Package 'microDecon'
A user's guide

Version 1.0.2

11-April-19

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 operational taxonomic units (OTUs), it preserves data from OTUs that are present in both the contamination and the system being studied. 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 that you use the `decon()` function on its default values (this runs both `remove.cont()` and `remove.thresh()`). This package and manual were originally written before the release of QIIME 2 and the advent of "amplicon sequence variants" (ASVs). Therefore, we continue to refer to OTUs throughout, but the package works equally well for ASVs or other outputs from metabarcoding studies.

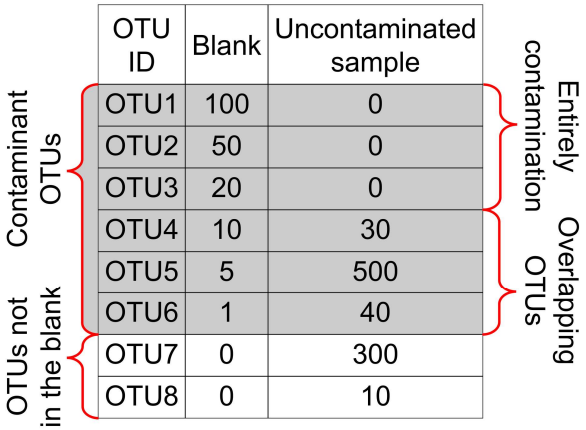
1.2.0 Citing this package

Please cite this package as McKnight, D. T., R. Huerlimann, D. S. Bower, L. Schwarzkopf, R. A. Alford, and K. R. Zenger. 2019. microDecon: A highly accurate read-subtraction tool for the post-sequencing removal of contamination in metabarcoding studies. Environmental DNA. https://doi.org/10.1002/edn3.11

To report bugs or ask questions that are not answered in the manual or associated paper, please contact donald.mcknight@my.jcu.edu.au

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

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 (referred to as a "constant"), we can use the ratios of the other contaminant OTUs relative to it to determine the number of reads that are from contamination, then subtract those from the sample. By relying on ratios of OTUs relative to each other, differences in read depths and starting material among samples do not adversely affect *microDecon*.

As an example, consider a sample and blank with two OTUs that amplified in the blank. In the blank, OTU1 has 1000 reads, and OTU2 has 100 reads. Thus, the ratio for those OTUs in the contamination is 10:1. If we also know that one of those OTUs is entirely contamination (i.e., should not be present in the sample), we can use that to determine the number of reads in the sample that are from contamination for both OTUs. If, for example,

we know that OTU1 is entirely contamination, and in the sample, OTU1 has 600 reads while OTU2 has 100 reads, we can deduce that all 600 reads for OTU1 are from contamination and, based on the 10:1 ratio in the blank, 60 of the reads in OTU2 are from contamination. Therefore, a decontaminated sample would have zero reads for OTU1 and 40 reads for OTU2. *microDecon* accomplishes this using the steps outlined below (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.

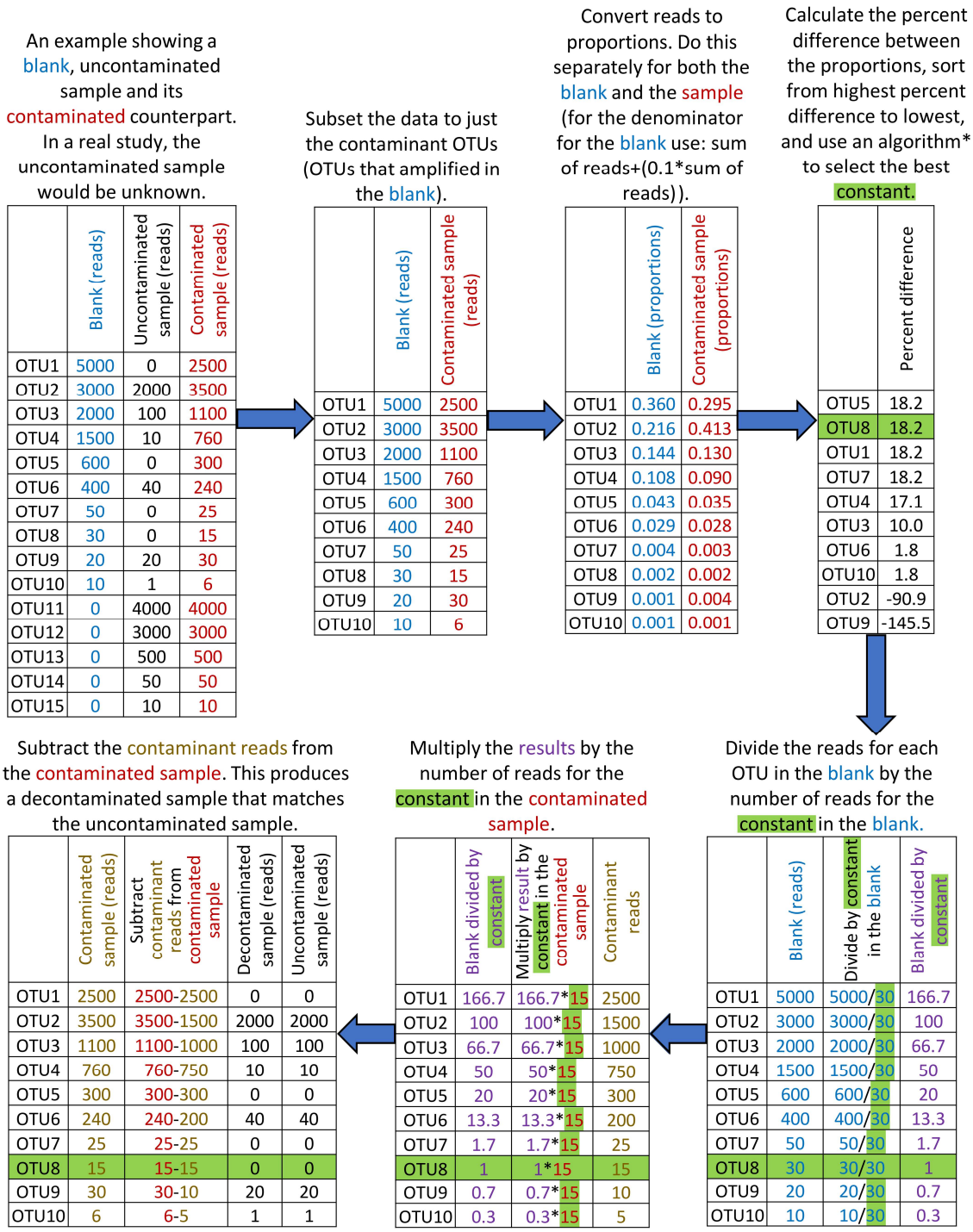
Next, it subsets the data to just the blank and a single actual sample. This process is iterative and treats each sample completely independently of every other sample. In other words, the following steps are iteratively looped over each sample.

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] for regression 1, and using all OTUs for regression 2; see section 1.4.2), but a slight modification is used for the sample. 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 is done because if the sum + 10% of the sum is not used, then in cases where there are no overlapping OTUs, the percent differences (calculated bellow) will 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 (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.

We have rigorously tested this method using an experimental data set, multiple *in silico* tests, and a 16S data set. We also compared it to existing methods for removing contamination (e.g., decontam). Please see our paper on this method (and its appendices) for results of these tests (McKnight, D. T., R. Huerlimann, D. S. Bower, L. Schwarzkopf, R. A. Alford, and K. R. Zenger. (in press) microDecon: A highly accurate read-subtraction tool for the post-sequencing removal of contamination in metabarcoding studies. Environmental DNA).



	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

	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

	Blank (proportions)	Contaminated sample (proportions)
OTU1	0.360	0.295
OTU2	0.216	0.413
OTU3	0.144	0.130
OTU4	0.108	0.090
OTU5	0.043	0.035
OTU6	0.029	0.028
OTU7	0.004	0.003
OTU8	0.002	0.002
OTU9	0.001	0.004
OTU10	0.001	0.001

	Percent difference
OTU5	18.2
OTU8	18.2
OTU1	18.2
OTU7	18.2
OTU4	17.1
OTU3	10.0
OTU6	1.8
OTU10	1.8
OTU2	-90.9
OTU9	-145.5

	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

	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

	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. Percent difference is calculated as: ([blank proportion – sample proportion]/ blank proportion) *100. Some numbers reported in the 4th table are slight deviations of the expected values based on the 3rd table. This is simply a result of rounding error from rounding the values in the 3rd table to four decimal places. 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. in review). 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. The results of the simulation that produced that regression equation are shown in Figure 3.

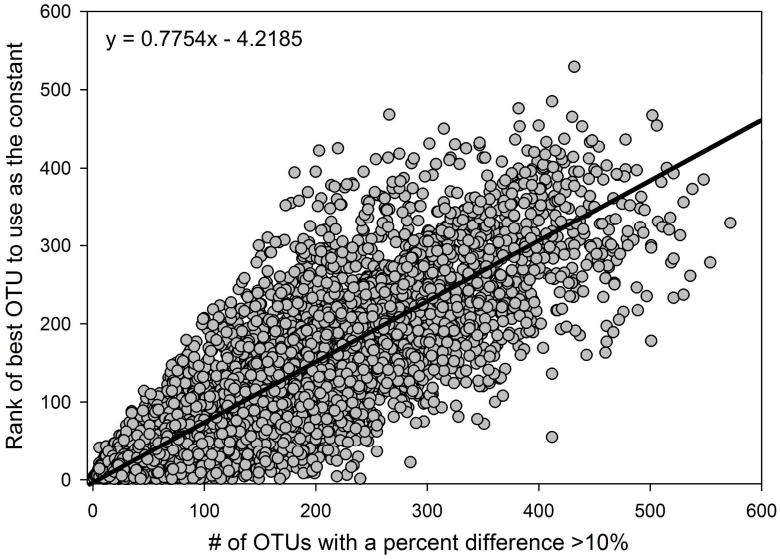


Figure 3: Results of simulations (8,100 iterations using various levels of contamination and numbers of OTUs) testing methods for identifying the best OTU to use as the constant.

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. Additionally, unlike regression 1, the proportions are based on all reads in a sample, not just the reads for contaminant OTUs. 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.

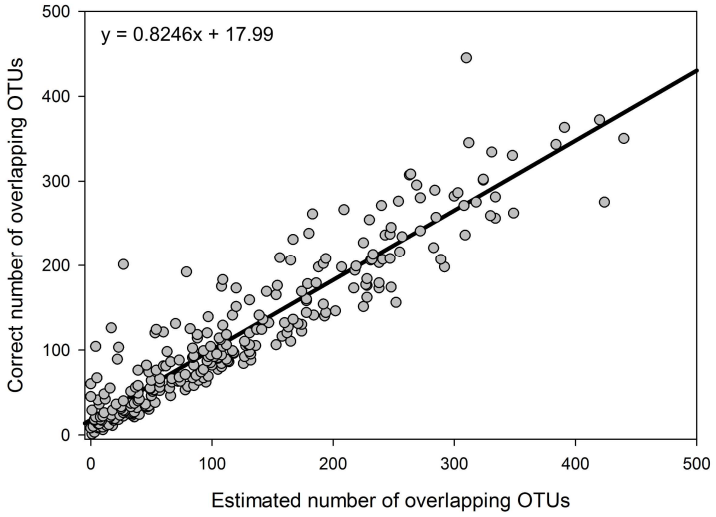


Figure 4: Correlation between the estimated number of overlapping OTUs and the actual number of OTUs (based on 270 iterations of a simulation).

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).

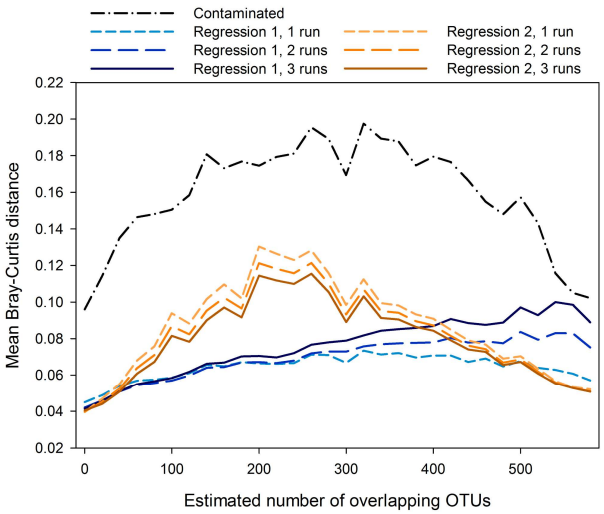


Figure 5: Results of simulations comparing regression 1 and regression 2 at one, two, and three runs each. The simulation was a modified version of simulation 1 in McKnight et al. in review. For each run, the Bray-Curtis distances were calculated between the decontaminated sample and an uncontaminated 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 recommend using regression 1 if the estimated number of overlapping OTUs is >/= 40 and </= 400 and regression 2 if the estimate number of overlapping OTUs is < 40 or > 400. On their default settings (`regression = 0`), the `remove.cont()` and `decon()` functions select between the regressions automatically for each sample, and if multiple runs are used, then the choice of regression is evaluated independently for each run. This can be changed by setting `regression = 1` (which always uses regression 1) or `regression = 2` (which always uses regression 2). Additionally, if `regression = 0`, the thresholds for when it switches between regressions can be changed. In general, we recommend using the default settings; however, regression 1 generally produces accurate results because its results are similar to those of regression 2 when few overlapping OTUs are present, and situations with very large numbers of overlapping OTUs are rare, and a case could be made for using a simpler algorithm that is not influenced by factors like total reads in a sample. Therefore, using only regression 1 is acceptable, but we do not recommend setting `regression = 2` because this is inaccurate for many data sets (Figure 5).

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 (Figure 6). The results show that the ideal number of runs increases as the contamination level increases (contamination level = 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.

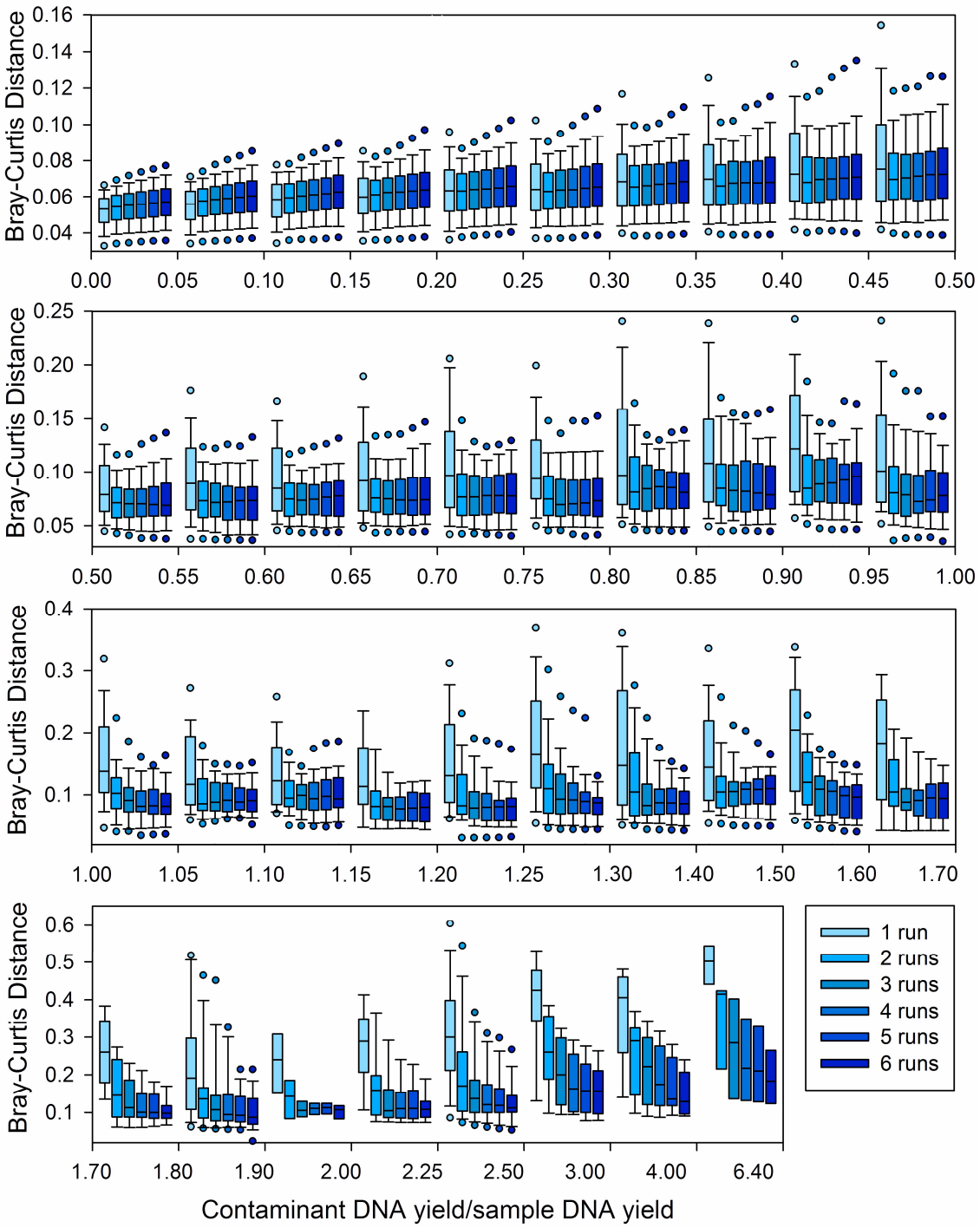


Figure 6: Results of simulations comparing numbers of runs. Results are Bray-Curtis distances between the decontaminated and uncontaminated sample (lower distance = more accurate results). For each iteration of the simulation (20,000 total), the same sample was decontaminated with each number of runs. 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 $10^{th}/90^{th}$ percentile, and for readability, the outliers simply represent the $5^{th}/95^{th}$ percentile.

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

Because the `decon()` and `remove.cont()` 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. As with all *microDecon* functions, only OTUs that amplified in the blank are affected.

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 all reads that belong to a given OTU within a given group of samples (i.e., all reads from all OTUs in that group [including OTUs that did not amplify in the blank] are used for calculating these proportions; however only OTUs that amplified in the blank are actually modified). 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. in review, but instead of returning Bray-Curtis distances between the groups, it returned the number of OTUs that were correctly assigned as present or absent from a group (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.

It should be noted that the default value for `prop.thresh` was obtained using simulations where the total number of reads per group varied from 90,000–400,000; however, when the total number of reads are lower than that, a more stringent threshold will be required to detect OTUs that should be set to zero (e.g., with 10,000 reads and `prop.thresh` = 0.00005 [default], OTUs would only be set to zero if they had 0.5 reads, which is impossible; whereas a threshold of 0.0005 would remove OTUs with 5 reads or fewer, which is reasonable and should be accurate based on simulations; Figure 7).

Additionally, please note that while setting `thresh` to 0.7 (default) worked best in the widest range of situations, there were exceptions, particularly when samples were very consistent (i.e., OTUs in the samples prior to contamination were reliably present or absent). In that case, a more stringent threshold (like 0.6) may slightly improve results. Similarly, highly contaminated samples may require more stringent thresholds (but keep the false positive rate in mind).

It is imperative that you use appropriate *a priori* groupings to avoid removing OTUs that should be retained. If, for example, you have a population of 20 individuals and a population of 80 individuals, but you include them as a single population, you may inadvertently remove OTUs that were only present (or only common) in the smaller population.

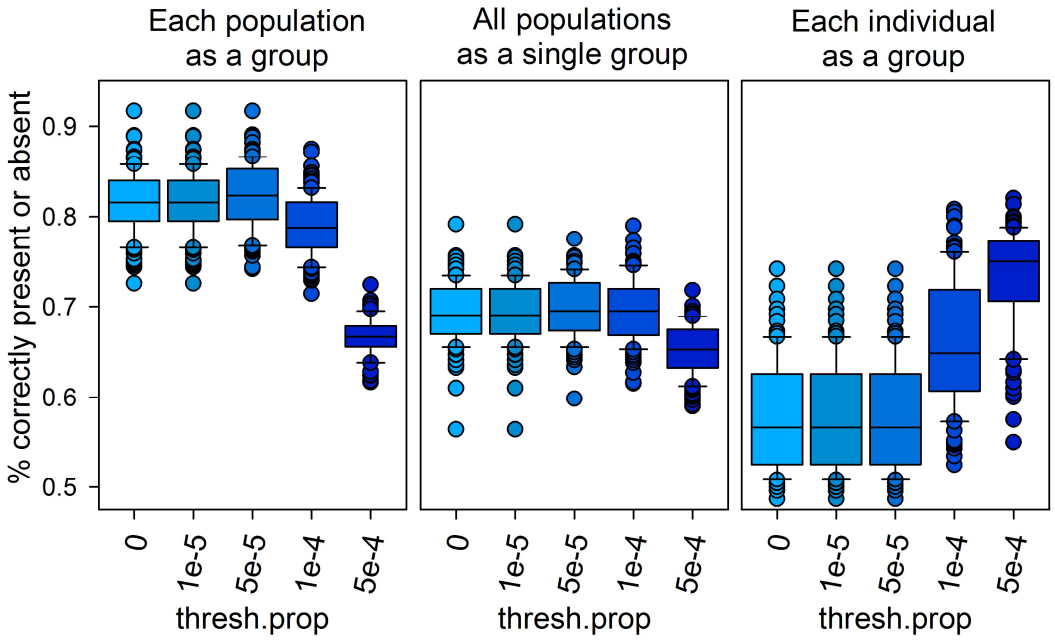


Figure 7. Results comparing different methods for assigning groups for the `remove.thresh()` function (`thresh` = 0.7). Each box within a panel is a different setting for the `thresh.prop` argument. The Y axis shows the percent of contaminant OTUs that were correctly identified as present or absent (i.e., 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.

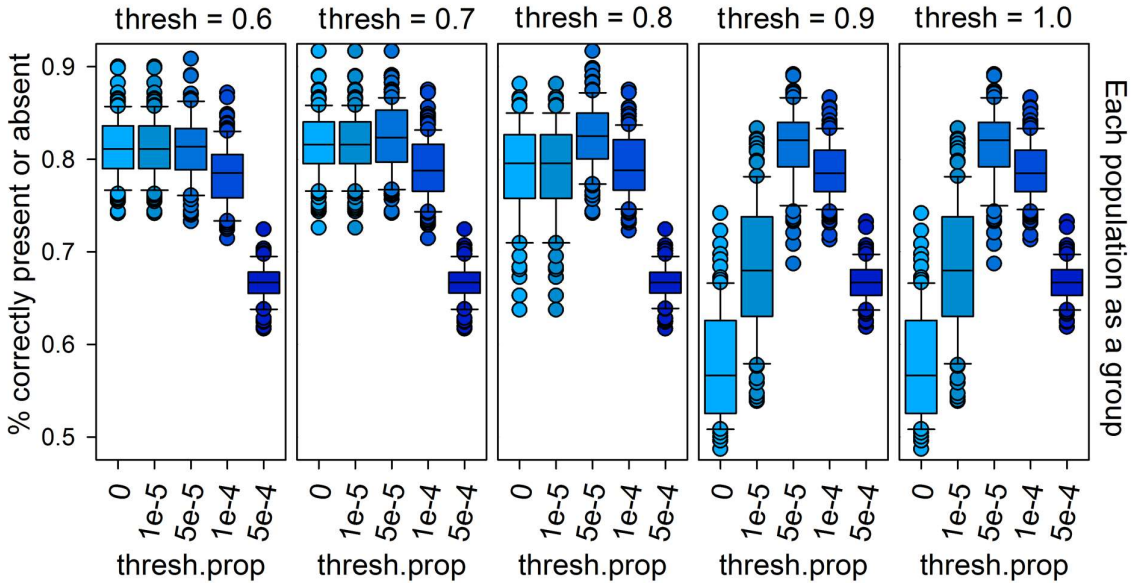


Figure 8. Results from simulations comparing different settings for the `remove.thresh()` function. Each column of panels is a different setting for the `thresh` argument, and each box within a panel is a different setting for the `thresh.prop` argument. Setting `thresh` to 1 and `thresh.prop` to 0 shuts 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. in review, 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 among blanks was fairly low (Figure 9; Table 1) and 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. *microDecon* converts the reads in each blank to proportions, then uses the average of those proportions.

Note: *microDecon* assumes that all samples received similar proportions of contaminants; therefore, it will not be accurate if blanks are highly heterogeneous. It is strongly recommended that you use multiple blanks and check them for heterogeneity. If blanks are strongly heterogeneous, you may need to subset the data based on the blanks (e.g., if you have three batches of samples with one blank each, and the blanks are heterogeneous, then use *microDecon* separately on each batch). Heterogeneity among blanks can be tested used stacked bar plots (e.g. Figure 9), ordination plots, and Bray-Curtis distances (Table 1). Note that the results below are from an experiment where samples and blanks were deliberately contaminated. Therefore, they provide a reasonable baseline for stochastic variation as opposed to actual differences in sources of contamination.

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).

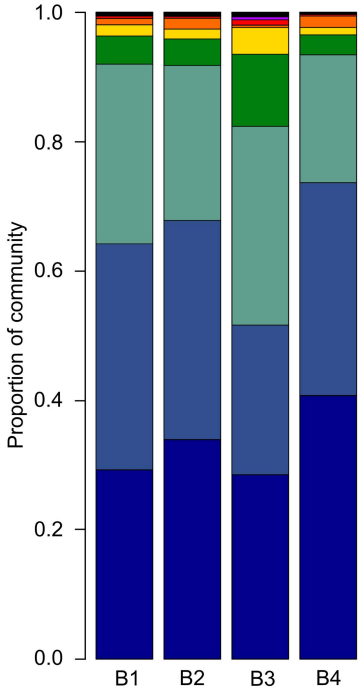


Figure 9. Proportions of OTUs in the four blanks. Seventy-four OTUs were present, but most were very unabundant and cannot be seen at this resolution

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	

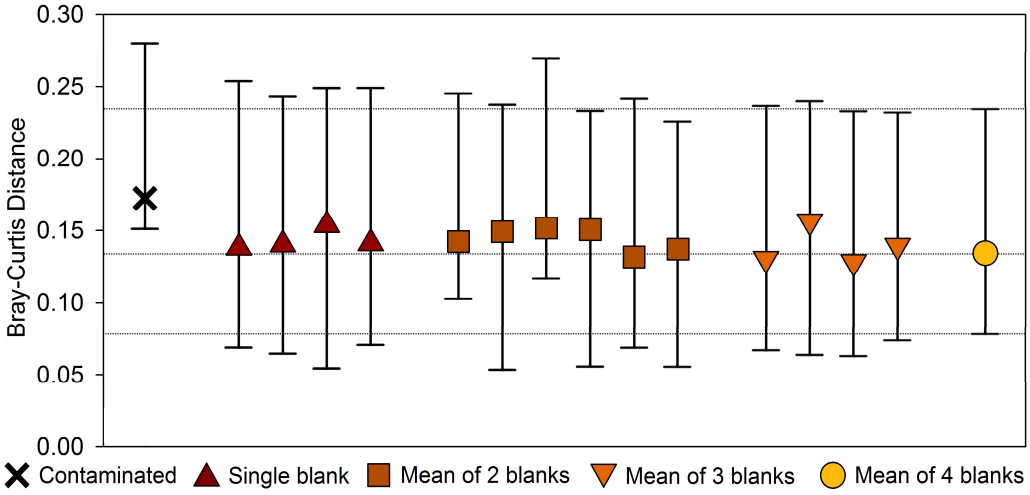


Figure 10. Bray-Curtis distances (decontaminated vs uncontaminated samples) using different combinations of blanks to decontaminate them. Symbols are the median from all eight samples, and error bars show the maximum and minimum values. The results for the contaminated samples are also shown for comparison.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

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 generally recommend that you use the `decon()` function on its default values (this runs both `remove.cont()` and `remove.thresh()`); however, at times it may be necessary to adjust the `runs`, `prop.thresh`, and `thresh` arguments (see sections 1.4.3 and 1.4.4). Individuals should be grouped by population ID, species, or some other sensible *a priori* grouping criteria (i.e., treat these groups as experimental blocks). Additionally, we recommend including several blanks in the analysis (see 1.4.5).

2.3.0 Input

All *microDecon* functions take a table of metabarcoding data structured as a data frame where each row is an OTU (or ASV or other metabarcoding output), 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 (these can be numeric or characters), and the last column should (optionally) contain taxonomic information. The second column should contain the reads from a blank sample. Ideally, 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.

`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()` or `remove.cont()`). The format of the taxa column is irrelevant, and the column itself is optional. Data frames 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). In this example, OTU2 is present as both contamination and actual reads, whereas OTUs 3, 4, and 6 are entirely contamination. This data frame loads as part of the *microDecon* package and can be accessed by entering `Example_1`.

OTU_ID	Blank1	Blank2	Blank3	Ind1	Ind2	Ind3	taxa
OTU1	0	0	0	60	64	40	K_Bacteria; P_Actinobacteria
OTU2	200	220	180	660	520	480	K_Bacteria; P_Proteobacteria
OTU3	1000	800	1300	1440	1000	700	K_Bacteria; P_Proteobacteria
OTU4	50	30	70	70	48	35	K_Bacteria; P_Bacteroidetes
OTU5	0	0	0	2400	1900	2100	K_Bacteria
OTU6	25	10	30	30	20	15	K_Bacteria

`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 (for later examples, assume that individuals 1–6 are from one population and individuals 7–10 are from a second population; thus, individuals are grouped by population). In this example, OTU2 is actually present on samples in group 1 (but not group 2), and OTU4 is actually present in group 2 (but not group 1). OTUs 3 and 6 are entirely contamination. This data frame loads as part of the *microDecon* package and can be accessed by entering `Example_2`.

OTU_ID	Blank	Ind1	Ind2	Ind3	Ind4	Ind5	Ind6	Ind7	Ind8	Ind9	Ind10
OTU1	0	50	80	50	70	85	50	50	40	50	40
OTU2	200	550	650	600	350	600	600	200	150	140	140
OTU3	1000	1200	1250	1200	850	1200	1200	850	900	1000	450
OTU4	50	60	60	60	50	70	45	250	200	200	250
OTU5	0	2000	2400	2650	2000	2900	2350	2500	1500	1000	600
OTU6	20	32	25	18	22	20	18	20	19	20	15

Important formatting points:

- Data should be run through standard quality control and filtering steps prior to using *microDecon* but **do not** normalize the reads before using the pacakge (e.g., via proportions [a.k.a. TSS], rarefying, CSS, DESeq, etc.)
- First column is OTU IDs (character or numeric)
- Blanks are in consecutive columns starting in column 2
    - `remove.cont()` and `decon()` can include multiple blanks, `remove.thresh()` cannot.
- Columns of individual data start after the last blank column
- Samples should be grouped into consecutive columns based on populations, collection sites, or some other sensible, *a priori* grouping criteria.
- The final column is an optional taxa column that has no specific formatting requirements other than that it consists of only one column.

2.4.0 Running the functions

2.4.1 `decon()`

This is a wrapper function for all other *microDecon* functions (i.e., it allows you to run the entire package in a single, easy step). It first decontaminates the data using `remove.cont()`, then it runs `remove.thresh()`, then it runs `decon.diff()`. If it is set to do multiple runs, they are completed before moving to `remove.thresh()`. To use `remove.cont()` and `decon.diff()` only, either run them independently, or set thresh = 1 and `prop.thresh = 0`. This will shut off the `remove.thresh()` arguments (you still must set `numb.ind`).

The input is the same as `remove.cont()` and the output is the same `decon.diff()`. Individuals must be sorted into groups (populations, species, etc.). All arguments and defaults are the same as `remove.cont()`, `remove.thresh()`, and `decon.diff()`.

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. Individuals must be ordered by group (e.g., species, populations, etc.)
`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).
`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.
`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 User's Guide section 1.4.3).

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 samples become 0 for that OTU). The threshold always rounds down to calculate the maximum number of zeros that can be present (e.g., 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 do not 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, the OTU will be set to zero for all individuals in that group (e.g., if a particular OTU makes up 0.001 percent 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 percent]). The proportions are based on all reads for all individuals in a group (including OTUs that were not in the blank). It is necessary to relax this threshold (e.g., 0.0005) for very small data sets (see User's Guide section 1.4.4) Set to 0 if you do not want to use this threshold.
regression	Numeric (default = 0). Specifies the regression equation used to calculate the constant. 0 = it chooses between regression 1 and regression 2 based on the low.threshold and up.threshold arguments (this is strongly recommended). 1 = it always uses regression 1. 2 = it always uses regression 2. See User's Guide section 1.4.2.
low.threshold	Numeric (default = 40). Selects the lower point for switching between regression 1 and regression 2. It uses regression 2 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 regression 1 and regression 2. It uses regression 2 anytime that the estimated overlap is <low.threshold or >up.threshold. It is usually best not to change this value.

Output:

`decon()` returns a list of five data frames that can be accessed with $. These are useful for both seeing and recording the changes *microDecon* made, as well as checking that the changes make sense based on the biological understanding of the system under study. "NA" indicates that an OTU had no reads for a given sample in the original OTU table.

`$decon.table`	A data frame of decontaminated OTU data. It is structured the same as the original data frame ("data"). However, if several blanks were input, the output will include only a single "Mean.blank" column that is the mean of the proportions of those blanks multiplied by the mean number of reads in the blanks. Additionally, the order of the rows may be different, and any OTUs for which all reads were removed will have been deleted (their information will still be shown in the other outputs).
`$reads.removed`	An OTU table showing the number of reads that were removed from each OTU that amplified in the blank (per individual).
`$difference.sum`	The total number of reads that were removed from each OTU that amplified in the blank (per group as well as for the entire data set; groups are in the same order as specified by the numb.ind argument).
`$difference.mean`	The average number of reads that were removed from each OTU that amplified in the blank (per group as well as for the entire data set; groups are in the same order as specified by the numb.ind argument).
`$OTUs.removed`	A data frame showing the identities of OTUs that were completely removed from either particular groups or the entire data set.

Example 1: Using the `Example_1` data frame from section 2.3.0 on default settings with all individuals as a single population

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

To access the table of decontaminated results use the code below (note that some OTUs have been entirely removed)

```
> result1$decon.table

 OTU_ID  Mean.blank  Ind1  Ind2  Ind3              taxa
  OTU1        0        60    64    40   K_Bacteria; P_Actinobacteria
  OTU2      208.1      359   318   329   K_Bacteria; P_Proteobacteria
  OTU5        0       2400  1900  2100           K_Bacteria
```

To access data frame of OTUs that were completely removed from at least one group (which in this case is the same as the entire data set) use the following code

```
> result1$OTUs.removed
```

```
 OTU_ID Mean.blank    All.groups        Group1             taxa
                                                    K_Bacteria;
  OTU3     1027.4   Totally.removed Totally.removed  P_Proteobacteria
                                                      K_Bacteria;
  OTU4      48.6    Totally.removed Totally.removed   P_Bacteroidetes
  OTU6      20.9    Totally.removed Totally.removed     K_Bacteria
```

Example 2: Using the `Example_1` data frame from section 2.3.0 with all individuals as a single population, but only doing one run and always using regression 1 (this is not as effective, and some contaminant reads are retained because it is usually best to do two runs as opposed to one).

```
> result2 <- decon(data = Example_1, numb.blanks = 3, numb.ind =
3, taxa = T, runs = 1, regression = 1)
```

Example 3: Using the `Example_2` data frame from section 2.3.0 on default settings, with individuals 1–6 as a group and individuals 7–10 as a group (this still retains two contaminant reads for OTU6 because the default setting for `prop.thresh` is too stringent for such a small data set; see section 2.4.3 for details and how to correct this).

```
> result3 <- decon(data = Example_2, numb.blanks = 1, numb.ind =
c(6,4), taxa = F)
> result3$decon.table
```

Example 4: Using the `Example_2` data frame from section 2.3.0 with individuals 1–6 as a group and individuals 7–10 as a group, but with `remove.thresh()` arguments (`prop.thresh` and `thresh`) turned off. Without those arguments, contaminant reads are retained for OTU4, and OTU6 (see section 2.4.3 for details).

```
> result4 <- decon(data = Example_2, numb.blanks = 1, numb.ind =
10, taxa = F, thresh = 1, prop.thresh = 0)
> result4$decon.table
```

2.4.2 remove.cont()

This is the primary function for removing contamination. It outputs a single OTU table of decontaminated results (OTUs that were entirely contamination are still included as rows of 0s).

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. Individuals must be ordered by group (e.g., species, populations, etc.).
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 User's Guide section 1.4.3).
regression	Numeric (default = 0). Specifies the regression equation used to calculate the constant. 0 = it chooses between regression 1 and regression 2 based on the low.threshold and up.threshold arguments (this is strongly recommended). 1 = it always uses regression 1. 2 = it always uses regression 2. See User's Guide section 1.4.2.
low.threshold	Numeric (default = 40). Selects the lower point for switching between regression 1 and regression 2. It uses regression 2 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 regression 1 and regression 2. It uses regression 2 anytime that the estimated overlap is <low.threshold or >up.threshold. It is usually best not to change this value.

Example 5: Using the `Example_1` data frame from section 2.3.0 on default settings.

```
> result5 <- remove.cont(Example_1, numb.blanks = 3, taxa = T)
```

Example 6: Using the `Example_2` data frame from section 2.3.0 on default settings.

```
> result6 <- remove.cont(Example_2, numb.blanks = 1, taxa = F)
```

2.4.3 `remove.thresh()`

This function removes residual contamination in the output from `remove.cont()`. As always, only OTUs that amplified in the blank(s) are affected. It outputs an OTU table of decontaminated results (OTUs that were entirely contamination are still included as 0s).

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. Individuals must be ordered by group (e.g., species, populations, etc.)
`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 or different species could be treated as different groups). Data must be sorted by these groups beforehand.
`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 samples become 0 for that OTU). The threshold always rounds down to calculate the maximum number of zeros that can be present (e.g., 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 do not 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, the OTU will be set to zero for all individuals in that group (e.g., if a particular OTU makes up 0.001 percent 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 percent]). The proportions are based on all reads for all individuals in a group (including OTUs that were not in the blank). It is necessary to relax this threshold (e.g., 0.0005) for very small data sets (see User's Guide section 1.4.4) 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 group (e.g., population) and individuals 7–10 are a group. Assume that these data were run through `remove.cont()` on default settings as shown below (note that some contaminant reads are retained [highlighted] which is why `remove.thresh()` is needed).

```
> result6 <- remove.cont(Example_2, numb.blanks = 1, taxa = F)
> result6
```

OTU_ID	Blank	Ind1	Ind2	Ind3	Ind4	Ind5	Ind6	Ind7	Ind8	Ind9	Ind10
OTU1	0	50	80	50	70	85	50	50	40	50	40
OTU2	200	230	400	360	148	360	360	0	0	0	0
OTU3	1000	0	0	0	0	0	0	0	0	0	0
OTU4	50	0	0	0	0	10	0	200	154	150	216
OTU5	0	2000	2400	2650	2000	2900	2350	2500	1500	1000	600
OTU6	20	0	0	0	2	0	0	0	1	0	1

Example 7: Default settings (other than taxa), with each group treated separately. This removes the remaining contaminant reads for group 1 (individuals 4 and 5), but they are still retained for group 2 because the `thresh` argument does not apply when there are four or fewer individuals in a group, and the default setting for `prop.thresh` is too liberal for such a small data set.

```
> result6.thresh.1.1 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(6,4))
> result6.thresh.1.1[order(result6.thresh.1.1$OTU_ID),] #returns
data frame ordered by OTU_ID
```

OTU_ID	Blank	Ind1	Ind2	Ind3	Ind4	Ind5	Ind6	Ind7	Ind8	Ind9	Ind10
OTU1	0	50	80	50	70	85	50	50	40	50	40
OTU2	200	230	400	360	148	360	360	0	0	0	0
OTU3	1000	0	0	0	0	0	0	0	0	0	0
OTU4	50	0	0	0	0	0	0	200	154	150	216
OTU5	0	2000	2400	2650	2000	2900	2350	2500	1500	1000	600
OTU6	20	0	0	0	0	0	0	0	1	0	1

Because there are only 6,502 reads retained in group 2, the default `prop.thresh` setting (0.00005) would only remove an OTU if it had less than one read. Therefore, a more liberal

setting of 0.0005, which will remove any OTUs with three or fewer reads, should be used. The decision of what threshold to use should usually be made *a priori* based on the number of samples per group (for `thresh`) and number of reads per group (for `prop.thresh`). Additionally, in our simulations, `prop.thresh` = 0.00005 was only a slight improvement over `prop.thresh` = 0.0005 (see Figure 7 in section 1.4.4); therefore, `prop.thresh` should be set based on the smallest group, and that setting should be applied to all samples.

```
> result6.thresh.1.2 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(6,4),prop.thresh=0.0005)
> result6.thresh.1.2[order(result6.thresh.1.2$OTU_ID),] #returns
data frame ordered by OTU_ID
```

OTU_ID	Blank	Ind1	Ind2	Ind3	Ind4	Ind5	Ind6	Ind7	Ind8	Ind9	Ind10
OTU1	0	50	80	50	70	85	50	50	40	50	40
OTU2	200	230	400	360	148	360	360	0	0	0	0
OTU3	1000	0	0	0	0	0	0	0	0	0	0
OTU4	50	0	0	0	0	0	0	200	154	150	216
OTU5	0	2000	2400	2650	2000	2900	2350	2500	1500	1000	600
OTU6	20	0	0	0	0	0	0	0	0	0	0

Example 8: Using `thresh` argument only, with each group treated separately (this retains contaminant reads for group 2). This is not recommended.

```
> result6.thresh.2 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(6,4), thresh = 0.7, prop.thresh = 0)
```

Example 9: Use `prop.thresh` argument only, with each group treated separately (this retains the contaminant reads for Ind5/OTU4). This is not recommended.

```
> result6.thresh.3 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(6,4), thresh = 1, prop.thresh = 0.0005)
```

Example 10: All individuals treated as a single group (this retains the contaminant reads for Ind5/OTU4). This should only be used if all individuals truly are from the same group.

```
> result6.thresh.4 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(10))
```

Example 11: Each individual treated separately (in this example, this retains all of the contaminant reads). This is not recommended.

```
> result6.thresh.5 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(rep(1,10)))
```

2.4.5 `decon.diff()`

This function takes the output of either `remove.cont()` or `remove.thresh()` as well your original, contaminated data, and it returns summary statistics of how many OTUs were removed. Additionally, if any OTUs were completely removed from all samples, it removes that entire row and returns a data frame without that row (otherwise, that row will be a row of entirely zeros). This is useful for documenting the changes made by *micoDecon* as well as ensuring that the results are reasonable based on biological knowledge of the system being studied.

Arguments:

`data`	The original data frame that you input into remove.cont(). Individuals must have been ordered by groups (populations, species, etc.) as in remove.thresh().
`output`	The data frame that was returned by remove.cont() or remove.thresh().
`numb.blanks`	Numeric (default = 1). Specifies the number of blanks included in the "data" argument (if multiple blanks are included, they must be in consecutive columns, starting with column 2). This only applies to the number in the "data" argument. The number in the "output" argument will always be 1 because remove.cont() returns a single mean blank.
`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 have been sorted by these groups before running remove.cont().
`taxa`	Logical (T/F). Specifies whether or not the last column contains taxonomic information (default = T).

Output:

`decon.diff()` returns a list of five data frames that can be accessed with $. "NA" indicates that an OTU had no reads for a given sample in the original OTU table.

`$decon.table`	A data frame of decontaminated OTU data. It is structured the same as the original data frame ("data"). However, if several blanks were input, the output will include only a single "Mean.blank" column that is the mean of the proportions of those blanks multiplied by the mean number of reads in the blanks. Additionally, the order of the rows may be different, and any OTUs for which all reads were removed will have been deleted (their information will still be shown in the other outputs).
`$reads.removed`	An OTU table showing the number of reads that were removed from each OTU that amplified in the blank (per individual).

$difference.sum	The total number of reads that were removed from each OTU that amplified in the blank (per group as well as for the entire data set; groups are in the same order as specified by the numb.ind argument).
$difference.mean	The average number of reads that were removed from each OTU that amplified in the blank (per group as well as for the entire data set; groups are in the same order as specified by the numb.ind argument).
$OTUs.removed	A data frame showing the identities of OTUs that were completely removed from either particular groups or the entire data set.

Example 12: Using the `Example_1` data frame from section 2.3.0 on default settings with all individuals as a single population. This example shows a `remove.cont()` run, followed by `decon.diff()`. The first line is the same as Example 5 in section 2.4.2.

```
> result5 <- remove.cont(data = Example_1, numb.blanks = 3, taxa =
T)

> final.result5 <- decon.diff(data = Example_1, output = result5,
numb.blanks = 3, numb.ind = 3, taxa = T)
```

Example 13: Using the `Example_2` data frame from section 2.3.0, where individuals 1–6 are a population and individuals 7–10 are a population. This example shows a `remove.cont()` run, followed by a `remove.thresh()` run, followed by `decon.diff()`. The first line is the same as Example 6 in section 2.4.2, and the second is the same as Example 7 in section 2.4.3.

```
> result6 <- remove.cont(data = Example_2, numb.blanks = 1, taxa =
F)

> result6.thresh.1.2 <- remove.thresh(data = result6, taxa = F,
numb.ind = c(6,4),prop.thresh=0.0005)

> final.result6 <- decon.diff(data = Example_2, output =
result6.thresh.1.2, numb.blanks = 1, numb.ind = c(6,4), taxa = F)
```

Collectively, this is the same as simply running `decon()`; however, running each function separately can be useful if you want more information on what happened at each stage. The following lines of code would produce the same final result, but would show the full `decon.diff()` output at each stage (keep in mind that after one `remove.cont()` run, the number of blanks will always be 1).

```
> #run 1
> result6.run1 <- remove.cont(data = Example_2, numb.blanks = 1,
taxa = F, runs= 1)

> #effects of run 1
> decon.diff(data = Example_2, output = result6.run1, numb.blanks
= 1, numb.ind = c(6,4), taxa = F)

> #run 2
> result6.run2 <- remove.cont(data = result6.run1, numb.blanks =
1, taxa = F, runs= 1)

> #total effects of effects of runs 1 and 2
> decon.diff(data = Example_2, output = result6.run2, numb.blanks
= 1, numb.ind = c(6,4), taxa = F)

> #effects of run 2 (i.e., changes run 2 made to the results of
run 1)
> decon.diff(data = result6.run1, output = result6.run2,
numb.blanks = 1, numb.ind = c(6,4), taxa = F)

> #use remove.thresh
> result6.thresh <- remove.thresh(data = result6.run2, taxa = F,
numb.ind = c(6,4),prop.thresh=0.0005)

> #final result and total effects of effects of run 1, 2, and
remove.thresh
> final.result6 <- decon.diff(data = Example_2, output =
result6.thresh, numb.blanks = 1, numb.ind = c(6,4), taxa = F)

> #effects of remove.thresh on results of run 2
> decon.diff(data = result6.run2, output = result6.thresh,
numb.blanks = 1, numb.ind = c(6,4), taxa = F)
```

The final result is the same as before, but doing it this way allows you to get the more details about what the package did at each step.

2.5.0 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 homogeneous (e.g., are similar

in stacked bar plots and ordination plots), then you can put all of the data together and use *microDecon* over the entire data set (including all blanks). If the blanks from different batches consistently differ, however, then use *microDecon* separately on each batch. If there is substantial heterogeneity among blanks within batches, this suggests contamination from poor laboratory or sampling techniques, which is not consistent across samples. microDecon will not work well in that situation.

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.* Include all of your OTUs, not simply the ones that amplified in the blank (OTUs that did not amplify in the blank are not affected, but regression 2 does use them as part of its calculations).
3. Group individuals into sensible *a priori* clusters based on species, populations, environments, etc. Groups should represent the smallest level at which you expect there to be consistent differences. The idea is that some OTUs might be present but rare in one group and absent in another group, Therefore, *microDecon* treats each group separately for `remove.thresh()` arguments and returns summary statistics for each group. Failing to correctly assign groups can result in OTUs being incorrectly removed from the data (as always, this function only affects OTUs that amplified in the blank).
4. 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.
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. Generally, the thresh and `prop.thresh` arguments should be used on their default values, but they may need to be adjusted (particularly relaxing `prop.thresh` for small data sets).
7. Generally, `regression = 0` (default) is best based on our simulations. However, `regression = 1` uses a more simplistic and consistent algorithm which generally performs well and is a suitable alternative. `regression = 2` only performs well in a small subset of cases and should not be selected manually.