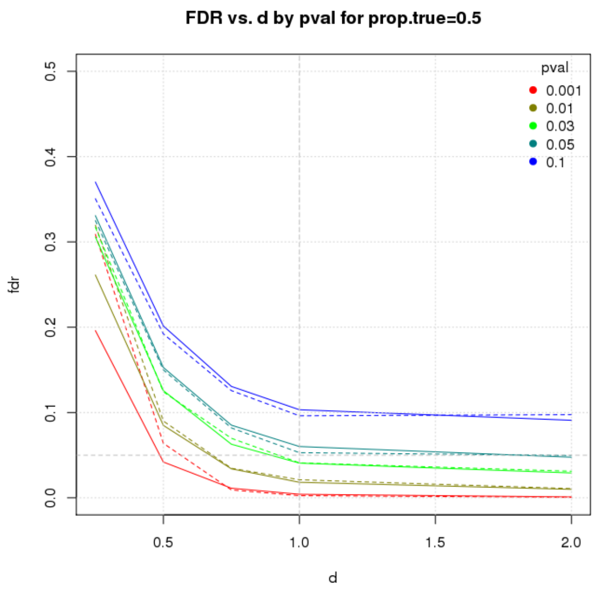
```
# this code block assumes your working directory is the root of the repository

source("script/swfdr_base.R");
# run default process
run();

# run default process and save results in directories data/guide01 and figure/guide01
run(save=T,datadir='data/guide01',figdir='figure/guide01');

# reduce runtime by reducing number of simulation runs and simulated cases
run(m=1e3,d=c(0.25,0.5,1),prop.true=c(0.3,0.5,0.8));

# specify power directly and let program adjust effect size
run(m=1e3,pwr=c(0.1,0.3,0.8),prop.true=c(0.3,0.5,0.8));

# skip computation by loading saved results and replotting graphs
loadit();
doplot();

# load saved results and do something completely different
# for example, plot distribution of effect size error for small effects with significant p-values
loadit();
with(subset(sim,subset=(d==0.25&d.true&pval<=0.05)),hist(diff-d));
```

The default process performs 2.5×10^5 simulations (taking about 3 minutes on my small Linux server) and produces four graphs similar to the ones below (in directory `figure/swfdr_base`). The other examples are much faster. The plots for the three `run` examples and the final histogram example are in directories `figure/guide01`, etc. The plots for the `loadit(); doplot();` example are identical to the default process.









The notation is

- solid lines show theoretical results; dashed lines are empirical results from the simulation
- *fdr.* false discovery rate
- *pval.* p-value cutoff for significance
- *prop.true.* proportion of simulated cases that have a real effect
- *d.* standardized effect size, aka *Cohen's d*

The user can change simulation parameters and control program operation by providing new values to `run()` as illustrated in the example code block above. The available parameters are

parameter	meaning	default
prop.true	fraction of cases where there is a real effect	`seq(.1,.9,by=.2)`
m	number of iterations	`1e4`
n	sample size	`16`
d	standardized effect size (aka *Cohen's d*)	`c(.25,.50,.75,1,2)`
pwr	power. if set, program adjusts *d* to achieve power	`NA`
sig.level	significance level for power calculations when *pwr* is set	`0.05`
pval.plot	p-values for which we plot results	`c(.001,.01,.03,.05,.1)`
scriptname	used to set output directories and in error messages	`'swfdr_base'`
datadir	data directory relative to distribution root	`'data/swfdr_base'`
figdir	figure directory relative to distribution root	`'figure/swfdr_base'`
save	save parameters, results, and plots; sets *save.rdata* and *save.plot*, not *save.txt*	`FALSE`
save.rdata	save parameters and results in RData format	`FALSE` (set by *save*)
save.txt	save results in txt format. **CAUTION: big and slow**	`FALSE` (not set by *save*)
save.plot	save plots	`FALSE` (set by *save*)
clean	remove contents of data and figure directories; sets *clean.data* and *clean.fig*	`FALSE`
clean.data	remove contents of data directory	`FALSE` (set by *clean*)

parameter	meaning	default
clean.fig	remove contents of figure directory	`FALSE` (set by *clean*)

Statistical Details

- The program assumes normally distributed data with equal variance.
- The p-values are from a two-sided, unpaired, equal variance t-test (R's `t.test` with default settings for `alternative`, `paired`, and `var.equal`).
- The default sample size ($n = 16$) gives about 80% power for $d = 1$ and $sig.level = 0.05$. Power for the full range of d is

d	0.25	0.50	0.75	1.00	2.00
power	0.10	0.28	0.54	0.78	0.9998

Directory Structure

The root directory contains the usual GitHub files: LICENSE, README.md, .gitignore. There's also a NEWS.md file that lists major differences between releases. The subdirectories are

- `css/` - style sheets used by the document generation process
- `data/` - data files
- `doc/` - documentation
- `figure/` - plots
- `script/` - scripts
- `tool/` - helper scripts for the document generation process

Other Scripts

TBD

Functions

`run` - *Run the program.*

Description

Top-level function. Sets parameters (via `init`), does the work (via `doit`) and optionally saves the results (via `saveit`)

Usage

```
run=function(...)
```

Arguments

argument	meaning	default
`...`	parameters passed to `init`	see `init`

Value

This function is invoked for its side-effect. It has no return value.

Examples

```
# this code block assumes your working directory is the root of the repository

source("script/swfdr_base.R");
# run default process
run();

# run default process and save results in directories data/guide01 and figure/guide01
run(save=T,datadir='data/guide01',figdir='figure/guide01');

# reduce runtime by reducing number of simulation runs and simulated cases
run(m=1e3,d=c(0.25,0.5,1),prop.true=c(0.3,0.5,0.8));

# specify power directly and let program adjust effect size
run(m=1e3,pwr=c(0.1,0.3,0.8),prop.true=c(0.3,0.5,0.8));
```

See Also

`init` for more information.

init - *Initialize program parameters.*

Description

Processes parameters and stores them in global variables. Creates parameter grid, called `cases`, containing all combinations of parameters. Creates output directories if they do not exist.

Usage

```
init(prop.true = seq(0.1, 0.9, by = 0.2),
     m = 10000,
     n = 16,
     d = c(0.25, 0.5, 0.75, 1, 2),
     pwr = NA, sig.level = 0.05,
     pval.plot = c(0.001, 0.01, 0.03, 0.05, 0.1),
     scriptname = "swfdr_base",
     datadir = file.path("data", scriptname),
     figdir = file.path("figure", scriptname),
     save = F, save.rdata = save, save.txt = F, save.plot = save,
     clean = F, clean.data = clean, clean.fig = clean)
```

Arguments

argument	meaning	default
`prop.true`	fraction of cases where where there is a real effect.	`seq(0.1, 0.9, by = 0.2)`
`m`	number of iterations.	`1e4`
`n`	sample size.	`16`
`d`	standardized effect size (aka *Cohen's d*)	`c(0.25, 0.5, 0.75, 1, 2)`
`pwr`	power. if set, program adjusts `d` to achieve power.	`NA`
`sig.level`	significance level for power calculation	`0.05`
`pval.plot`	p-values for which we plot results	`c(1e-03, 0.01, 0.03, 0.05, 0.1)`
`scriptname`	script name. Used to construct output directory path names.	`"swfdr_base"`
`datadir`	path name of directory for data files.	`"data/swfdr_base"`
`figdir`	path name of directory for plots.	`"figure/swfdr_base"`
`save`	logical. sets `save.rdata` and `save.plot`	`FALSE`
`save.rdata`	if TRUE, save parameters and results (actually, all global variables) in RData format. The output filename is `globals.RData` in directory `datadir`.	`FALSE` (set by `save`)
`save.txt`	save simulation and interpolation results as tab-delimited text files. The output filenames are `sim.txt` amd `interp.txt` in directory `datadir`. **CAUTION: big & slow!**	`FALSE`
`save.plot`	if TRUE, save plots as png files in `figdir`. The output filenames are `plot_byd.png`, `plot_byprop.png`, `plot_vsd.png`, `plot_vsprop.png`	`FALSE` (set by `save`)
`clean`	logical. sets `clean.data` and `clean.fig`.	`FALSE`
`clean.data`	if TRUE, delete contents of datadir and start fresh	`FALSE` (set by `clean`)
`clean.fig`	if TRUE, delete contents of figdir and start fresh	`FALSE` (set by `clean`)

Value

The `cases` data frame is returned invisibly.

Details

For the default parameters, the `cases` parameter grid expands to 25 cases (5 values of prop.true x 5 values of d; all other parameters have single-valued defaults). We do 10,000 simulations for each case for a total of 250,000 simulations. This takes about 3 minutes on my small Linux server.

This function is usually called by `run`. It may be called directly if the user wishes to perform custom initialization.

Examples

```
# initialize parameters with default values
init();

# initialize parameters with default values but save results in directories data/guide01 and plots in f
```

```
init(save=T,datadir='data/guide01',figdir='figure/guide01');

# initialize parameters with values requiring less runtime by reducing number of simulation runs and si
init(m=1e3,d=c(0.25,0.5,1),prop.true=c(0.3,0.5,0.8));

# specify power directly and let program adjust effect size
init(m=1e3,pwr=c(0.1,0.3,0.8),prop.true=c(0.3,0.5,0.8));
```

`doit` - *Do the work.*

Description

Runs simulation (via `dosim`), interpolates relevant columns of the simulation results at fixed p-values (via `dointerp`), and plots the results and optionally save the plots (via `doplot`).

Usage

```
doit=function()
```

Value

This function is invoked for its side-effect. It has no return value.

Plot Functions - *Plot the results*

Description

These functions operate on the `sim` and `interp` data frames produced by `dosim` and `dointerp` respectively.

Usage

```
doplot=function(save.plot=F)

plot_byprop=function(save.plot=F,d1=1,sig.level=.05)

plot_byd=function(save.plot=F,prop.true1=0.5,sig.level=.05)

plot_vsprop=function(save.plot=F,d1=1,sig.level=.05)

plot_vsd=function(save.plot=F,prop.true1=0.5,sig.level=.05)
```

Arguments

argument	meaning	default
`save.plot`	if TRUE, save the plot. The output format is PNG. The output filename is the name of the function with `.png` suffix in directory `figdir`, eg, `figure/plot_byprop.png`.	`FALSE`
`d1`	fixed value of `d` for dimension reduction. If `d1` is not in the `d` vector, it is set d1 to `max(d)`.	`1`
`prop.true1`	fixed value of `prop.true` for dimension reduction. If `prop.true` is not in the `prop.true` vector, it is set to the first value in `prop.true`.	`0.5`
`sig.level`	values of p-value and FDR marked by dashed lines on the plot.	`0.05`

Details

`doplot` is the main plot function. It calls separate functions for each of the four kinds of plot. * `plot_byprop` plots FDR by prop.true for one value of d * `plot_byd` plots FDR by d for one value of prop.true * `plot_vsprop` plots FDR vs prop.true for one value of d at fixed p-values * `plot_vsd` plots FDR vs d for one value of prop.true at fixed p-values

Each of the `plot_` functions plots a different slice of theoretical and empirical FDR as a function of three variables: FDR=f(prop.true,d,pval) The functions differ in how they reduce four dimensions (FDR and the three variables) to something that can be plotted in two dimensions.

Each function starts by fixing one variable to a single value. Next, the function splits the data into groups based on a second variable. Finally, it plots each group vs. the remaining variable, using different line types (solid vs dashed) to distinguish theoretical and empirical FDR.

`plot_byprop` and `plot_byd` operate on `sim`; `plot_vsprop` and `plot_vsd` operate on `interp`.

Documentation

The documentation comprises

- this User Guide
- README
- NEWS lists major differences between releases
- R-style function-level documentation
- Science: Science-Wise False Discovery Rate explains the scientific concepts underlying the software
- Software: Science-Wise False Discovery Rate discusses the software design and design choices
- Exercises: Science-Wise False Discovery Rate provides exercises for the reader to improve the program

Document files have several formats.

1. R markdown (`Rmd`) is the source format. I generate the ooutput formats programmatically using, e.g.,

```
rmarkdown::render(doc/README.Rmd,'github_document')
rmarkdown::render(doc/guide.Rmd,(c('html_document','pdf_document'))
```

2. markdown (`md`) is the output format GitHub prefers for documents displayed on its site. Due to limitations in GitHub's `md` support, I only use this format for simple documents, viz,. README and NEWS.
3. `html`. These are self-contained HTML files that should display correctly on your local computer or website. Because they are self-contained, they tend to be huge and inscrutable. Note that GitHub

shows HTML files as raw text, which is not very illuminating, but has an HTML preview feature that renders the files as expected.

4. `pdf`. GitHub renders the files as expected **but disables links**.

See Also

Felix Schönbrodt's blog post and ShinyApp got me going down this path and offer an insightful, different perspective. Blog posts by Daniel Lakens and Will Gervais are also interesting.

Papers by David Colquhoun, Leah Jager and Jeffrey Leek, and John Ioannidis cover it all with full statistical rigor and much more detail.

Author

Nathan (Nat) Goodman, (natg at shore.net)

Bugs and Caveats

Please report any bugs, other problems, and feature requests using the GitHub Issue Tracker. I will be notified, and you'll be apprised of progress.

Known Bugs and Caveats

- The simulation results are noisy for $m <$ 5x103 and dubious for $m <$ 103; the program does not run at all for $m < 2$.
- The software is pretty basic. See Exercises: Science-Wise False Discovery Rate for a list of known limitations.

Copyright & License