Orthology bonus extra exercise

Bonus exercise using OrthoDB orthology data to build an orthology landscape

*Required

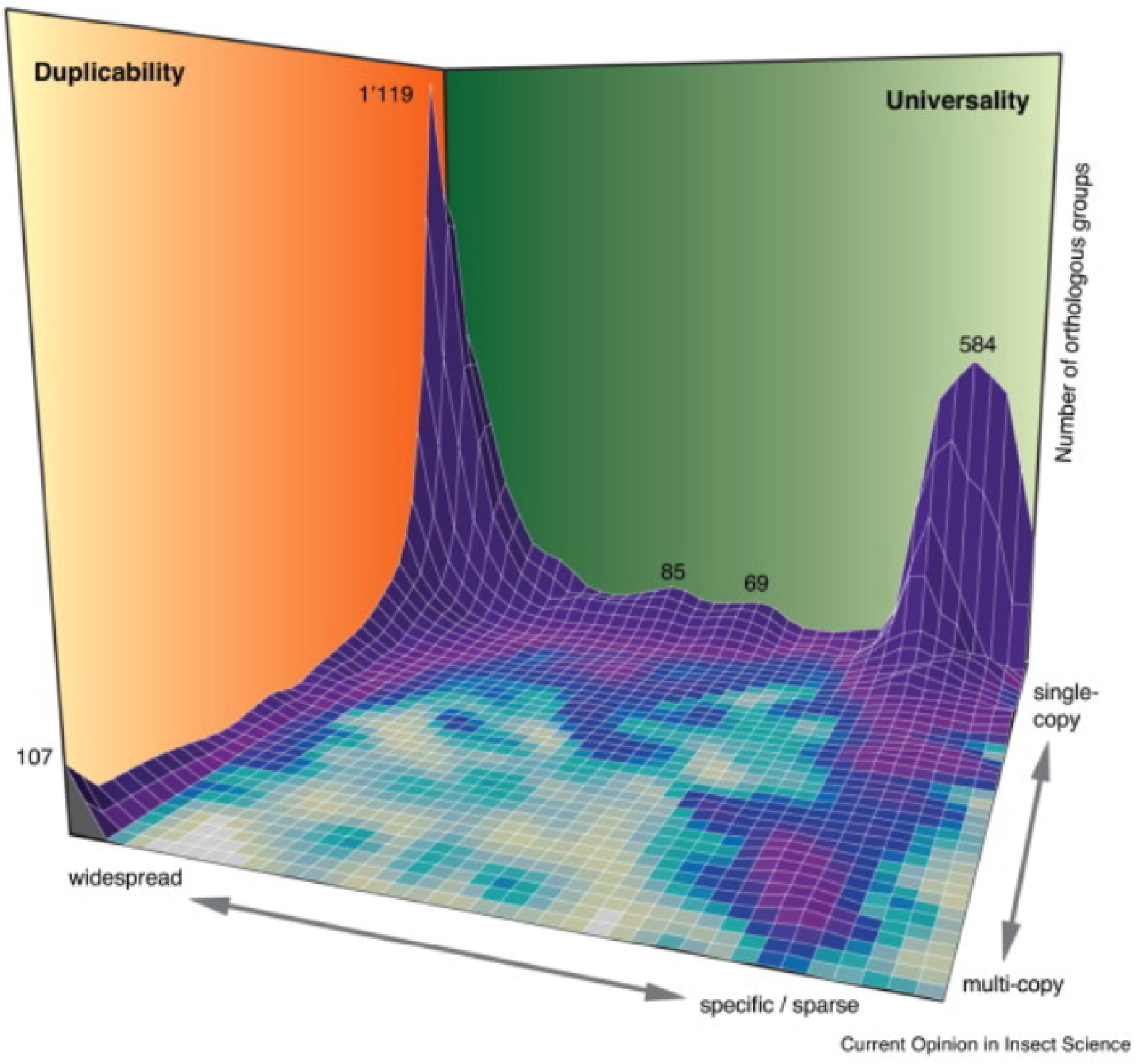
Your mission:

1. Collect counts of genes per species for all hymenopteran orthologous groups.
2. Compute universality (% of species) and duplicability (% of multi-copy species).
3. Plot orthology landscape using R.

Scripts are provided, as well as pre-computed intermediate and final results files, but the main idea is for you to use the hints and code snippets provided to write your own scripts and develop them while progressing through the exercise.

From the Moodle site, find the folder under 'Day 2 Rob Waterhouse' called 'Orthology_bonus', inside you should see several scripts and datafiles, so if you get stuck then you can try using these instead of writing your own scripts.

D. melanogaster orthology landscape, Waterhouse, COIS 2015



1. **To proceed you must accept this mission ...** *

   Mark only one oval.

   ◯ I accept the mission!

1. Collecting gene & species counts

Some hints:

* View OrthoDB API help page to find out about query structures:
http://www.orthodb.org/?page=api

* We aim to collect counts of genes per species for all hymenopteran orthologous groups

* NCBI taxonomy ID for the Hymenoptera level (node) is 7399

* To get a list of all Hymenoptera groups the API URL query would be:
http://www.orthodb.org/search?level=7399

* Note that the default limit for OrthoDB API queries is 1000, so if more than 1000 groups are returned you will have to increase the limit by adding, e.g. &limit=25000 to the API URL

* To view the same query using the OrthoDB website instead of the API, try:
http://www.orthodb.org/?level=7399

2. **How many Hymenoptera groups are returned?** *

   ____

2. Scripting collection of orthologous groups

Some hints:

* For Perl scripts: #!/usr/bin/perl
* For Python scripts: #!/usr/bin/python

* There are handy Perl and Python modules to connect to APIs and to interpret the returned JSON (JavaScript Object Notation) results:

```
use REST::Client;   # Perl for database connection and querying
use JSON;           # Perl for JSON to Perl interoperability
import requests     # Python for database connection and querying
import json         # Python for JSON to Python interoperability
```

==>>>>
* Example in Perl:

```
# Load required modules:
use lib '/path/to/your/MyPmods';   # location of your locally installed REST module
use REST::Client;
use JSON;
# Set connection to the OrthoDB website:
my $host = 'http://www.orthodb.org/';
my $client = REST::Client->new(host => $host);
# define the query to get the JSON result
$client->GET("search?limit=25000&level=7399");
# Convert JSON to PERL data structure
my $result = decode_json( $client->responseContent() );
# Get all orthologous groups returned from the query
my @groups = @{ $result->{'data'} };
# Print the total number of groups returned
print scalar(@groups);
```

==>>>>
* Example in Python

```
# Load required modules:
import requests
import json
# define the query to get the JSON result
url = "http://www.orthodb.org/search?limit=25000&level=7399"
myResponse = requests.get(url)
# Convert JSON to PYTHON data structure
jData = json.loads(myResponse.content)
# NB: Did not work in python3, have to decode from bytes to strings with:
jData = json.loads(myResponse.content.decode('utf-8'))
# Get all orthologous groups returned from the query
groups = jData['data']
# Print the total number of groups returned
print(len(groups))
```

3. **Were you successful?** *

Mark only one oval.

- ⬭ Yes
- ⬭ No, but I want to continue regardless

3. Scripting collection of gene & species counts

Some hints:

* The previous step returned a list of all orthologous groups, so now we can loop through this list to fetch the genes from each species in these groups.

* Perhaps start with a low query limit (e.g. 100) just to get it working.

==>>>>

* Perl example of looping through the groups returned, and printing the results to file:

```
open(OUT, ">My_species_genes_counts.txt") || die $!;
foreach my $group (@groups) {
   $client->GET("orthologs?id=$group");
   # Convert JSON to PERL data structure
   my $result = decode_json( $client->responseContent() );
   # Loop through each species in the group
   foreach my $species ( @{ $result->{'data'} } ) {
      my $spec=$species->{'organism'}{'id'};          # get species NCBI taxonomy ID
      my $orgn=$species->{'organism'}{'name'};     # get species name
      my @genes=@{$species->{'genes'}};              # get all genes for this species
      print OUT "$group\t$spec\t$orgn\t" . scalar(@genes) . "\n";
   }
}
close(OUT);
```

==>>>>

* Python example of looping through the groups returned, and printing the results to file:

```
groupToSpeciesToCountDict = defaultdict(dict) ## Dictionary where we will store all genes belonging
to an orthology group belonging to a species. Note its a dictionary in a dictionary
for group in orthoGroups: ## Iterate over all orthology groups we retrieved from the JSON data
        query = 'orthologs?id='+group ## query to retrieve all genes belong to an ortholog
        response = requests.get("http://www.orthodb.org/"+query)
        jsonData = json.loads(response.content.decode('utf-8')) ## Convert the response we got from
the API to python JSON format
        for value in jsonData['data']: ## Iterate over all JSON data and assign all required parts to
variables
                speciesId = value['organism']['id']
                speciesName = value['organism']['name']
                speciesIdToNameDict[speciesId] = speciesName ## Assign a species numerical identifier to
a human readable species name
                speciesGenes = value['genes']
                groupToSpeciesToCountDict[group][speciesId] = len(speciesGenes) ## Assign the amount
of genes belonging to an orthologous group, for each species
```

4. **Were you successful?** *

Mark only one oval.

- ◯ Yes!
- ◯ No, but I want to continue regardless

4. Collection of all gene & species counts

Some hints:

* Now you can count genes per species per group you can loop through all groups and save all the counts per species and print a complete data matrix that will be easy to extract what you need from for plotting using R later.

```
=============================================>>>>
* Perl example of counting number of genes per species in each group:
# First loop through each returned group to count and save the number of genes per species
undef my %all_species;                    # hash to save full species list
undef my %group2species2count;    # hash to save gene count per species per group
foreach my $group (@groups) {
   # Define the query to return all orthologs from the group
   $client->GET("orthologs?id=$group");
   # Convert JSON to PERL data structure
   my $result = decode_json( $client->responseContent() );
   # Loop through each species in the group
   foreach my $species ( @{ $result->{'data'} } ) {
      my $spec=$species->{'organism'}{'id'};              # get species NCBI taxonomy ID
      my $orgn=$species->{'organism'}{'name'};          # get species name
      $all_species{$spec}=$orgn;                              # save species ID to name hash
      my @genes=@{$species->{'genes'}};                 # get all genes for this species
      $group2species2count{$group}{$spec}=scalar(@genes);     # save gene count
   }
}
# Now open output file ($level defined earlier, Hymenoptera node)
open(OUT,">ODB_SPECIES_GENES_COUNTS_from\_$level\.txt") || die $!;
print OUT "OrthoGroup\tSpecies\tGenes\tSCspecies";          # column names for output file
# Open species list output file (so you have a record of all species names)
open(LST,">ODB_SPECIES_GENES_COUNTS_from\_$level\_species-list.txt") || die $!;
foreach my $spec (sort { $all_species{$a} cmp $all_species{$b} } keys %all_species) {
   print LST "$all_species{$spec}\t$spec\n";
   print OUT "\t$spec";       # column names for output file
}
close(LST);
print OUT "\n";                    # add return to end of header line for output file
foreach my $group (sort @groups) {
   my $allcounts='';     # string to collect all gene counts (in sort order)
   my $specount=0;    # species counter
   my $gencount=0;    # gene counter
   my $scpcount=0;    # single-copy species counter
   foreach my $spec (sort { $all_species{$a} cmp $all_species{$b} } keys %all_species) {
      if(defined($group2species2count{$group}{$spec})) {
         $allcounts.="\t$group2species2count{$group}{$spec}";   # add count to string
         $specount++;                                                                    # increment species counter
         $gencount+=$group2species2count{$group}{$spec};       # sum gene counter
         # if single-copy, increment single-copy species counter
         if($group2species2count{$group}{$spec}==1) { $scpcount++; }
      }
      else { $allcounts.="\t0"; }                                                   # if not defined then count=0
   }
   # print GroupID, number of species, number of genes, number of single-copy species
   print OUT "$group\t$specount\t$gencount\t$scpcount";
   # print ordered string of counts per species (including zeros)
   print OUT $allcounts . "\n";
}
close(OUT);
```

======================================>>>>
* Python example of counting number of genes per species in each group:

```
speciesIdToNameDict = {}
groupCount = 0
groupToSpeciesToCountDict = defaultdict(dict) ## Dictionary where we will store all genes belonging
to an orthology group belonging to a species. Note its a dictionary in a dictionary
for group in orthoGroups: ## Iterate over all orthology groups we retrieved from the JSON data
    query = 'orthologs?id='+group ## query to retrieve all genes belong to an ortholog
    response = requests.get(host+query)
    if response.status_code != 200: ## Response not 200 means its an erroneous query
        print('Bad query: '+host+query+'\n')
        print('Response code: '+str(response.status_code)+'\n')
        print('Response content: '+response.text+'\n')
        sys.exit() ## Quits the script, because we have a bad query
    jsonData = json.loads(response.content.decode('utf-8')) ## Convert the response we got from
the API to python JSON format
    for value in jsonData['data']: ## Iterate over all JSON data and assign all required parts to
variables
        speciesId = value['organism']['id']
        speciesName = value['organism']['name']
        speciesIdToNameDict[speciesId] = speciesName ## Assign a species numerical identifier to
a human readable species name
        speciesGenes = value['genes']
        groupToSpeciesToCountDict[group][speciesId] = len(speciesGenes) ## Assign the amount
of genes belonging to an orthologous group, for each species
    groupCount += 1
    if groupCount % 10 == 0:
        print(str(groupCount)+' orthologous groups processed ... '+str(datetime.now()))

## Create a file where we will store all the processed data
outfile = open('ODB_SPECIES_GENES_COUNTS_from_'+str(level)+'.txt','w')
outfile.write('Orthogroup\tSpecies\tGenes\tSCspecies') ## Start of the header of the file

## Create a file where we will store the IDs belonging to a species code
outfile2 = open('ODB_SPECIES_GENES_COUNTS_from_'+str(level)+'_species-list.txt','w')

## Write a tabular file with species ID - species name pairs
speciesIdToNameDict = sorted(speciesIdToNameDict.items(),key=operator.itemgetter(1)) ## Sorts the
dictionary by its values (species names)
for species in speciesIdToNameDict:
    outfile2.write(species[1]+'\t'+species[0]+'\n')
    outfile.write('\t'+species[0]) ## Additionally, add each species name to the header of our main
output file
outfile2.close()

outfile.write('\n')

## Iterate over all orthology groups, and for every orthology group and every species the amount of
orthologs belonging to the orthology group and the species
for group in sorted(groupToSpeciesToCountDict):
    allCounts = ''
    specCount = 0
    geneCount = 0
    scpCount = 0
    for spec in speciesIdToNameDict: ## Iterate over the species
        specId = spec[0] ## Species identifier
        if specId in groupToSpeciesToCountDict[group]: ## If the species has orthologs belonging to
this orthology group, continue
            allCounts += '\t'+str(groupToSpeciesToCountDict[group][specId]) ## Write the amount
of species specific orthologous genes belong to the group to the output file
            specCount += 1 ## Increase the amount of species having a gene belonging to this
orthology group
            geneCount += groupToSpeciesToCountDict[group][specId] ## Increase amount of
genes belonging to the orthology group by the amount of genes belong to this species orthology
group
            if groupToSpeciesToCountDict[group][specId] == 1: ## If this orthologous gene is
single copy, increase single copy counter by 1
```

```
                    scpCount += 1
            else: ## If species doesn't have an orthologue, write '0' to the output file
                allCounts += '\t0'
    outfile.write(group+'\t'+str(specCount)+'\t'+str(geneCount)+'\t'+str(scpCount)+allCounts+'\n') ##
Process all numbers to the output file
outfile.close()
```

* This type of query can take some time, so if you have managed to get it working but wish to proceed you can use the counts file prepared earlier:
* From the Moodle site, find the folder under 'Day 2 Rob Waterhouse' called 'Orthology_bonus', inside you should see the text file called 'ODB_SPECIES_GENES_COUNTS_from_7399.txt'
* Right click to get the full URL of the file (Copy Link Location) and then wget it to your VM

```
$ wget
https://edu.sib.swiss/pluginfile.php/6286/mod_folder/content/0/ODB_SPECIES_GENES_COUNTS_from_7399.txt
```

* NB: if the URL you copied ends with '?forcedownload=1' then delete this part

5. **Were you successful?** *

Mark only one oval.

- ⬭ Yes
- ⬭ No, but I want to continue regardless

5. Prepare grid counts for plotting

Now we can choose a species for which we want to plot an orthology landscape, and then prepare the counts of each type of orthologous group - from universal to specific and all single-copy to all multi-copy.

Some hints:

* Look at the species-list file you made earlier to choose your species (i.e. to know which NCBI taxonomy ID to select)
* Or from the Moodle folder

```
$ wget
https://edu.sib.swiss/pluginfile.php/6286/mod_folder/content/0/ODB_SPECIES_GENES_COUNTS_from_7399_species-list.txt
```

* Essentially, we need to examine each orthologous group and see where it falls on the grid (see image below) in terms of its universality (% of total species) and duplicability (% single/multi-copy), and count the numbers of groups that fall into each section of the grid.

==>>>>
* Perl example of preparing the grid counts:

```
my $spec=7460;   # Apis mellifera selected here, you could choose another species
# Open full counts file from previous step
open(IN,"ODB_SPECIES_GENES_COUNTS_from_7399.txt") || die $!;
my @lines=<IN>;
close(IN);
my $header=shift(@lines);                    # save the header line, contains order of species
chomp($header);
my @columns=split(/\s+/,$header);      # split header line into array
my $selected=0;
for (my $i=4; $i<=$#columns; $i++) {
   if($columns[$i]==$spec) { $selected=$i; }   # to find which column is your selected species
}

my $maxspecies=0;                 # to find out the maximum number of species possible
undef my %group2species;      # hash to save species counts per group
undef my %group2singles;       # hash to save single-copy species counts per group
# Now loop through each line (group) and get species and single-copy species counts
foreach my $line (@lines) {
   chomp($line);
   my @data=split(/\s+/,$line);
```

```
    if($data[1]>$maxspecies) { $maxspecies=$data[1]; }  # to save the highest species count
    if($data[$selected]==0) { next; }                    # ignore group if selected species=0
    $group2species{$data[0]}=$data[1];            # save species counts per group
    $group2singles{$data[0]}=$data[3];            # save single-copy species counts per group
}

undef my %grid_counts;        # hash to save group counts per grid category
# Make a 20x20 grid and count number of groups per category:
foreach my $group (sort keys %group2species) {
    for(my $i=0;$i<=1;$i=$i+0.05) {                    # universality categories from 0 to 1 by 0.05
        for(my $j=0;$j<=1;$j=$j+0.05) {                # duplicability categories from 0 to 1 by 0.05
            my $k=0;
            if($j==0) { $k=0-0.01; }              # if zero set just below zero because later we use greater
than '>' $k
            else { $k=$j; }
            $k=sprintf("%.2f",$k);
            $i=sprintf("%.2f",$i);
            $j=sprintf("%.2f",$j);
            my $iplus=sprintf("%.2f",$i+0.05);
            my $jplus=sprintf("%.2f",$j+0.05);
            my $id="$i:$j";                       # ID of the grid category, universality category : duplicability
category
            my $prop_species=sprintf("%.2f",$group2species{$group}/$maxspecies);              # calculate
proportion of species (universality)
            my $prop_singles=sprintf("%.2f",$group2singles{$group}/$group2species{$group});    #
calculate proportion of single-copy species (duplicability/single-copyness)
            # if universality is between A & B and duplicability is between X & Y then increment grid
category counter:
            if($prop_species>$i && $prop_species<=$iplus && $prop_singles>$k &&
$prop_singles<=$jplus) {
                if(defined($grid_counts{$id})) { $grid_counts{$id}++; }
                else { $grid_counts{$id}=1; }
                print "$group $id SP: $i < $prop_species <= $iplus\tSC: $k < $prop_singles <= $jplus\n";
            }
        }
    }
}


# Now print out counts per category across the grid, in two columns: grid ID and group count
# These data will be used as input for plotting using R
open(OUT,">grid_counts\_$spec\.txt") || die $!;
print OUT "BIN\tCNT\n";
for(my $i=0;$i<1;$i=$i+0.05) {
    for(my $j=0;$j<1;$j=$j+0.05) {
        $i=sprintf("%.2f",$i);
        $j=sprintf("%.2f",$j);
        my $id="$i:$j";
        if(defined($grid_counts{$id})) { print OUT "$id\t$grid_counts{$id}\n"; }
        else { print OUT "$id\t0\n"; }
    }
}
close(OUT);
```

==>>>>

* Python example of preparing the grid counts:

```
specId = '7460' ## Apis mellifera selected here, you could choose another species

print('START: '+str(datetime.now()))
print('Selected species: '+specId+'\n')

## Open full counts file from previous step
infile = open('ODB_SPECIES_GENES_COUNTS_from_7399.txt')
header = infile.readline().strip() ## save the header line, contains order of species
columns = header.split('\t') ## Convert header line into a list = columns
selected = 0
```

```
for i in range(0,len(columns)): ## find out which column belongs to your selected species
    if columns[i] == specId:
        selected = i

maxSpecies = 0
groupToSpecies = {} ## dictionary to save species counts per group
groupToSingles = {} ## dictionary to save single-copy species counts per group

## Now loop through each line (group) and get species and single-copy species counts
for line in infile.readlines():
    ssline = line.strip().split('\t')
    if int(ssline[1]) > maxSpecies: # to save the highest species count, i.e. total
        maxSpecies = int(ssline[1])
    if ssline[selected] != '0': # ignore group if selected species=0, else process counts and save in
respective dictionary
        groupToSpecies[ssline[0]] = int(ssline[1])
        groupToSingles[ssline[0]] = int(ssline[3])


gridCountDict = defaultdict(int) ## dictionary to save group counts per grid category

# Make a 20x20 grid and count number of groups per category:
for group in sorted(groupToSpecies.iterkeys()):
    for i in range(0,100,5): # universality categories from 0 to 1 by 0.05 (python does not allow steps
of 0.5, so everything is done x100 and later divided by 100)
        for j in range(0,100,5): # duplicability categories from 0 to 1 by 0.05
            iSmall = round(i/100.0,2) ## Divided by 100 to get 'real' steps we want
            jSmall = round(j/100.0,2)
            k = 0
            if jSmall == 0: # if zero set just below zero because later we use greater than '>' $k
                k = 0-0.01
            else:
                k = jSmall
            id = str(iSmall)+':'+str(jSmall) ## ID of the grid category, universality category :
duplicability category
            iPlus = round(iSmall+0.05,2)
            jPlus = round(jSmall+0.05,2)
            propSpecies = round(groupToSpecies[group]/float(maxSpecies),2) ## calculate
proportion of species (universality)
            propSingles = round(groupToSingles[group]/float(groupToSpecies[group]),2) ##
calculate proportion of single-copy species (duplicability/single-copyness)
            ## if universality is between A & B and duplicability is between X & Y then increment
grid category counter:
            if propSpecies > iSmall and propSpecies <= iPlus and propSingles > k and
propSingles <= jPlus:
                gridCountDict[id] += 1
                print group+' '+id+' SP: '+str(iSmall)+' < '+str(propSpecies)+' <= '+str(iPlus)+"\tSC:
"+str(k)+' < '+str(propSingles)+' <= '+str(jPlus)



## Now print out counts per category across the grid, in two columns: grid ID and group count
## These data will be used as input for plotting using R
outfile = open('grid_counts_'+specId+'.txt','w')
outfile.write('BIN\tCNT\n')
for i in range(0,100,5):
    for j in range(0,100,5):
        iSmall = round(i/100.0,2)
        jSmall = round(j/100.0,2)
        id = str(iSmall)+':'+str(jSmall)
        if id in gridCountDict:
            outfile.write(id+'\t'+str(gridCountDict[id])+'\n')
        else:
            outfile.write(id+'\t0\n')

outfile.close()
print('END: '+str(datetime.now()))
```

* If you did not manage to get it working but wish to proceed you can use the counts file prepared earlier which you can download from the Moodle site:
$ wget https://edu.sib.swiss/pluginfile.php/6286/mod_folder/content/0/grid_counts_7460.txt
* Or Perl/Python scripts already prepared
$ wget https://edu.sib.swiss/pluginfile.php/6286/mod_folder/content/0/bonus-prepare_grid_counts.py
$ wget https://edu.sib.swiss/pluginfile.php/6286/mod_folder/content/0/bonus-prepare_grid_counts.pl

Universality versus Duplicability

present in few/none <= **Universality** => present in most/all

mostly single-copy <= **Duplicability** => mostly multi-copy

	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	40-45	45-50	50-55	55-60	60-65	65-70	70-75	75-80	80-85	85-90	90-95	95-100
0-5																				
5-10																				
10-15																				
15-20																				
20-25																				
25-30																				
30-35																				
35-40																				
40-45																				
45-50																				
50-55																				
55-60																				
60-65																				
65-70																				
70-75																				
75-80																				
80-85																				
85-90																				
90-95																				
95-100																				

6. **Were you successful?** *
Mark only one oval.

◯ Yes
◯ No, but I want to continue regardless

6. Landscape plotting in R

Now we have the counts we needs we can proceed to build a landscape plot using R.

Some hints:

* Plotting will require the R library 'akima' - further info here: https://cran.r-project.org/web/packages/akima/index.html

* The main plotting call is made with 'persp' - further info here: https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/persp.html

* 'persp' input requires x & y lists which are locations of grid lines at which the values in the z dimension are measured, we will need 20 x 20 points (because that is what we calculated counts for), so from 0.05 to 1 by steps of 0.05, then x is each value repeated 20 times, and y is 20 times the series of values:

```
n<-seq(0.05,1,by=0.05)
x<-rep(n,each=20)
y<-rep(n,20)
```

* Read in the counts from the grid_counts file, these will make up out 'z' dimension values (once they have been interpolated)

```
cnt<-read.table(file="grid_counts_7460.txt",header=TRUE,row.names=1)
z<-cnt$CNT
```

* Use akima's 'interp' function to interpolate x-y-z values from raw input values

```
a<-interp(y,x,z)
```

* Make a colour ramp that scales with your group counts:

```
nrz <- nrow(a$z)
ncz <- ncol(a$z)
mycolors <- colorRampPalette( c("aquamarine", "darkblue", "yellow", "darkred") )
nbcol <- nrz*ncz
# Add white as the first colour (to clearly see zeros)
color <- c("white",mycolors(nbcol-1))
# Compute the z-value at the facet centres
zafacet <- a$z[-1, -1] + a$z[-1, -ncz] + a$z[-nrz, -1] + a$z[-nrz, -ncz]
# Recode facet z-values into colour indices
facetcola <- cut(zafacet, nbcol)
```

* Now plot your landscape:

```
pdf("my_landscape.pdf")
persp(a, shade=0.3, theta=-70, phi=10, col=color[facetcola], ticktype="detailed", nticks=10,
xlim=c(0.05,1), ylim=c(0.05,1), cex.axis=0.6, xlab="Single-copyness", ylab="Universality",
zlab="Group counts")
dev.off()
```

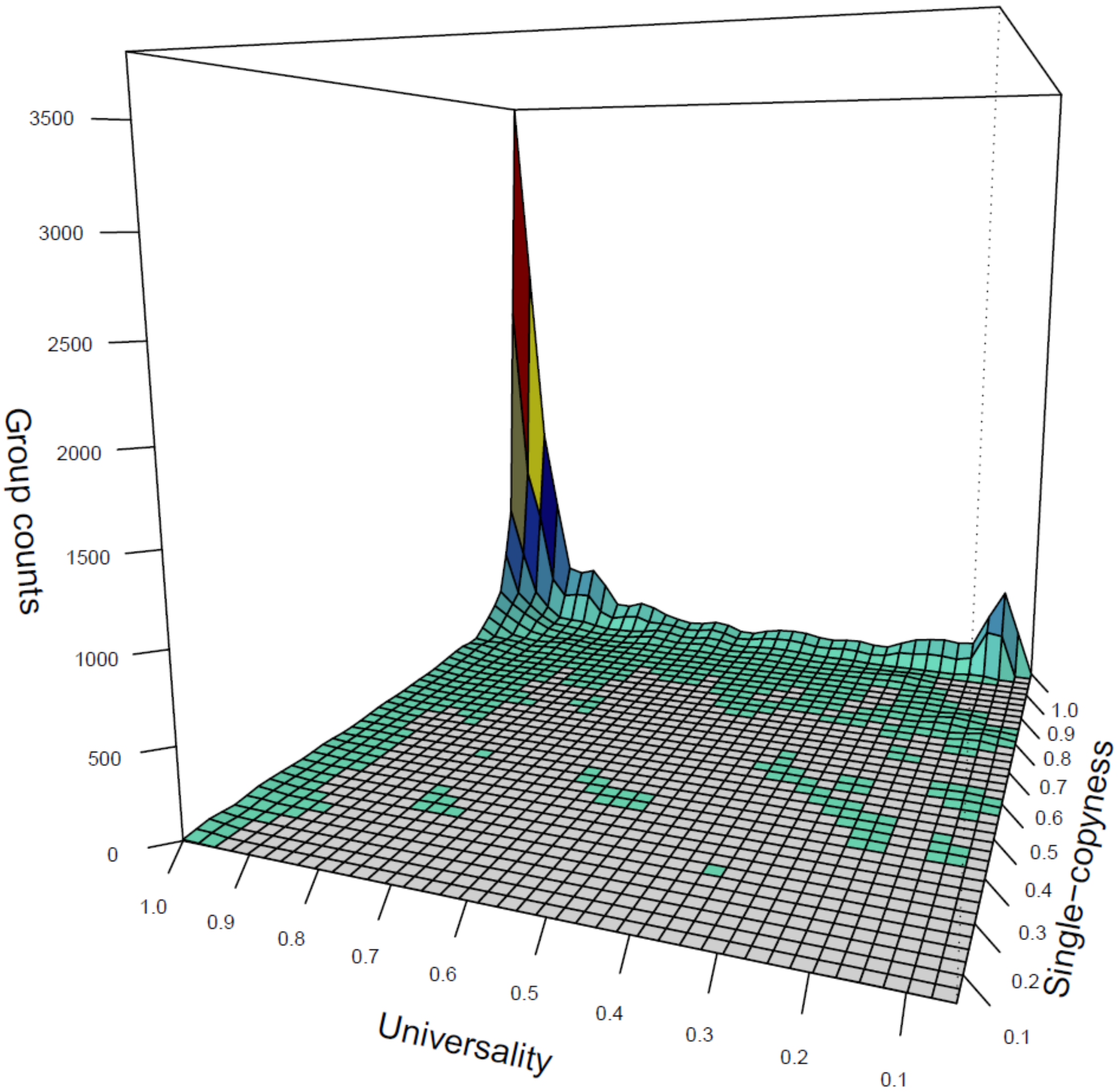
* Note that with 'persp', theta & phi are parameters used to control the angles defining the viewing direction: theta gives the azimuthal direction and phi the colatitude
* shade controls the shading of the surface, see https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/persp.html for various other options to control your plot

* NB: as there are many small counts and few large counts, you might want to scale your z-values so that your plot appears more mountainous, e.g. by taking the square root of the group counts instead of the raw counts:

```
z2 <- z^0.5
a<-interp(y,x,z2)
```

* Play around with scaling, colours, and viewing options to tailor your landscape plot as you wish!

Non-scaled Apis mellifera orthology landscape



7. **Were you able to plot an orthology landscape?** *

Mark only one oval.

◯ Yes, and tailor it! *Skip to "Congratulations!."*

◯ No, I have failed my mission and am very confused! *Skip to "We're here to help!."*

Congratulations!

Changes made to get the plot below:

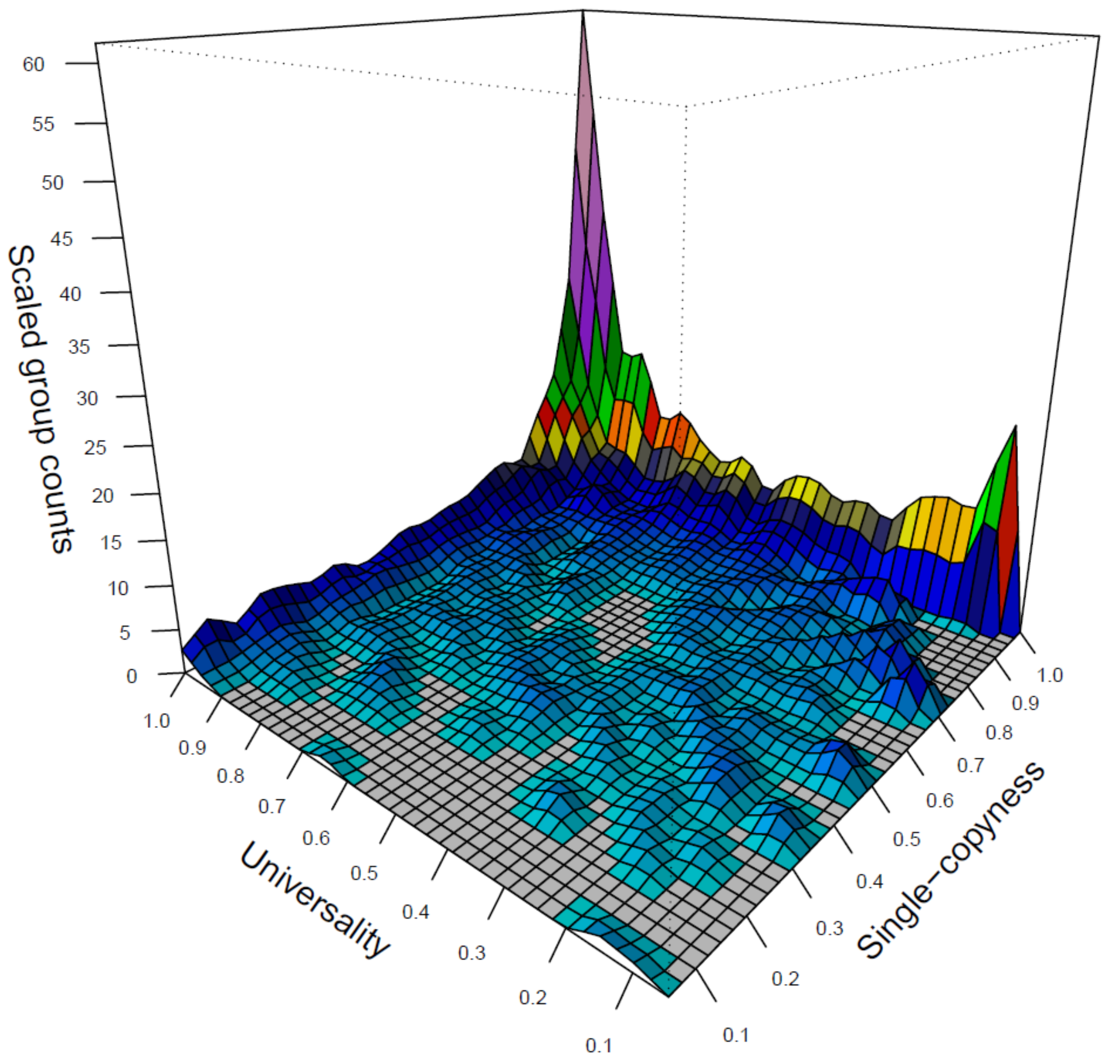
```
# SCALE
z2<-z/(z/(z^0.5))
z2[z2=="NaN"]<-0

a<-interp(y,x,z2)

nrz <- nrow(a$z)
ncz <- ncol(a$z)
mycolors1 <- colorRampPalette( c("cyan", "blue", "darkblue","yellow", "red") )
mycolors2 <- colorRampPalette( c("green", "darkgreen") )
mycolors3 <- colorRampPalette( c("purple", "pink") )
nbcol <- nrz*ncz
color <- c("white",mycolors1(nbcol-((nbcol/4)*3)),mycolors2(nbcol/4),mycolors3((nbcol/4)+500))
zafacet <- a$z[-1, -1] + a$z[-1, -ncz] + a$z[-nrz, -1] + a$z[-nrz, -ncz]
facetcola <- cut(zafacet, nbcol)

pdf("my_landscape2.pdf")
persp(a, shade=0.5, theta=-50, phi=20, col=color[facetcola], ticktype="detailed", nticks=10,
xlim=c(0.05,1), ylim=c(0.05,1), cex.axis=0.6, xlab="Single-copyness", ylab="Universality",
zlab="Scaled group counts")
dev.off()
```

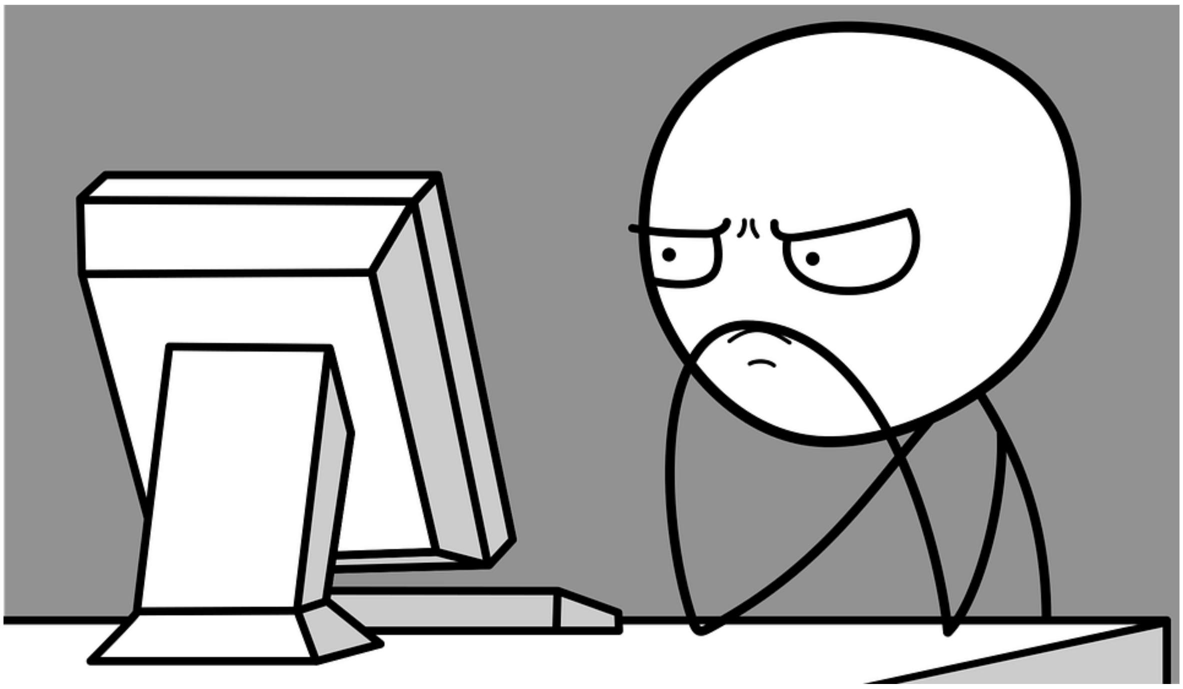
Scaled Apis mellifera orthology landscape



Skip to "Submit to conclude this mission."

We're here to help!

Come and find us during the breaks etc. and we'll see what we can do to help.



Skip to "Submit to conclude this mission."

Submit to conclude this mission