# PFunc

PFunc is a set of tools for fitting and analyzing function-valued traits. It was designed for quantifying preference functions (hence the name) in biological studies of sexual selection and evolution of mate preferences. PFunc accepts a bivariate dataset composed of stimuli and responses to those stimuli as its input. It then fits cubic splines to these data using the gam function in R, and it displays the curves, along with several useful measurements of the shape of the curve. Users have several options for adjusting the curves and the tools used to measure the curves, and for outputting results for further analysis.

This file was last updated 2022-09-22 for PFunc version 1.0.3.

## Table of Contents



## Files

`PFunc.py` - the main file to run for the full PFunc GUI experience
`PFunc_RCode.R` - the supporting R code that fits the splines and extracts the useful metrics. This code *can* be run on its own in R without the GUI

`README.md` - important information for installing and using the program
`README.pdf` - important information for installing and using the program
`demo_data_horizontal.csv` - example data in one of two possible layouts
`demo_data_vertical.csv` - example data in one of two possible layouts
`PFunc.dockerfile` - Docker build file for people wanting to run PFunc in a container
`PFuncIcon.png` - icon file
`PFuncPath.txt` - may help users who encounter path-related errors
`COPYING.txt` - the full GPLv3 license

## Setup Overview

There are two broad approaches you can take when it comes to setting up PFunc on your computer:

**OPTION 1** Install Python and R, along with the necessary supporting packages, and run the script directly on your computer. Pros: Lighter on system resources. Cons: A future update for Python or R could mess something up in the script.

**OPTION 2** Run PFunc inside of a Docker container. Pros: Once it's set up, it will always work, because the core program versions will never change. Cons: Docker can take up a bunch of system resources, so you might not get the best performance on old or cheap computers. Also note: You'll have to sign up for a Docker account and you'll need admin privileges on the computer, so if your computer is managed by your university/employer, you might need special permission to install and run Docker.

## Setup Option 1: Script

**A note on package versions**

PFunc was written with particular versions of software and packages in mind. Given that PFunc depends on these packages to run, it is possible that dow the road, these packages could be updated in a way that negatively impacts the functionality of PFunc. If this happens, you have two options:

- Check https://github.com/Joccalor/PFunc/releases/latest for the most recent version of PFunc to see if we've updated it to run with newer packages.

or:

- Search online for old versions of the packages that work with this version of PFunc. Below is a list of versions known to work with PFunc. These packages should all be archived online. Don't worry if you cannot find these exact versions; other close versions will probably function just fine.
    - R 3.3.1, 3.5.1, 3.6.1, 4.2.1
    - mgcv 1.8-17, 1.8-27, 1.8-40
    - Python 3.5.2, 3.7.1
    - matplotlib 1.5.3, 2.0.1
    - rpy2 2.8.5, 3.0.1, 3.5.3

For packages installed through pip, you can specify a version number with the double equals sign (==). So for example, if you were to want to uninstall your current version of matplotlib and install version 2.0.1, you would use these commands:

```
pip uninstall matplotlib
pip install matplotlib==2.0.1
```

**1. Install and Set Up R**

1. Visit https://cran.r-project.org/ and follow the links to download the latest version of R for your operating system. Install R as you would a normal program.
   **For Windows users:** The setup wizard will give you the option of installing the 32-bit files or the 64-bit files. Install both.
   **For Mac users:** As noted on the R homepage, you will need to install XQuartz https://www.xquartz.org/.
   **For Linux users:** You may instead install R from the command line.

2. Install the mgcv package in R by opening R and entering the following command: `install.packages("mgcv")`. You will be prompted to select a mirror from which to download the package. Select an option that is relatively close to your location. Follow any instructions R gives you for installing the package.

**2. Install and Set Up Python**

1. Visit https://www.python.org/ and follow the links to download the latest version of Python 3.x for your operating system. Install Python as you would a normal program.
   **Note:** If you run into display issues with the latest version of Python, try downloading and installing Python version 3.7.1 instead https://www.python.org/downloads/release/python-371/. You can tell your computer to run this specific version by using `pip3.7` and `python3.7` where appropriate in the commands below.
   **For Windows users:** the first screen of the installation wizard will ask if you want to add Python to PATH. Make sure to select this option.
   **For Linux users:** you may instead install Python from the command line.

2. Install the required Python libraries (matplotlib and rpy2). To do this, open up the terminal (if you are not sure how, run a search for "terminal" or "command prompt" on your computer), and run one of the following commands.
   **Note:** You may encounter display issues with newer versions of matplotlib. If you do, you can install version 2.0.1 instead by relpacing `matplotlib` with `matplotlib==2.0.1` in the commands below. **For Windows users:** enter this command: `pip install matplotlib rpy2`. If that doesn't work, try calling pip this way instead: `py -m pip install matplotlib rpy2`
   **For Mac and Linux users:** enter this command: `pip3 install matplotlib rpy2`. If your computer doesn't recognize pip3, try using `pip install matplotlib rpy2` instead.

You are now ready to run PFunc.

## Setup Option 2: Docker

If the script setup described above worked for you, you can skip this section and continue to the usage section below.

Special thanks to Peter Naylor for helping us with this https://github.com/PeterJackNaylor.

**1. Install and Set Up Docker**

1. Download and install Docker https://docs.docker.com/engine/install/
   **Note:** As part of setup, you will likely have to sign up for a (free) Docker account.
   **For Windows users:** You'll need to install VcXsrv https://sourceforge.net/projects/vcxsrv/ or another similar X server.
   **For Mac users:** Make sure XQuartz is installed (you probably did this while installing R above) https://www.xquartz.org/. Once installed, open XQuartz, go to Preferences > Security and tick the box that says, "allow connections from network clients."

**2. Acquire the PFunc Docker image**

1. Start Docker.
2. Open a terminal window (as described above in Install and Set Up Python).
3. Run `docker pull joccalor/pfunc:latest`
   **For Mac and Linux users:** you may need to run this with superuser permissions, like `sudo docker pull joccalor/pfunc:latest`
   **Note:** the Docker build file (PFunc.dockerfile) is included for those who wish to inspect it or build their own PFunc images. Everyone else can ignore this file.

**3. Run the PFunc Docker image the first time**

1. Start Docker. **For Mac users:** Start XQuartz as well. Then in a terminal, run: `xhost +`, or if that doesn't seem to be working, try `xhost + 127.0.0.1` instead.
   **For Windows users:** Start VcXsrv as well (it'll be somewhere like c:\Program Files\VcXsrv\xlaunch.exe). If it asks, you can tell it to display in multiple windows and to start no client.
2. Get the full path for where your data files are stored. In the commands below, replace DATAPATH with that filepath.
   **Note:** If there are spaces anywhere in your filepath, you need to surround the filepath with quotation marks.
3. Start up the container with the appropriate terminal command below:
   **For Windows users:** `docker run --name PFunc -v DATAPATH:/root/PFunc/data -e DISPLAY=host.docker.internal:0 joccalor/pfunc:latest`
   **For Mac users:** `sudo docker run --name PFunc -v DATAPATH:/root/PFunc/data -e DISPLAY=host.docker.internal:0 joccalor/pfunc:latest`
   **For Linux users:** `sudo docker run --name PFunc -v DATAPATH:/root/PFunc/data -v /tmp/.X11-unix/:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY joccalor/pfunc:latest`
   **Note:** Instead of specifying the entire filepath, you can navigate to the location of your data using the `cd` command on the command line. Then in the place of DATAPATH in the commands above, you can just

write the shortcut for the current directory. In Windwos it's %CD%, and in Mac and Linux it's $PWD.
**Note:** You'll need to run these steps again anytime you want to specify a new location for your data on your computer.
**Note:** This command will name the container "PFunc". You can assign a different name by chaning what comes directly after "--name" in the command.

**4. Start an existing the PFunc Docker container**

1. Start Docker.
   **For Mac users:** Start XQuartz as well. Then in a terminal, run: `xhost +`, or if that doesn't seem to be working, try `xhost + 127.0.0.1` instead.
   **For Windows users:** Start VcXsrv as well (it'll be somewhere like c:\Program Files\VcXsrv\xlaunch.exe). If it asks, you can tell it to display in multiple windows and to start no client.
2. Start the PFunc container either from the Docker UI or from the command line with `docker start PFunc` (or if you gave your container a different name, replace "PFunc" with that name).

## Usage

Below is a bare-bones overview of using PFunc. For more details on usage as well as general theory underlying the study of preference functions, check out our paper "Describing mate preferences and other function-valued traits" (2017) by Kilmer, Fowler-Finn, Gray, Höbel, Rebar, Reichert & Rodríguez in the Journal of Evolutionary Biology 30(9):1658-1673; doi: 10.1111/jeb.13122.

### Data Input

PFunc is expecting data with two main components: a set of **stimuli** (plotted along the *x*-axis) and a set of **responses** to those stimulus values (plotted on the *y*-axis).

PFunc needs a bare minimum of three data points to fit one spline (although we HIGHLY recommend using more than three data points). This means that if you are splining at the individual level (which is what we tend to do), each individual must be tested against a range of stimulus values.

The other main way to create preference functions is to test a sample of individuals across a range of stimuli, and then use the combination of those individual responses to create one spline. This would be the way to go in a species where you could only test each individual once. If you use this second option, then be aware that you are no longer splining at the individual level, but at a group level, and so it will be necessary to communicate this to PFunc by grouping individuals in your data file. If you are using the horizontal data file layout (described below), then you can create one response column with each of the different responses. If you are using the vertical data layout (described below), then you can create a new column to code for group identity. When you open the vertical file in PFunc, then you can tell PFunc to spline according to group identity, rather than individual identity.

You have the option of formatting your data in one of two ways:

- In the **horizontal** layout, the first column of data contains the stimulus values, and each subsequent column contains the responses of an individual to those stimulus values. See the

`demo_data_horizontal.csv` file for an example.
- In the **vertical** layout, data are organized into a minimum of three columns: one with individual identities, one with stimulus values presented to those individuals, and one with the responses of those individuals to those stimuli. See the `demo_data_vertical.csv` file for an example.

Regardless of which layout option you choose, you must save your data as a **.csv** file (any type of .csv will work).

## Running the PFunc GUI

### Startup

You have several different options for running PFunc. Make sure that `PFunc.py` and `PFunc_RCode.R` are together in the same directory. If you encounter an error, see the Troubleshooting section below.

- **Option 1: Run PFunc through IDLE**
  The most straightforward way to run PFunc is through IDLE. Right-click (or the equivalent on a Mac) on the `PFunc.py` file, and choose to "Open With..." IDLE (or "Edit With Idle" on Windows if that option is available). This will start up IDLE with the code for PFunc displayed in a window. Click on the window of colorful code to make sure it is the current active window, and then run the code either by pressing F5 or by clicking Run > Run Module in the menu at the top bar.
- **Option 2: Run PFunc from the command line**
  - Take note of the directory where PFunc is saved.
  - Open up an instance of your terminal (command prompt in Windows) and navigate to the PFunc directory using the `cd` command. Examples: `cd "C:\Users\name\PFunc"` in Windows; `cd "/Users/name/PFunc"` in Mac OS X; `cd "/home/name/PFunc"` in Linux.
  - Then enter this command for Windows: `python PFunc.py` (or if that doesn't work, try: `py PFunc.py`), or this command for Mac or Linux: `python3 PFunc.py`.
- **Troubleshooting**
  You might encounter an error while trying to run PFunc if rpy2 has trouble finding R on your computer. One sign of this error is if the error message mentions R_HOME or PATH. To solve this, you'll have to find where R is installed on your computer and follow the instructions in the `PFuncPath.txt` file that accompanies this program.

As stated on the R downloads page, Mac users may encounter errors with R if XQuartz is not installed. In PFunc, problems are most likely to be encountered when trying to output graph files without XQuartz installed. XQuartz can be downloaded from https://www.xquartz.org/.

### The Interface

Once you open your data file, PFunc will display a page of graphs, each one showing data from a single individual with a spline fit through the points. A page displays up to nine individuals, and you can navigate to different pages with the controls at the bottom of the screen.

You can select a graph by clicking on it, and you can enlarge a graph by double-clicking on it.

When a graph is selected, its smoothing parameter is displayed in the Smoothing box on the control panel on the right-hand side of the window, and several important metrics are displayed in the Summary box right below the Smoothing box.

***Smoothing Parameter***

The smoothing parameter controls how stiff or how wobbly a spline is. A low smoothing parameter yields a wobbly spline, and at extremely low values, the spline runs through every data point. A high smoothing parameter yields a spline that is a gentle curve, and at extremely high values, the spline is a straight line. The gam function in R (which fits the splines) chooses an optimal smoothing parameter for you. In most cases, it is not necessary to alter the smoothing parameter. However, should you wish to change it, you have two main options:

- **Specify your own smoothing parameter.** You can use the "-" and "+" buttons next to the smoothing parameter to increase and decrease the value. You may also type in your own smoothing parameter and set it by pressing the Enter key. When a smoothing parameter has been altered, the small magenta dots in the bottom left corner of the graphs change to cyan. If at any point you want to revert to the default smoothing parameter, simply press the Reset button in the Smoothing box (this will also change the cyan dot back to magenta).
- **Adjust the limits imposed upon smoothing parameters.** PFunc was built with the philosophy that preferences function splines should be neither too wobbly nor too stiff. Therefore, by default, PFunc restricts all smoothing parameter values to be between 0.05 and 5. We found this to be a reasonable range for the types of data we were working with, but you may find a different range to be more suitable for your data. You can change this range in the Smoothing Limits setting (described below). Alternatively, you may decide that you want no limits on your smoothing parameters, and instead want to see what default the program gives you, no matter how big or how small. You have the option of turning off smoothing parameter limits altogether (also described in the Smoothing Limits setting below). When you make changes to the Smoothing Limits setting, only the graphs with magenta (and not cyan) dots will change. This prevents you from accidentally wiping away any intentional changes to specific smoothing parameters.

***Summary***

The summary window displays several useful measurements of the spline in the selected graph:

- **Peak Preference:** This is the *x*-axis value that corresponds to the highest point of the curve, as illustrated by the red vertical lines on the graphs.
- **Peak Height:** This is the *y*-axis value that corresponds to the highest point of the curve.
- **Tolerance:** This is the width of the curve at a certain height. By default it is measured 1/3 of the way down from the peak of the curve to zero. See the Tolerance settings below to change the proportion of the drop or to measure tolerance at a set *y*-axis value for all curves.
- **Strength:** This is a measure of how far the curve drops away from the peak. By default, the measure of strength depends on the elevation of the spline, because the same absolute drop in a curve will be a larger proportion of a curve that is close to the baseline, compared to a curve that is high up. This

height-dependent strength is calculated as the squared coefficient of variation of the *y*-axis values along the spline. A height-independent option may be used instead, which calculates strength as the standard deviation of the *y*-axis values along the spline divided by their range.
- **Responsiveness:** This is the average height of the curve.
- **Smoothing:** This displays the smoothing parameter used to generate the curve with the above values. Most of the time it is the same as the value in the Smoothing box.

**Settings**

You have access to a variety of settings that give you control over how your data are analyzed. Below are descriptions of the available settings. PFunc will always start up with the default settings, but you can save any changes you make to the settings through the File menu ("Save Current Settings"), and then load them again later ("Load Previous Settings"). You can also revert to the default settings ("Restore Default Settings").

- **View** Here you can toggle the visibility of five different things on or off, depending on how you want to view your graphs. If you choose to export your spline graphs, these View settings will carry over to the output file.
- **Smoothing Limits** Here you can adjust the limits to the smoothing parameters that are chosen for your splines. By default, PFunc limits them to values between 0.05 and 5, but you have full control over the minimum and maximum values here, and you also have the option to turn these limits off entirely with the checkbox.
- **Find Local Peak** Here you can narrow the range in which PFunc searches for peaks. If it finds a peak in your specified range, it uses that value. If no peak is found, then PFunc uses the regular peak instead.
- **Tolerance** This gives you fine control over how tolerance (the width of the curve at a given height) is measured.
    - First you have the option of measuring tolerance relative to the height of the peak (e.g. 1/3 of the way down from the top), or measuring it at a set *y*-axis value. To switch between these two options, click one of the two radio buttons next to the appropriate settings: the radio button next to "Drop from peak" is for the relative option, and the radio button next to "At set value" is for the absolute option.
    - Once you've selected *how* tolerance should be measured, you can control *where* it is measured. If you've chosen the relative option, you can control the drop from the peak. This takes the distance from the peak of the curve to the floor (which you can also set), and measures tolerance at a particular proportion of that distance down. If, on the other hand, you've chosen to measure tolerance at an absolute height, you then can specify that exact height with the "At set value" setting.
    - Finally, you can decide whether you want a broad measure of tolerance or a strict one. The difference comes into play when there are secondary peaks in your curves. If you choose the broad option, then tolerance will be measured as the total width of the curve under the primary peak plus any relevant secondary peaks. If you choose the strict option, then tolerance will be measured as the width of the curve *only* under the primary peak.
- **Strength** Here you have the option of using the height-dependent measure of strength (recommended) or the height-independent measure of strength (see above for brief description of the two, and see our paper for a more complete discussion).

### Group-Level Splines

You can combine individual splines to form group-level splines. This can come in handy if you want to generate splines at the replicate-, treatment-, family-, population-, or species-level. Group splines can even be combined to form higher-order group splines.

To do this, go to Advanced > Construct Group-Level Spline... . The pop-up window allows you to name your new group and select which individuals belong in the group (click and drag or use the Shift and Ctrl keys to select multiple individuals). Once you are finished, press Okay, and your new group-level spline will be added alongside your other splines.

Note that this does not affect your input data file; if you want to retain these values, you'll need to output them (see below). Also note that group-level splines may be best fit with lower smoothing parameters than individual-level splines.

### Message Log

PFunc keeps track of all its warnings and confirmations, even ones that it doesn't explicitly make pop-ups for. To see the running log of messages, go to Advanced > Show Message Log.

### Output

Here are descriptions of the various output options available under the File menu.

- Output Spline Figures: Creates a .pdf, .eps or .svg file with all the graphs using the current smoothing parameters.
- Output Spline Summaries: Creates a spreadsheet that contains all of the information in the Summary window for every individual.
- Output Spline Points: PFunc extracts the *y*-values at 201 evenly-spaced points along the curve, and it saves these as a spreadsheet. This is useful if you want to plot your curves in a different program.
- Output Tolerance Points: Creates a spreadsheet containing all of the *x*-axis values that correspond to the upper and lower limits of tolerance--that is, the start and stop points of the horizontal blue lines in the graphs.

## Running PFunc from the R Command Line

If you are comfortable working in the R command line environment, you may use Pfunc without the GUI. Note that when PFunc is used this way, data **must** be set up in the horizontal format (as in `demo_data_horizontal.csv`), never in the vertical format (as in `demo_data_vertical.csv`).

### Startup

Open R. Set the working directory to wherever the `PFunc_RCode.R` file is saved on your computer.

- One way to do this is by finding the directory path and entering the command `setwd(directory/path)`. You can check your current working directory with the command `getwd()`.

- There is another way to do this if you are not running R directly from your terminal. Depending on your version of R, you will find the option either under "File > Change dir..." or "Misc > Change Working Directory..."
- Load the script into R with this command `source("PFunc_RCode.R")`
- Load your data into R with a command like `mydata <- read.csv("datafile.csv")`
- Now you can run the default analysis on your data with the command `PFunc(mydata)`. See below for a full set of options.

**Arguments of PFunc**

Here is a list of arguments that you can specify when calling PFunc directly from R.

- `input.data` - the name of your dataset in the R environment (no default)
- `diagnose.col` - an optional argument used only when assessing individual functions. To see the graph and metrics for a single individual without creating a new file, set this number equal to the corresponding column in your datasheet (default = 0).
- `diagnose.sp` - an optional argument used only when assessing individual functions. By setting this to a positive number, you can specify the smoothing parameter of individual functions that you are examining (default = -1).
- `auto.sort` - an option to sort the data by increasing values of the stimulus variable (default = TRUE).
- `blocklist` - if there are columns of your data that you wish to exclude from analysis, place their numbers in this argument with the concatenate `c()` function. For example, to exclude the third and fourth columns, set `blocklist = c(3, 4)` (default = NULL).
- `allowlist` - if there are only a few specific columns of your data that you wish to analyze and exclude the rest, you can place the included columns' numbers in the allowlist with the concatenate `c()` function. For example, to include only the third and fourth columns in analysis, set `allowlist = c(3, 4)` (default = NULL).
- `sp.binding` - when true, it constrains the auto-generated smoothing parameters to be between `min.sp` and `max.sp` (see below) (default = TRUE).
- `sp.assign` - allows you to specify your own smoothing parameters (sp) for individual columns rather than letting the script generate optimized values. Must be in the form of a two-column matrix in which the first column contains the column numbers for which you want to specify smoothing parameters. The second column contains the corresponding desired smoothing parameters. For example, `new.sp <- matrix(c(2, 1, 3, .5, 4, 0), ncol=2, byrow=T)`. Here, the individual in input.data column 2 gets an sp of 1, the individual in input.data column 3 gets an sp of .5, and the one in column 4 gets an sp of 0. To run with these values, set `sp.assign = new.sp` (default = 0).
- `max.sp` - if you are constraining auto-generated smoothing parameters with the `sp.binding` argument (above), `max.sp` is the upper limit (default = 5).

- `min.sp` - if you are constraining auto-generated smoothing parameters with the `sp.binding` argument (above), `min.sp` is the lower limit (default = 0.05).
- `peak.within` - tells PFunc where to search for local peaks. If a single number, x (0 < x <= 1), is passed for this argument, PFunc will search in the central x proportion of the stimulus range. For example, if this argument is passed with 0.7, then PFunc will look in the central 70% of the function for a peak. If it finds one, it uses it. If it doesn't find one, it uses the regular peak. If two numbers are passed instead (e.g., `peak.within = c(120, 150)`), then those will serve as the lower and upper bounds between which PFunc will look for local peaks.
- `summary.out` - filename for the output summary that includes peak, tolerance, and other measures of each function. If FALSE, no file will be made (default = "spline_summaries.csv").
- `graph.out` - filename for the output graphs for all analyzed functions. If FALSE, no file will be made (default = "spline_graphs.pdf").
- `points.out` - filename for the output points that make up each function. Useful for plotting functions in external programs. If FALSE, no file will be made. You can specify the number of points you want to output for each function in the `predictions` argument below (default = FALSE).
- `max.y` - for the output graphs in `graph.out` (above), this argument specifies the height of the vertical axis. Leave at default value of 0 for automatic.
- `pdf.row` - for the output graphs in `graph.out` (above), this argument specifies the number of rows of graphs each page of the pdf will have (default = 4).
- `pdf.col` - for the output graphs in `graph.out` (above), this argument specifies the number of columns of graphs each page of the pdf will have (default = 3).
- `graph.points` - an option to turn data points on or off in the output graphs (default = TRUE).
- `graph.peak` - an option to turn on or off a red vertical line at the peak of the function (default = TRUE).
- `graph.tol` - an option to turn on or off a blue horizontal line showing the width calculated for tolerance (default = TRUE).
- `graph.sp` - an option to print the smoothing parameter of each function above its corresponding graph (default = FALSE).
- `graph.se` - an option to plot the standard error around the spline (default = FALSE).
- `drop` - specify how far down from the peak of the function that tolerance (the width) should be measured (default = 1/3).
- `tol.mode` - how would you like the script to calculate tolerance? The default value "broad" (alternatively "norm") looks at the width of the function under the main peak plus any other areas at that same *y*-value that may be under secondary peaks. The alternative value "strict" only counts the width of the part of

the function that contains the main peak and ignores other peaks, even if they rise above the chosen *y*-value.

- `tol.floor` - this specifies the baseline of the data. It is used in conjunction with `drop` to determine the height where tolerance should be measured. For example, if `drop` equals 1/3, and `floor` equals 0, then tolerance will be measured at 1/3 of the distance from the peak to 0 (default = 0).
- `predictions` - if you want to extract points along your splines using `points.out`, this argument tells the script how many points you would like. The default value 0 outputs points only at the input stimulus values. Anything greater than 0 will output that many points evenly spaced along the *x*-axis.
- `ghost` - if you are specifying your own smoothing parameters with the `assign.sp` argument, and if you are creating an output graph file with the `graph.out` argument, then setting ghost to TRUE will plot splines with the original default smoothing parameters in gray alongside your new splines. This is useful for comparing the effects of your altered smoothing parameters (default = FALSE).
- `allfromsplines` - an option to specify how you would like strength and responsiveness to be calculated. With the default value of TRUE, they will be calculated from the splines. Change this to FALSE only if you want to calculate strength and responsiveness from your input data points.

**Examples**

The following examples assume that your data file is called "mydata" in the R environment.

- To output spline summaries and a pdf of the graphs using automatic smoothing parameters bound between 0.05 and 5, use this command `PFunc(mydata)`
- To look at an individual spline (say, the one in the third column of your spreadsheet) without outputting any files, use this command `PFunc(mydata, 3)`
- To see what that individual spline would look like with a new smoothing parameter (say, 0.7), use this command `PFunc(mydata, 3, 0.7)`
- To keep this change for the final output, start by constructing a matrix as described in `sp.assign` above. Then set `sp.assign` to the name of the matrix, like this:

```
new.sp <- matrix(c(3, 0.7), ncol=2, byrow=T)
PFunc(mydata, sp.assign=new.sp)
```

- By default, the height at which tolerance is measured is a proportion of the distance between the peak of a curve and the value of the `floor` argument. If you prefer to measure tolerance at the same *y*-axis height across all individuals (say, 2.5), regardless of the height of the peak, set `floor` equal to your desired height, and set `drop` equal to 1, like this `PFunc(mydata, floor=2.5, drop=1)`.

## Acknowledgements

Many thanks to Rafael Rodríguez, Kasey Fowler-Finn, Gerlinde Höbel, David Gray, Darren Rebar, Michael Reichert, Bretta Speck, Danny Neelon, Susan Bertram, Sarah Harrison, Damian Elias, and an anonymous reviewer for their testing and feedback during development.

## Contact

For comments or questions, contact Joey Kilmer at joeykilmer@gmail.com.
Find PFunc on GitHub at https://github.com/Joccalor/PFunc, or download the latest version at https://github.com/Joccalor/PFunc/releases/latest

## License