Package ‘microDecon’
A user’s guide

Version 0.1.0

22/12/18

Donald T. McKnight

donald.mcknight@my.jcu.edu.au

Table of Contents

1.0.0 Background

1.1.0 Introduction

The *microDecon* package is designed to remove contaminant reads from metabarcoding studies (e.g., from bacterial contamination in reagents during a microbiome study). Because it removes contaminant reads, rather than contaminant OTUs, it preserves data from OTUs that are present in both the contamination and the host species or environment. It relies on data from blank samples that are carried through the entire collection, extraction, amplification, and sequencing process, and it only affects OTUs that were sequenced in those blank samples (i.e., OTUs that are at least partially contamination). Additionally, with the exception of some settings for the `remove.thresh()` function, each sample is treated completely independently from all other samples.

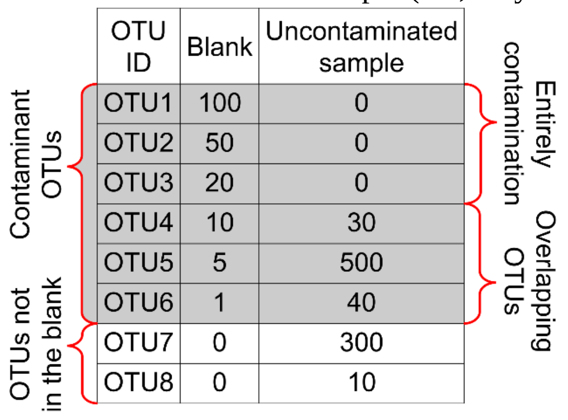
We recommend using the `decon()` function on default settings for decontaminating data, followed by `remove.thresh()` on default settings for removing small amounts of residual contamination.

1.2.0 Citing this package

Please cite this package as McKnight, D. T., R. Huerlimann, D. S. Bower, L. Schwarzkopf, R. A. Alford, and K. R. Zenger. 2018. microDecon: A highly accurate read-subtraction tool for the post-sequencing removal of contamination in metabarcoding studies.

1.3.0 Definitions of terms

- Blank = a negative control that is collected at the same time as the samples and is carried through the entire extraction, amplification, and sequencing process
- Constant = an OTU that is entirely contamination and is used as the basis for decontaminating samples
- Contaminant OTUs = OTUs that amplified in the blank
- Entirely contamination = contaminant OTUs that would not be found on an uncontaminated sample (i.e., they are not present on the host species or environment)
- OTU = operational taxonomic unit
- OTUs not in the blank = OTUs that did not amplify in the blank
- Overlapping OTUs (overlap) = contaminant OTUs that would also be found on an uncontaminated sample (i.e., they are present on the host species or environment)



OTU ID	Blank	Uncontaminated sample
OTU1	100	0
OTU2	50	0
OTU3	20	0
OTU4	10	30
OTU5	5	500
OTU6	1	40
OTU7	0	300
OTU8	0	10

Contaminant OTUs
OTUs not in the blank
Entirely contamination
Overlapping OTUs

Figure 1. Hypothetical sequencing reads, illustrating the terms used in this paper (in an actual study, the uncontaminated sample would be unknown).

1.4.0 Methods and algorithms

1.4.1 Overview

microDecon works on the principle that all of the samples will be contaminated roughly equally from a common source of contamination. For example, if an extraction reagent contains ten species of contaminant bacteria, then, assuming that extractions were done in a standardized manner, we expect each sample to receive roughly the same proportions of each of the ten species. Thus, if we can identify an OTU that is entirely from contamination, we can use the proportions of the other contaminant OTUs relative to it to determine the number of reads that are from contamination, then subtract those from the sample. *microDecon* does this using the following steps (illustrated in Figure 2):

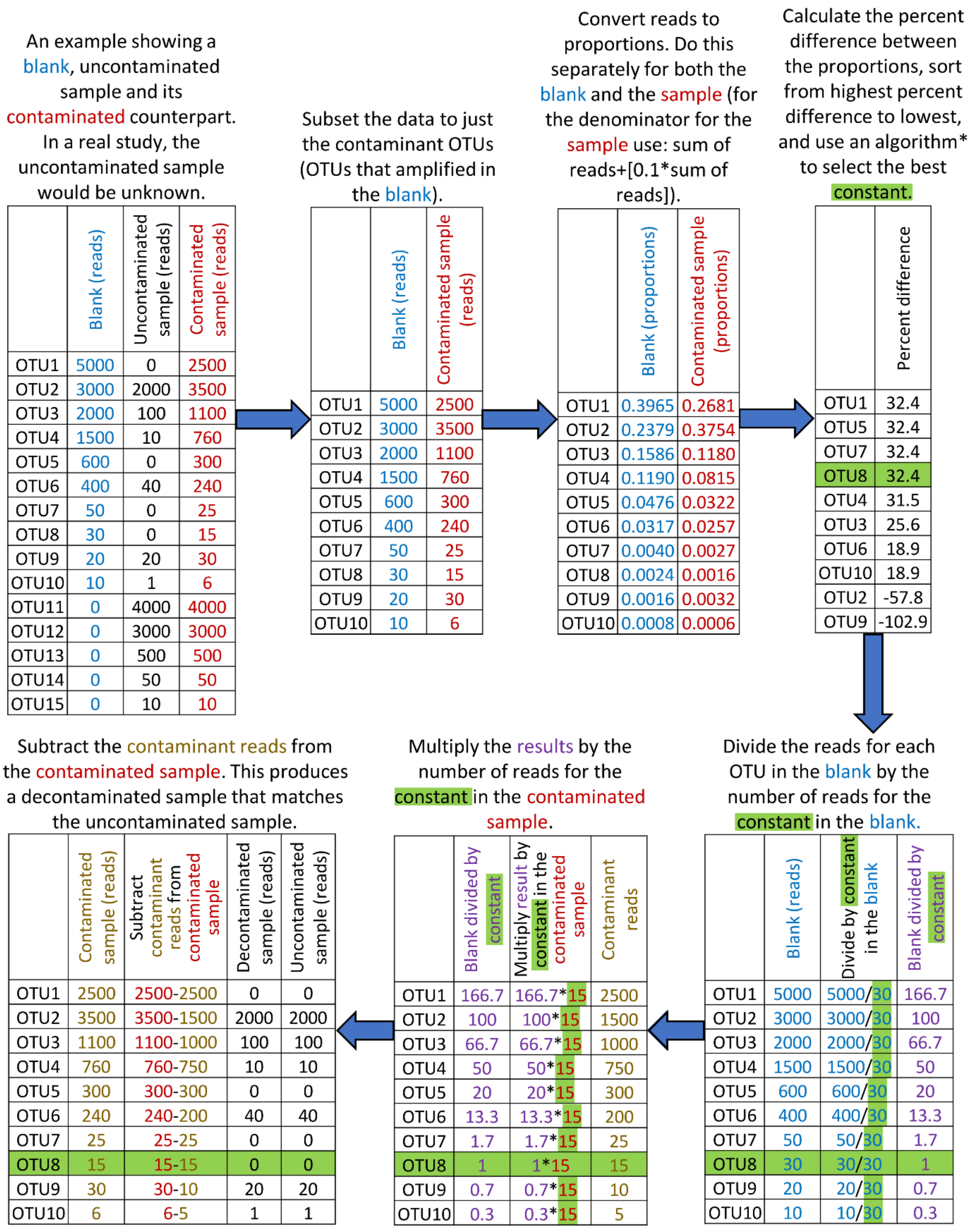
First, it subsets the data to only the OTUs that were present in the blank sample. These are the "contaminant OTUs," which may contain a mix of OTUs that are entirely contamination as well as OTUs whose reads are partially from contamination and partially from organisms that are actually found on the host species or environment being studied (i.e., overlapping OTUs). OTUs that did not amplify in the blank are unaffected by this method.

Next, it subsets the data to just the blank and a single actual sample. This process is iterative and treats each sample completely independently of every other sample. In other words, the following steps are iteratively looped over each sample.

Next, for the sample and the blank, it converts the reads into proportions. For the blank, this is done by dividing the reads for each OTU by the total number of reads in the blank. This is also done for the sample (using only the subset of OTUs that amplified in the blank [i.e., the contaminant OTUs] with a slight modification. Instead of dividing each OTU by the sum of all contaminant OTUs in the sample, each OTU is divided by the sum + 10% of the sum. This was done because if the sum + 10% of the sum was not used, then in cases where there were no overlapping OTUs, the percent differences (calculated bellow) would all be zero (assuming no heterogeneity). Using the sum + 10% of the sum for the sample corrects this situation and allows *microDecon* to more accurately select an appropriate constant.

Next, for the proportions calculated above, *microDecon* calculates the percent difference between the blank and the sample for each OTU. A positive percent difference generally indicates that an OTU is over-represented in the blank (suggesting that it is present in the sample only as contamination); whereas a negative percent different generally indicates that an OTU is over-represented in the sample (suggesting that the OTU was actually present on the host, as well as being present in the contamination).

The percent differences are then used to identify an OTU that is entirely from contamination (hereafter referred to as the "constant"; see section 1.4.2). The constant is then used to calculate the number of reads in the actual sample that are from contamination. The math behind this is simple. For the reads in the blank, it calculates the proportion of each OTU relative to the constant (i.e., each OTU is divided by the constant). Then each of those proportions is multiplied by the number of reads for the constant in the actual sample. Because the constant should not actually be present in the sample and the proportions of the OTUs relative to each other in the contamination are expected to be similar across samples, this results in the number of reads for each OTU in the actual sample that are from contamination. Those reads are then subtracted from the actual sample.



An example showing a blank, uncontaminated sample and its contaminated counterpart. In a real study, the uncontaminated sample would be unknown.

	Blank (reads)	Uncontaminated sample (reads)	Contaminated sample (reads)
OTU1	5000	0	2500
OTU2	3000	2000	3500
OTU3	2000	100	1100
OTU4	1500	10	760
OTU5	600	0	300
OTU6	400	40	240
OTU7	50	0	25
OTU8	30	0	15
OTU9	20	20	30
OTU10	10	1	6
OTU11	0	4000	4000
OTU12	0	3000	3000
OTU13	0	500	500
OTU14	0	50	50
OTU15	0	10	10

Subset the data to just the contaminant OTUs (OTUs that amplified in the blank).

	Blank (reads)	Contaminated sample (reads)
OTU1	5000	2500
OTU2	3000	3500
OTU3	2000	1100
OTU4	1500	760
OTU5	600	300
OTU6	400	240
OTU7	50	25
OTU8	30	15
OTU9	20	30
OTU10	10	6

Convert reads to proportions. Do this separately for both the blank and the sample (for the denominator for the sample use: sum of reads+[0.1*sum of reads]).

	Blank (proportions)	Contaminated sample (proportions)
OTU1	0.3965	0.2681
OTU2	0.2379	0.3754
OTU3	0.1586	0.1180
OTU4	0.1190	0.0815
OTU5	0.0476	0.0322
OTU6	0.0317	0.0257
OTU7	0.0040	0.0027
OTU8	0.0024	0.0016
OTU9	0.0016	0.0032
OTU10	0.0008	0.0006

Calculate the percent difference between the proportions, sort from highest percent difference to lowest, and use an algorithm* to select the best constant.

	Percent difference
OTU1	32.4
OTU5	32.4
OTU7	32.4
OTU8	32.4
OTU4	31.5
OTU3	25.6
OTU6	18.9
OTU10	18.9
OTU2	-57.8
OTU9	-102.9

Divide the reads for each OTU in the blank by the number of reads for the constant in the blank.

	Blank (reads)	Divide by constant in the blank	Blank divided by constant
OTU1	5000	5000/30	166.7
OTU2	3000	3000/30	100
OTU3	2000	2000/30	66.7
OTU4	1500	1500/30	50
OTU5	600	600/30	20
OTU6	400	400/30	13.3
OTU7	50	50/30	1.7
OTU8	30	30/30	1
OTU9	20	20/30	0.7
OTU10	10	10/30	0.3

Multiply the results by the number of reads for the constant in the contaminated sample.

	Blank divided by constant	Multiply result by constant in the contaminated sample	Contaminant reads
OTU1	166.7	166.7*15	2500
OTU2	100	100*15	1500
OTU3	66.7	66.7*15	1000
OTU4	50	50*15	750
OTU5	20	20*15	300
OTU6	13.3	13.3*15	200
OTU7	1.7	1.7*15	25
OTU8	1	1*15	15
OTU9	0.7	0.7*15	10
OTU10	0.3	0.3*15	5

Subtract the contaminant reads from the contaminated sample. This produces a decontaminated sample that matches the uncontaminated sample.

	Contaminated sample (reads)	Subtract contaminant reads from contaminated sample	Decontaminated sample (reads)	Uncontaminated sample (reads)
OTU1	2500	2500-2500	0	0
OTU2	3500	3500-1500	2000	2000
OTU3	1100	1100-1000	100	100
OTU4	760	760-750	10	10
OTU5	300	300-300	0	0
OTU6	240	240-200	40	40
OTU7	25	25-25	0	0
OTU8	15	15-15	0	0
OTU9	30	30-10	20	20
OTU10	6	6-5	1	1

Figure 2: The basic steps used by *microDecon* to decontaminate samples. The process is iterative and each sample is treated completely independently. Note: in this example, OTU1, 5, 7 and 8 had the same percent difference, and any of the four would have produced identical results if used as the constant. In these situations, microDecon simply selects the first of the tied OTUs as the constant, but for this illustration, we selected OTU8 to avoid the appearance that the constant is always the first OTU (after sorting by percent differences). Additionally, this shows only a single run of *microDecon.*

1.4.2 Algorithms for finding the constant

The most challenging part of this method is identifying the best OTU to use as a constant. Any OTU with a percent difference between the blank and the sample that is greater than zero (for proportions) will generally work as a constant to at least a limited degree. However, because of chance variation in the sequencing some OTUs will produce more accurate estimates than others. On one extreme, when OTUs with a very large percent difference are used as the constant, they tend to underestimate the amount of contamination that is present; whereas, on the other extreme, OTUs with very low percent differences tend to overestimate the level of contamination. In between those two extremes, there is a broad Goldilocks zone of OTUs that will produce accurate estimates when used as a constant. Note: OTUs with a percent difference of 1 are filtered out and not considered as candidates for the constant, because a percent difference of 1 indicates that there were no reads for that OTU in the sample).

To test various methods for identifying and selecting a constant from this Goldilocks zone, we simulated a series of metabarcoding studies, including contaminating a sample and sequencing a contaminated blank (these simulations were modifications of simulation 1 in McKnight et al. 2018). The simulations then ranked the contaminant OTUs by the percent difference between the blank and the sample (greatest to smallest), and attempted to decontaminated the sample by iteratively using each OTU with a percent difference greater than zero as the constant. They returned the rank of the OTU that produced the most accurate corrections when used as the constant, as well as any input information about the samples that would be available in an actual study (e.g., number of OTUs in the blank, number of OTUs in the sample, etc.). From those results, we examined and tested correlations between the input information and the rank of the best OTU to use as the constant. This allowed us to identify two useful regression equations.

The first equation (hereafter referred to as "regression 1") simply takes the percent differences calculated previously (using the sum of the reads in the sample plus 10% of the sum as the denominator for the calculations for the sample), calculates the number of OTUs with a percent difference >10%, and uses that number as x in the following equation: rank of best constant = 0.7754*x-4.2185. This is the equation used by the `decon.regress1()` function. The results of the simulation that produced that regression equation are shown in Figure 3.

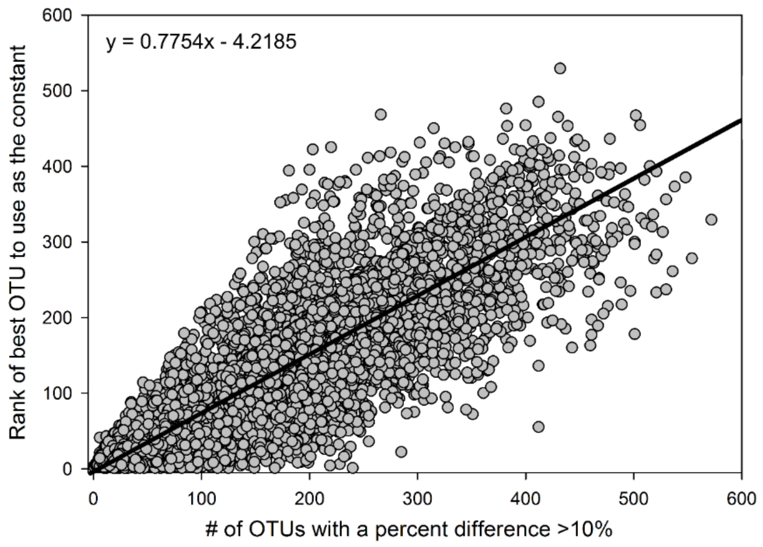


Figure 3: Results of simulations (8,100 iterations using various levels of contamination and numbers of OTUs) testing methods for identifying the best OTU to use as the constant.

The second equation (hereafter referred to as the “regression 2”) is more complicated. It has several different regression equations that it choses from based on the estimated number of OTUs that overlap between the blank and the sample (like regression 1, these regressions also use the number of OTUs with a percent difference >10% as x and the rank of percent differences as y). Therefore, it first attempts to calculate the number of overlapping OTUs, then it selects a regression based on the results of that calculation.

It does this by, once again, calculating the percent differences between proportions for the blank and the sample, but for the denominator for the sample, it uses the sum of the reads plus 0.3 times the sum of the reads. It then subsets to just the OTUs where that percent difference is greater than zero, then it does the same calculations again using only the reads from that subset of OTUs. The number of OTUs with a percent difference greater than zero from that second calculation closely correlates with the number of overlapping OTUs (Figure 4). Therefore, it uses the equation of a line from that regression and the results of the methods above to calculate the approximate number of overlapping OTUs (i.e., it takes the result of the method above and subtracts 17.99 then divides by 0.8246). Then it applies the appropriate regression based on that number. This is the method used by `decon.regress2()`.

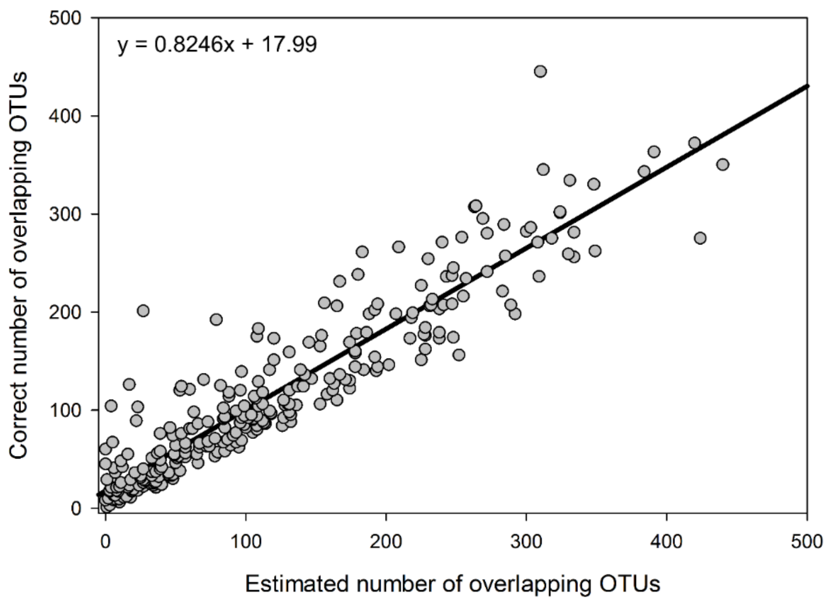


Figure 4: Correlation between the estimated number of overlapping OTUs and the actual number of OTUs (based on 270 iterations of a simulation).

We used a series of simulations to compare regression 1 and regression 2, and we found that regression 1 was more accurate in most cases, but regression 2 was more accurate when the estimated number of overlapping OTUs (calculated as above) was less than 40 or greater than 400 (Figure 5). Additionally, both equations generally worked best when they were run twice (i.e., they were used to decontaminate the data, then used a second time to decontaminate the data again). Using them only once often failed to remove some of the contamination, and using them three times or more removed reads that should have been retained (Figure 5, but see section 1.4.3).

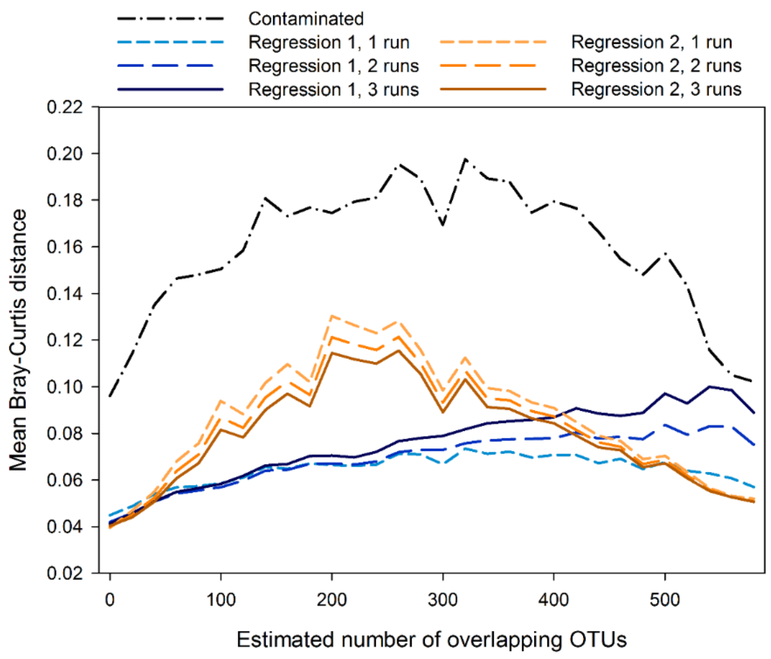


Figure 5: Results of simulations comparing regression 1 and regression 2 at one, two, and three runs each. The simulation was a modified version of simulation 1 in McKnight et al. 2018. For each run, the Bray-Curtis distances were calculated between the decontaminated sample and an uncontaminated sample (the distances for the contaminated sample are also shown). A total of 3,600 iterations were run across a wide range of scenarios. To produce this figure, those results were clustered into groups based on the estimated number of overlapping OTUs. Each group had a range of 20, and the tick marks on the figure show the top end of the range (e.g., the point at 100 includes any runs with an estimated number of OTUs between 81–100 inclusive).

Based on these results, we wrote `decon()` as the primary function for the package. Using the methods described above, it attempts to estimate the number of OTUs that overlap, then it uses either regression 1 (if the estimated number of overlapping OTUs is >/= 40 and </= 400 on default settings) or regression 2 (if the estimate number of overlapping OTUs is < 40 or > 400 on default settings). Additionally, it can do several runs of the regressions (default = 2). It evaluates which regression to use separately for each individual and each run within an individual (e.g., it may use regression 1 at first, then switch to regression 2 for the second round of decontamination within an individual).

Because regression 2 is only a slight improvement in situations with few overlapping OTUs and situations with an estimate of > 400 overlapping OTUs are very uncommon, `decon.regress1()` produces accurate results that are similar to the results of `decon()` in the majority of cases. However, on average, using `decon()` produces slightly more accurate results. We do not recommend manually selecting `decon.regress2()`.

1.4.3 Comparing numbers of runs

We used another simulation (built from simulation 1) to more closely examine the results of using different numbers of runs for `decon()` (Figure 6). The results show that the ideal number of runs increases as the contamination level increases (DNA yield in the contamination divided by DNA yield in an uncontaminated sample). Unfortunately, measuring the amount of contamination is difficult in actual studies. If possible, researchers should quantify the DNA in their blanks as well as the DNA in the samples immediately after extraction. Subtracting the DNA yield in the blanks from the DNA yield in the sample, then dividing by the DNA yield in the blanks should give a crude estimate of the amount of contamination. However, if it is not possible to do this, then researchers will have to use their best judgement to select the appropriate number of runs. Using just a single run was only best with low levels of contamination (< 0.15 contaminant yield/sample yield). Using two runs worked well over a broad range of realistic contamination levels, therefore we recommend it for most applications (it is the default). However, for highly contaminated samples (where the amount of contamination is suspected to be nearly equal to or greater than the amount of sample) it may be beneficial to do additional runs.

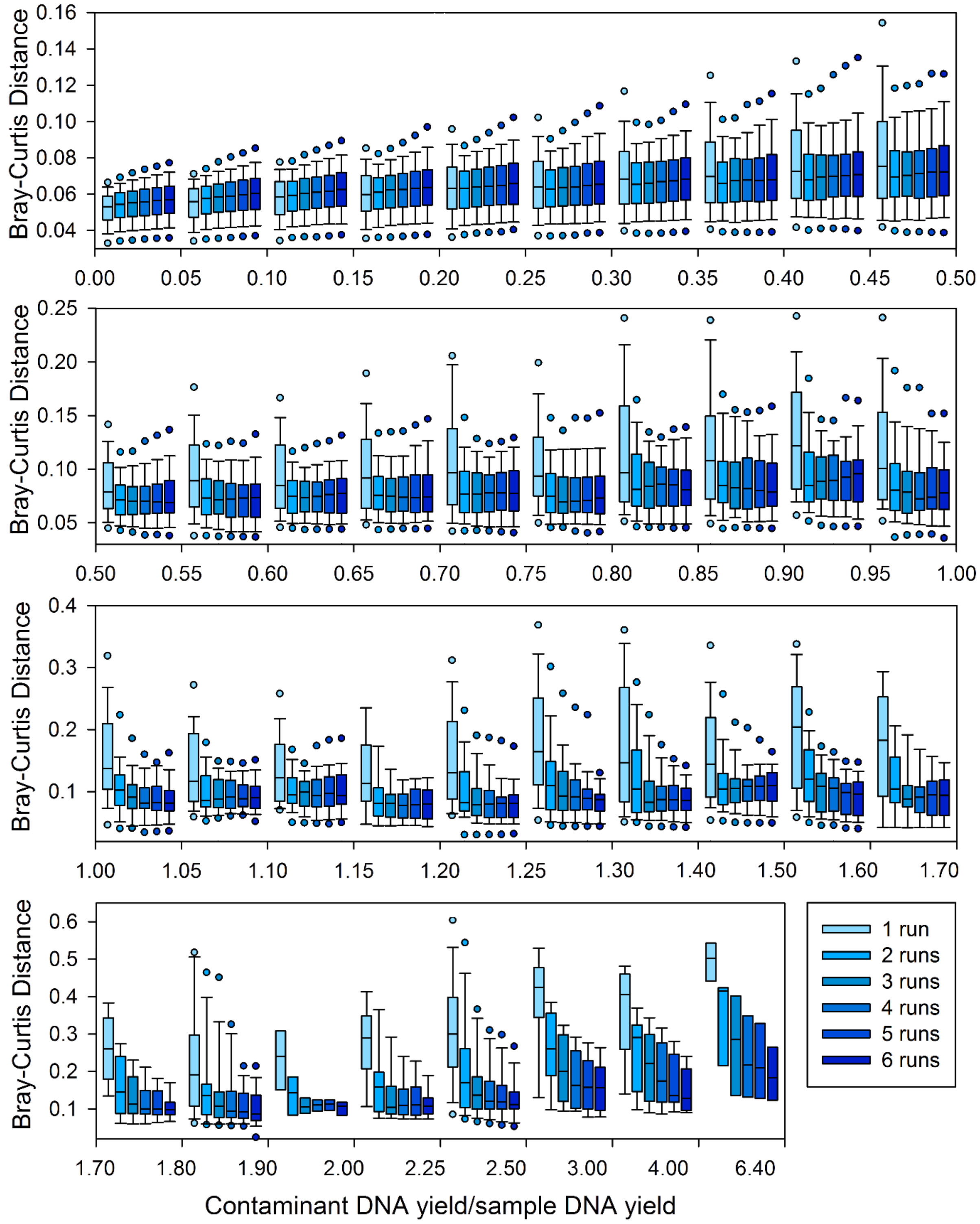


Figure 6: Results of simulations comparing numbers of runs. Results are Bray-Curtis distances between the decontaminated and uncontaminated sample (lower distance = more accurate results). For each iteration of the simulation (20,000 total), the same sample was decontaminated with each number of runs. To produce this figure, results were grouped based on the contamination level, and the tick marks on the X axis show the ranges of those groups (e.g., the first group of six boxes shows the data from iterations where the contamination was between 0–0.05. For higher levels of contamination, the bin width for the groups was increased because of low numbers of iterations. Thus, the scale of the X axis increases starting at the 1.30–1.40 group. Also note that the Y axis increases with each panel. Wickers represent the 10th/90th percentile, and for readability, the outliers simply represent the 5th/95th percentile.

1.4.4 Removing residual contamination (`remove.thresh()`)

Because the `decon()` functions treat each sample separately, sometimes an OTU that should have been removed from all samples ends up being retained in low numbers in a few samples even though it was removed from all of the others. Therefore, it may be desirable to apply filtering thresholds after decontamination to correct for those cases. We provided the `remove.thresh()` function for that purpose. It has two options for filtering, both of which are based on user-defined groups of samples (e.g., populations, host species, environments, etc.). The samples are grouped in this way to account for situations where an OTU is actually present in some groups, but not others. This can be circumvented by setting each individual as a group or by setting all individuals as a single group, but based on the results of our simulations (below), we do not recommend either of those options.

The first option for filtering is the `thresh` argument. It is based on the proportion of samples for which a given OTU is zero. It is set to a default of 0.7, meaning that if 70% or more of the samples in a group for a given OTU are zero, then the remaining samples in that group will also be set to zero for that OTU. The second option (the argument `prop.thresh`) is based on the proportion of reads for a given OTU within a given group of samples. It is set to a default of 0.00005, meaning that if a given OTU makes up less than 0.005% of the reads for a given group of samples, that OTU will be set to zero for all individuals in that group. When both options are utilized, it filters by `thresh` then by `prop.thresh`. Both arguments are run independently. Thus, an OTU will be set to zero if either condition is met, rather than if both conditions are met.

We used a simulation to test this function and establish optimal default settings. This simulation was based on simulation 2 in McKnight et al. 2018, but instead of returning Bray-Curtis distances between the populations, it returned the number of OTUs that were correctly assigned as present or absent from a population (i.e., they were scored as correct if they were either present in at least one uncontaminated sample and at least one decontaminated sample or absent in all uncontaminated and all decontaminated samples).

We used this simulation to run 100 iterations with two populations of five individuals each, 100 with two populations of 10 individuals each, and 100 with two populations of 20 individuals each. Each iteration tested all pairwise comparisons of the following settings for the two filtering arguments: `thresh` = 0.6, 0.7, 0.8, 0.9, 1, `prop.thresh` = 0, 0.0005, 0.0001, 0.00005, 0.00001. Setting `thresh` to 1 and `prop.thresh` to 0 shuts off those arguments. Additionally, we compared the results when each population was set as a separate group (recommended), when all individuals were set as a single group, and when each individual was set as its own group. We judged the success of a run based on the percent of contaminated OTUs that were correctly identified as present or absent in the decontaminated population.

We found that entering each population as a group worked better than treating each individual separately or treating all samples as a single group (Figure 7). Additionally, setting the `thresh` argument to 0.7 and the `prop.thresh` argument to 0.00005 (their defaults) produced the best results (Figure 8). Finally, as expected, the benefits of using the `remove.thresh()` function increased as the population size increased. This is simply because the probability that an OTU will be erroneously retained in at least one sample increases as the number of individuals increases.

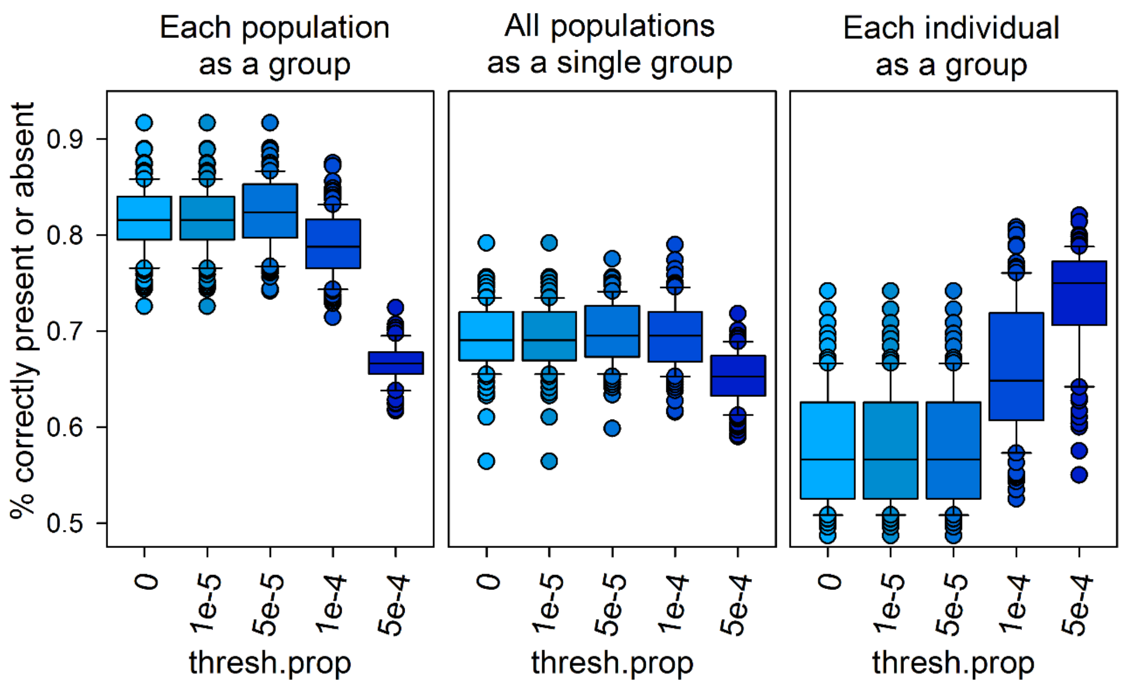


Figure 7. Results comparing different methods for assigning groups for the `remove.thresh()` function (`thresh` = 0.7). Each box within a panel is a different setting for the `thresh.prop` argument. The Y axis shows the percent of contaminant OTUs that were correctly identified as present or absent (i.e., an OTU was scored as correct if it was either present in at least one uncontaminated sample and at least one decontaminated sample or if it was absent in all uncontaminated and all decontaminated samples). Results are shown for population 1 from simulations with 20 individuals per population (results for population 2 and simulations with 5 or 10 individuals were similar). Whiskers represent the 90th and 10th percentile, and all outliers are shown.

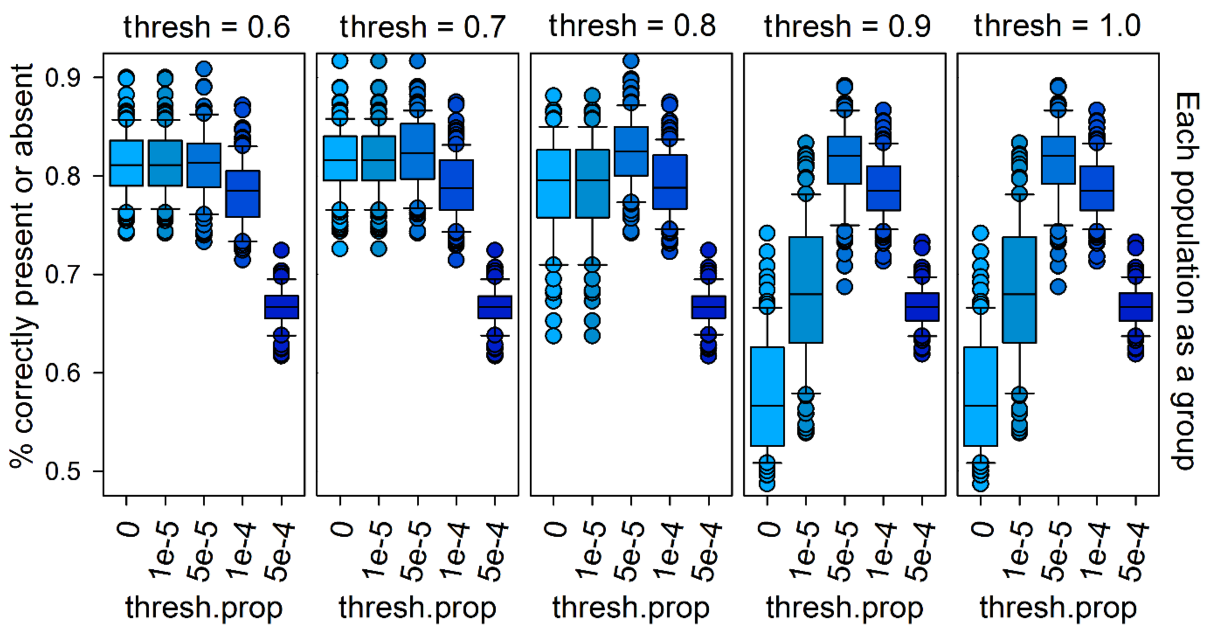


Figure 8. Results from simulations comparing different settings for the `remove.thresh()` function. Each column of panels is a different setting for the `thresh` argument, and each box within a panel is a different setting for the `thresh.prop` argument. Setting `thresh` to 1.0 and `thresh.prop` to 0 shuts off each argument. Thus, the first box of the last panel shows the results if the `remove.thresh()` function is not used at all. The Y axis shows the percent of contaminant OTUs that were correctly identified as present or absent. Results are shown for population 1 from simulations with 20 individuals per population (results for population 2 and simulations with 5 or 10 individuals were similar). Whiskers represent the 90th and 10th percentile, and all outliers are shown.

.

1.4.5 Effects of multiple blanks

microDecon allows the use of multiple blanks for determining the amount of contamination present. Therefore, we tested the results from using several combinations of blanks. To do this, we used the data from the experiment described in McKnight et al. 2018, and we compared the results of decontaminating the data 1) using each blank by itself, 2) using each pairwise combination of two blanks, 3) using each possible combination of three blanks, and 4) using all four blanks. Heterogeneity was present among the blanks, so the results varied depending on which blank or combination of blanks was used (Figure 9; Table 1). Nevertheless, using any blank or any combination of blanks was reasonably effective, but averaging all four blanks appeared to be the best solution (Figure 10). Therefore, we recommend that researchers include multiple blanks and average them prior to using *microDecon*.

In cases where sequencing depth requirements prevent researchers from actually sequencing multiple blanks, we recommend that they extract and amplify DNA for several blanks, then pool those samples prior to indexing (this will not account for heterogeneity from sequencing, but it should account for heterogeneity from extraction and amplification).

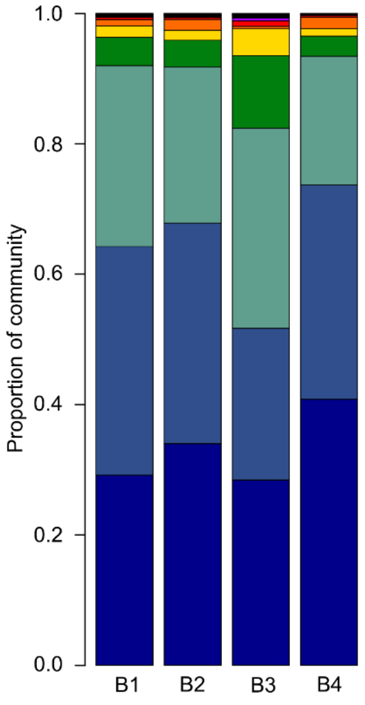


Figure 9. Proportions of OTUs in the four blanks. Seventy-four OTUs were present, but most were very unabundant and cannot be seen at this resolution

Table 1. Bray-Curtis distances for pairwise comparisons of the four blanks.

	B1	B2	B3	B4
B1		0.059	0.136	0.126
B2	0.059		0.179	0.070
B3	0.136	0.179		0.237
B4	0.126	0.070	0.237	

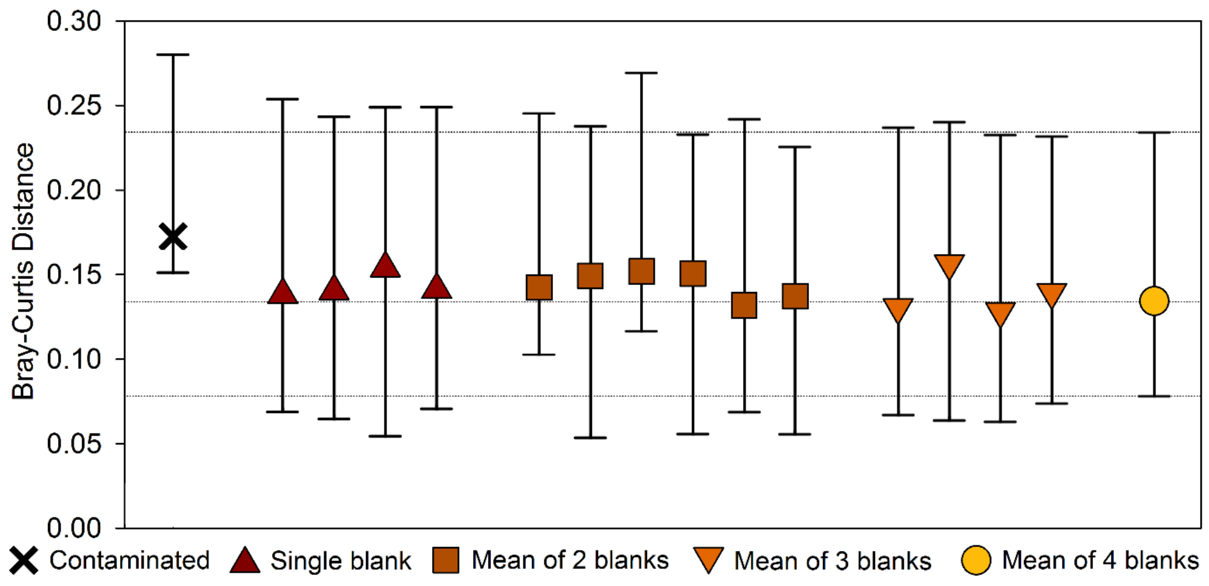


Figure 10. Bray-Curtis distances (decontaminated vs uncontaminated) using different combinations of blanks to decontaminate them. Symbols are the median from all eight samples, and error bars show the maximum and minimum values. The results for the contaminated samples are also shown for comparison. The dotted lines follow the maximum, median, and minimum for the mean of all four blanks for easy comparison.

2.0.0 Using the package

2.1.0 Installing the package

microDecon can be installed from Github via the following address:

https://github.com/donaldtmcknight/microDecon

For easy install, use the devtools package. Use the following commands in R (skip the first step if devtools is already installed

```
install.packages("devtools")
library(devtools)
devtools::install_github(“donaldtmcknight/microDecon)
```

2.2.0 Recommended functions and settings

We recommend that you use the `decon()` function followed by the `remove.thresh()` function on their default values. Individuals in the `remove.thresh()` function should be grouped by population ID, species, or some other sensible grouping criteria. Please see sections 1.4.2 and 1.4.4 for explanations and justifications for those recommendations. Additionally, we recommend including several blanks in the analysis (see 1.4.5).

2.3.0 Input

All *microDecon* functions take an OTU table of metabarcoding data structured as a data frame where each row is an OTU, each column is an individual sample, and each cell contains the number of reads for a given OTU for a given individual. Additionally, the first column should contain OTU IDs, and the last column should (optionally) contain taxonomic information. The second column should contain the reads from a blank sample. For all functions except `remove.thresh()`, you should ideally include several blanks, which should be entered as consecutive columns starting in column 2. These will be averaged by *microDecon* to produce a mean blank that is used in the analyses. Data for `remove.thresh()` should only include one blank column, and individuals should be sorted into groups (e.g., populations).

`Example_1`: A data frame with three blank columns, three columns of data from individuals, and a taxa column (this is formatted correctly for `decon()`, `decon.regress1()`, or `decon.regress2()`). The format of the taxa column is irrelevant, and the column itself is optional. This table should be loaded such that the first row becomes column IDs; however, the first column should be retained as a column (i.e., do not make the OTU IDs row names).

OTU_ID	Blank1	Blank2	Blank3	Ind1	Ind2	Ind3	Taxa
OTU1	0	100	50	100	120	60	k_Fungi
OTU2	200	0	25	50	60	20	k__Fungi
OTU3	1000	1300	1500	1000	1200	1400	k__Fungi; p__Ascomycota
OTU4	50	10	20	2	4	3	k__Fungi; p__Basidiomycota
OTU5	0	0	0	500	400	600	k__Fungi; p__Basidiomycota

`Example_2`: A data frame with a single blank, ten columns of data from individuals, and no taxa column. This is formatted correctly for any *microDecon* function.

OTU_ID	Blank	Ind1	Ind2	Ind3	Ind4	Ind5	Ind6	Ind7	Ind8	Ind9	Ind10
OTU1	0	50	40	60	100	120	60	20	40	100	60
OTU2	200	50	60	20	50	60	20	20	50	40	30
OTU3	1000	1000	1200	1400	1000	1200	1400	1000	800	1200	1400
OTU4	50	2	4	3	2	4	3	1	0	5	0
OTU5	0	500	400	600	500	400	600	200	300	400	250

Important formatting points:

- First column is OTU IDs
- Blanks are in consecutive columns starting in column 2
    - `decon()`, `decon.regress1()`, and `decon.regress2()` can include multiple blanks, `remove.thresh()` cannot.
- Columns of individual data start after the last blank column
- The final column is an optional taxa column that has no specific formatting requirements other than that it consists of only one column.
- For `remove.thresh()` individuals should be grouped into consecutive columns based on populations, collection sites, or some other sensible, *a priori* grouping criteria.

2.4.0 Running the functions

2.4.1 `decon()`

This is the primary function for removing contamination and it is recommended that you use it on its default settings (see section 1.4.2).

Arguments:

`data`	`A data frame of metabarcoding read data consisting of at least 3 columns in this order: a column of unique OTU names/labels, at least one column of read data from a blank sample (this contains your known contaminant reads), at least one column of read data for an actual sample (each column is a sample, each row is an OTU, and each cell is the number of reads). It can optionally include a final column with taxonomy information. If multiple blanks are included (recommended), they must be in consecutive columns, starting with column 2.`
`numb.blanks`	`Numeric (default = 1). Specifies the number of blanks included in the data set (if multiple blanks are included, they must be in consecutive columns, starting with column 2).`
`taxa`	`Logical (T/F). Specifies whether or not the last column contains taxonomic information (default = T)`
`runs`	`Numeric (default = 2). Specifies the number of times that the function should run the decontamination procedure on the data. Based on simulation results, using two runs is best on average, but using one run is better if there is very little contamination, and using more than two runs is better if there is substantial contamination (see 1.4.3).`
`low.threshold`	`Numeric (default = 40). Selects the lower point for switching between the two decon functions (decon.regress1() and decon.regress2()). It uses decon.regress2 anytime that the estimated overlap is <low.threshold or >up.threshold. It is usually best not to change this value.`
`up.threshold`	`Numeric (default = 400). Selects the higher point for switching between the two decon functions (decon.regress1() and decon.regress2()). It uses decon.regress2 anytime that the estimated overlap is <low.threshold or >up.threshold. It is usually best not to change this value.`

Example 1 (on default settings, using the `Example_1` data frame from section 2.3.0):

```
> result1 <- decon(data = Example_1, numb.blanks = 3, taxa = T)
```

Example 2 (on default settings, using the `Example_2` data frame from section 2.3.0):

```
> result2 <- decon(data = Example_2, numb.blanks = 1, taxa = F)
```

2.4.2 `decon.regress1()`

This function uses regression 1 for decontaminating samples (see section 1.4.2). Its use is identical to `decon()` except it can only do a single run, and because it does not select between algorithms, there are no `up.threshold` or `low.threshold` arguments. It is generally recommended to us `decon()` rather than `decon.regress1()`.

If you choose to use `decon.regress1` and you want to do multiple runs, simply call the function over the results of the previous run. Keep in mind, however, that if multiple blank columns are entered, a single mean blank column will be returned. Therefore, all runs except the first one will only have a single blank.

Example (doing two runs using the Example_1 data frame from section 2.3.0):

```
> result_1run <- decon.regress1(data = Example_1, numb.blanks = 3,
  taxa = T)
> result_2runs <- decon.regress1(data = result_1run, numb.blanks =
  1, taxa = T)
```

2.4.3 `decon.regress2()`

This function uses regression 2 for decontaminating samples (see section 1.4.2). Its use is identical to `decon()` except it can only do a single run, and because it does not select between algorithms, there are no `up.threshold` or `low.threshold` arguments. It is generally recommended to us `decon()` rather than `decon.regress2()`.

If you choose to use `decon.regress2` and you want to do multiple runs, simply call the function over the results of the previous run. Keep in mind, however, that if multiple blank columns are entered, a single mean blank column will be returned. Therefore, all runs except the first one will only have a single blank.

Example (doing two runs using the Example_1 data frame from section 2.3.0):

```
> result_1run <- decon.regress2(data = Example_1, numb.blanks = 3,
   taxa = T)
> result_2runs <- decon.regress2(data = result_1run, numb.blanks =
   1, taxa = T)
```

2.4.4 `remove.thresh()`

This function removes residual contamination on the output from the `decon()` functions.

Arguments:

data	A data frame of metabarcoding read data consisting of at least 3 columns in this order: a column of unique OTU names/labels, a column of read data from a blank sample (this contains your known contaminant reads; do not include multiple blank columns; include only the mean blank column), at least one column of read data for an actual sample (each column is a sample, each row is an OTU, and each cell is the number of reads). It can optionally include a final column with taxonomic information. If multiple groups are present, order all your sample columns so that groups are together, and include the corresponding number of individuals per group in numb.ind.
taxa	Logical (T/F). Specifies whether or not the last column contains taxonomic information (default = T)
numb.ind	A vector of numbers listing the number of individuals in each user-specified group (e.g., different populations could be treated as different groups). Data must be sorted by these groups beforehand. To apply the thresholds across all individuals as a single group, make a vector with a single number that is the total number of individuals. Alternatively, to treat each sample independently, you can set each individual as a group. Do this by making a vector with the number 1 repeated as many times as you have samples (e.g., if you have 20 samples, you can make this as follows: my.vector <- rep(1,20)). If each individual is treated separately, then only the prop.thresh argument will be used for filtering.
thresh	Numeric (default = 0.7). A number written as a proportion. This is the threshold at which if that proportion of 0s are present for an OTU within a group, all samples will be set to 0 for that OTU for that group (e.g., if thresh = 0.7, then if, for a particular OTU, 70 percent of samples are 0 within a group, all become 0 for that OTU). The threshold always rounds down (i.e. if thresh = 0.7 and there are 11 samples, then any OTU with 7 or more 0s will become 0 for all samples in that group). It will not do anything to groups with four or fewer samples. Set to 1 if you don't want to apply this threshold.

`prop.thresh`	`Numeric (default = 0.00005). A number written as a proportion. This is the threshold at which if the number of reads for a particular OTU are below this proportion (based on all reads for a group), the OTU will be set to zero for all individuals in that group (e.g., if a particular OTU makes up 0.001% of all of the reads for a group, then at prop.thresh = 0.00005, that OTU would be set to 0 for all individuals in the group [0.00005 = 0.005%]). Set to 0 if you do not want to use this threshold.`

Examples: The following examples will use the `Example_2` data frame from section 2.3.0, where individuals 1–6 are a population and individuals 7–10 are a population. Assume that these data were run through `decon()` on default settings as shown below.

```
> result2 <- decon(data = Example_2, numb.blanks = 1, taxa = F)
```

Example: Default settings (other than taxa), with each population treated separately.

```
> result2.thresh <- remove.thresh(data = result2, taxa = F,
numb.ind = c(6,4))
```

Example: Using `thresh` argument only, with each population treated separately.

```
> result2.thresh <- remove.thresh(data = result2, taxa = F,
numb.ind = c(6,4), thresh = 0.7, prop.thresh = 0)
```

Example: Use `prop.thresh` argument only, with each population treated separately.

```
> result2.thresh <- remove.thresh(data = result2, taxa = F,
numb.ind = c(6,4), thresh = 1, prop.thresh = 0.00005)
```

Example: All individuals treated as a single population.

```
> result2.thresh <- remove.thresh(data = result2, taxa = F,
numb.ind = c(10))
```

Example: Each individual treated separately.

```
> result2.thresh <- remove.thresh(data = result2, taxa = F,
numb.ind = c(rep(1,10)))
```

2.5.0 Output

All *microDecon* functions output a data frame that is structured the same as the input data frame. However, if several blanks were input, the output will include only a single "mean blank" column that is the mean of those blanks. Additionally, the order of the rows may be different.

2.6.0 Important points

1. This package is only as good as the data you put into it. For it to be effective, several steps must be taken during data collection
    a. Blanks need to be collected at the same time that the actual samples are collected, and they must be processed identically to the actual samples (this includes extraction, PCR, standardization, and sequencing. Do not simply use no template controls for your blanks, because they will not account for contamination from collection or extraction.
    b. If possible, collect and use several blanks. If sequencing costs prevent you from sequencing all of them, then pool them prior to the indexing PCR and sequence a single blank.
    c. Standardize everything. If at all possible, use the same batches of vials, reagents, etc. If several batches are required, then randomize your samples across the batches and include separate blanks for each batch. After sequencing, compare the blanks, and if they are indistinguishable from each other, then you can put all of the data together and use `decon()` over the entire data set (including all blanks). If the blanks from different batches consistently differ, however, then use `decon()` separately on each batch.
    d. Randomize samples across batches, PCR runs, etc.
2. Use your usual data filtering and quality control steps prior to using *microDecon*, but do not normalize, transform, or otherwise manipulate your read data prior to using *microDecon.*
3. In some cases, there may be OTUs which you know or suspect are entirely contamination. Do not remove these prior to using *microDecon* because they are likely the best constants. Leave them in and check them after using the package. If the package worked on your data (and you were correct about them being entirely contamination) they should be removed (set to zero) or at least greatly reduced.
4. The `decon()` function generally produces more accurate results than manually selecting a regression (i.e., using `decon.regress1()` or `decon.regress2()`), so we recommend using it on default settings.
5. Setting `decon()` to two runs (the default) is generally best. A single run tends to underestimate the amount of contamination that is present, while three or more runs tends to overestimate it. However, using a single run may be desirable with very low levels of contamination, and using three or more runs may be desirable with high levels of contamination.
6. For the `remove.thresh()` function, group individuals into sensible *a priori* clusters based on species, populations, environments, etc. The idea is that some OTUs might be present but rare in one group and absent in another group, and if you do not put individuals into groups for this function, then those OTUs will erroneously be set to zero for all individuals. Thus, groups should represent the smallest level at which you expect there to be consistent differences.