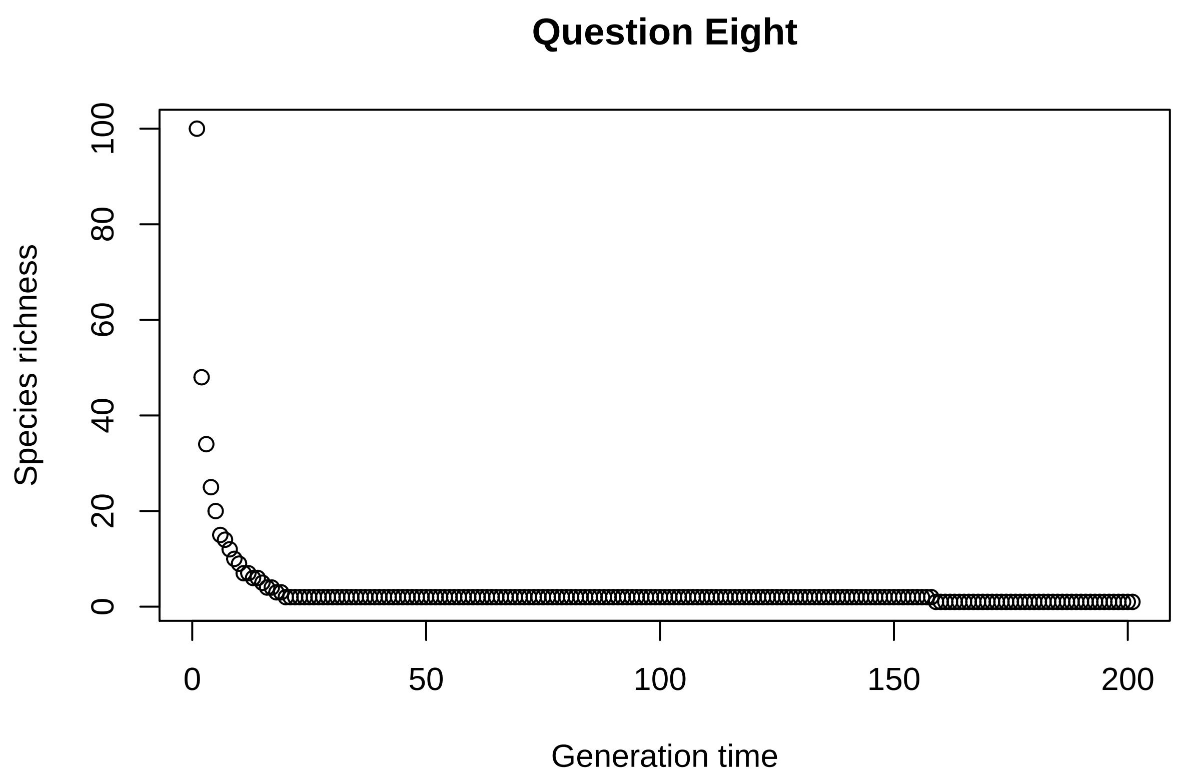
HPC_Excercises

Abigail Millward

8 December 2017

```r
#Question 8

question_8 = function(){
  plot(neutral_time_series(initialise_max(100), 200), xlab = "Generation time",
       ylab = "Species richness", main = "Question Eight")
}

question_8()
```

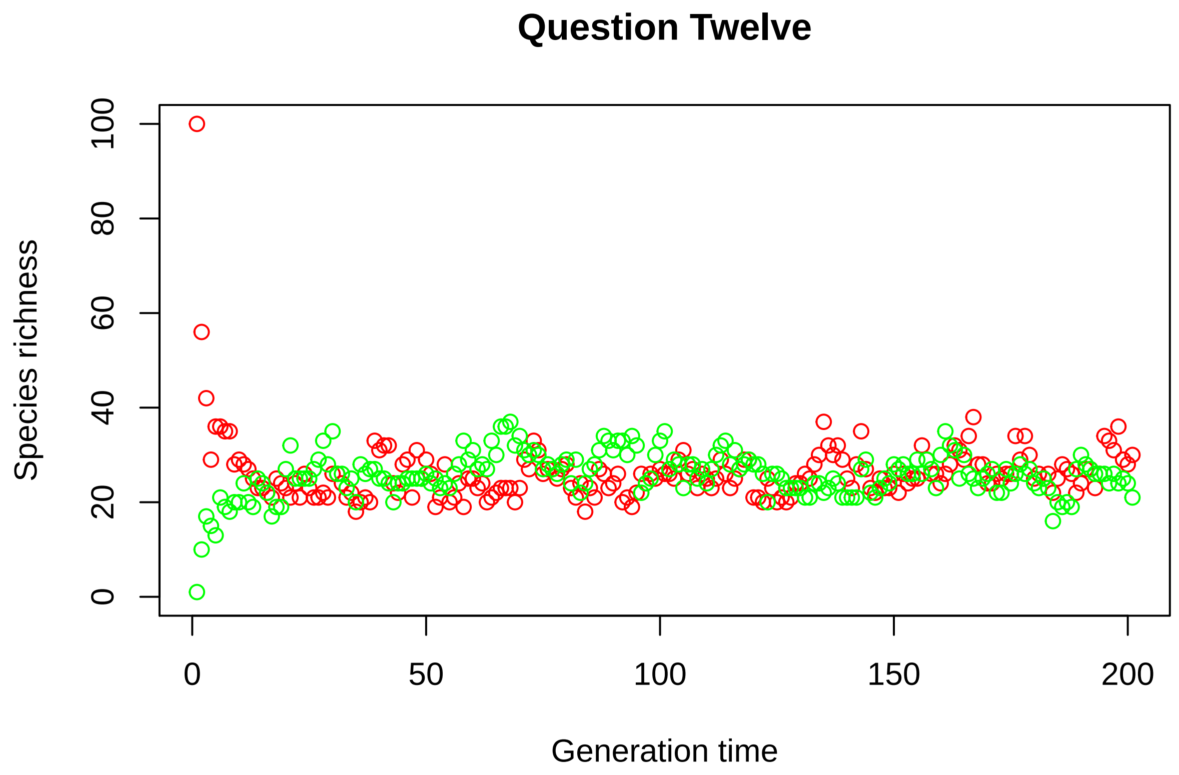


This graph shows the species richness of a community over time, in generations. The plot is of a neutral theory model without speciation and with zero sum assumption.

The plot shows that given enough time, species richness will decline to ~1; this is due to the fact that there is no speciation/ mutation, and without this eventually all individuals will become the same. This is because as generations progress and individuals die, the likelihood that they are replaced by a particular species increases as that particular species becomes more frequent in the population, thus the loop continues until only that particular species remains.

```r
#Initialising constants for Q12

spec_rate = 0.1
J = initialise_max(100)
generation = 200
X = initialise_min(100)
```

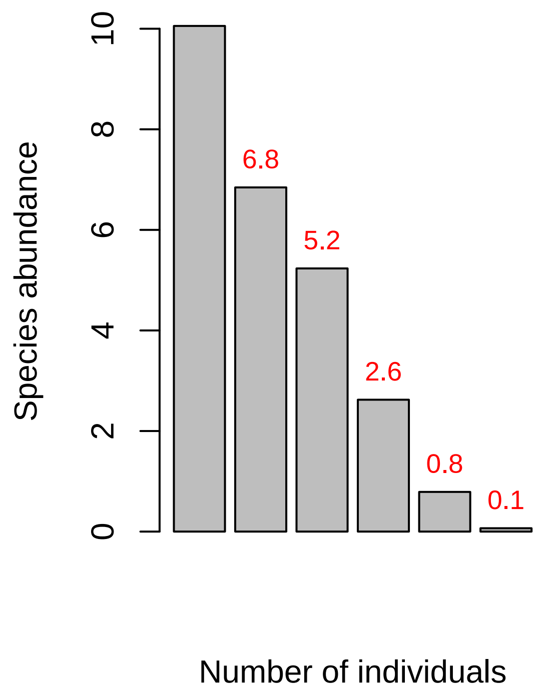
```
x1 = neutral_time_series_speciation(J, spec_rate, generation)


question_12 = function(){
  plot(neutral_time_series_speciation(J, spec_rate, generation),
       ylim = range(0,100), xlab = "Generation time", ylab = "Species richness",
       main = "Question Twelve", col = "red")
  points(neutral_time_series_speciation(X, spec_rate, generation),
         col="green", ylim = range(0,100))
}

question_12()
```

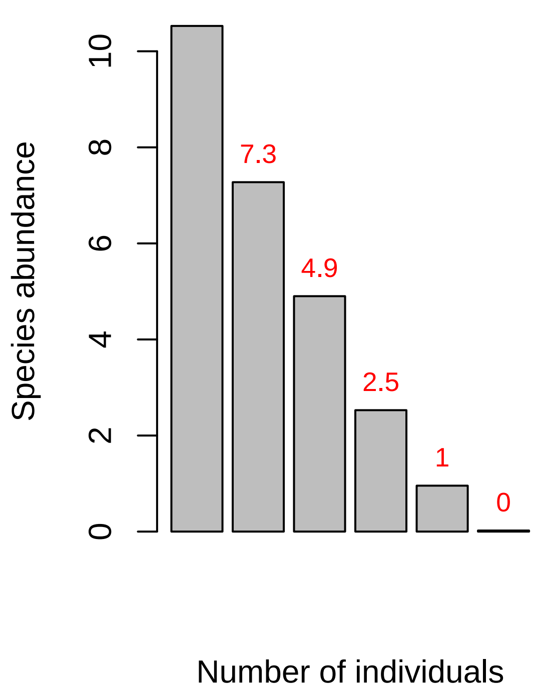


This graph is similar to that of Question 8, the difference is that this graph also has a speciation rate of 10%. Because of the speciation rate, the number of species converges around 25 rather then 1. The initial conditions of this model do not affect the overall result, as the speciation rate controls most of this model; changing the speciation rate changes the overall outcome. This is because new species are constantly arising in the population, and eventually the likelihood of removing a species (extinction) equals the likelihood of creating a new species (immigration/ speciation), and for this graph, the balance is ~ 25 species.

```
#Question 16

# %% remainder i.e. 7 %% 2 = 1 as 7/2 3 times, with 1 leftover

#Question 16
Question_16 = function(){
  plotty = neutral_time_series_speciation_nsr(J, spec_rate, 2000)
  for_plot = plotty[[220]] #Inc 220 is 1st after burnin
  c = 1
  for (i in 221:2001){       #After the "burn in period"
```

```
    if (i %% 20 == 0){
      c = c + 1
      for_plot = sum_vect(for_plot, plotty[[i]]) #Recording species abundance octaves as vector
     }
  }
  for_plot = for_plot/ c #Calc average
  xx <- barplot(for_plot, xlab = "Number of individuals", ylab = "Species abundance")
  ## Add text at top of bars
  text(x = xx, y = for_plot, label = round(for_plot, 1), pos = 3, cex = 0.8, col = "red")
}


#Question 16
Question_16_2 = function(){
  plotty = neutral_time_series_speciation_nsr(c(initialise_min(70),
                                                initialise_max(30)), spec_rate, 2000)
  for_plot = plotty[[201]]
  c = 1
  for (i in 201:2001){
    if (i %% 20 == 0){
      c = c + 1
    #for_plot = c(for_plot, octaves(species_abundance(plotty[i])))
      for_plot = sum_vect(for_plot, plotty[[i]])
     }
  }
  for_plot = for_plot/ c
  len = length(for_plot)
  xx <- barplot(for_plot, xlab = "Number of individuals", ylab = "Species abundance")
  ## Add text at top of bars
  text(x = xx, y = for_plot, label = round(for_plot, 1), pos = 3, cex = 0.8, col = "red")
}

par(mfrow= c(1,2))
Question_16()
Question_16_2()
```



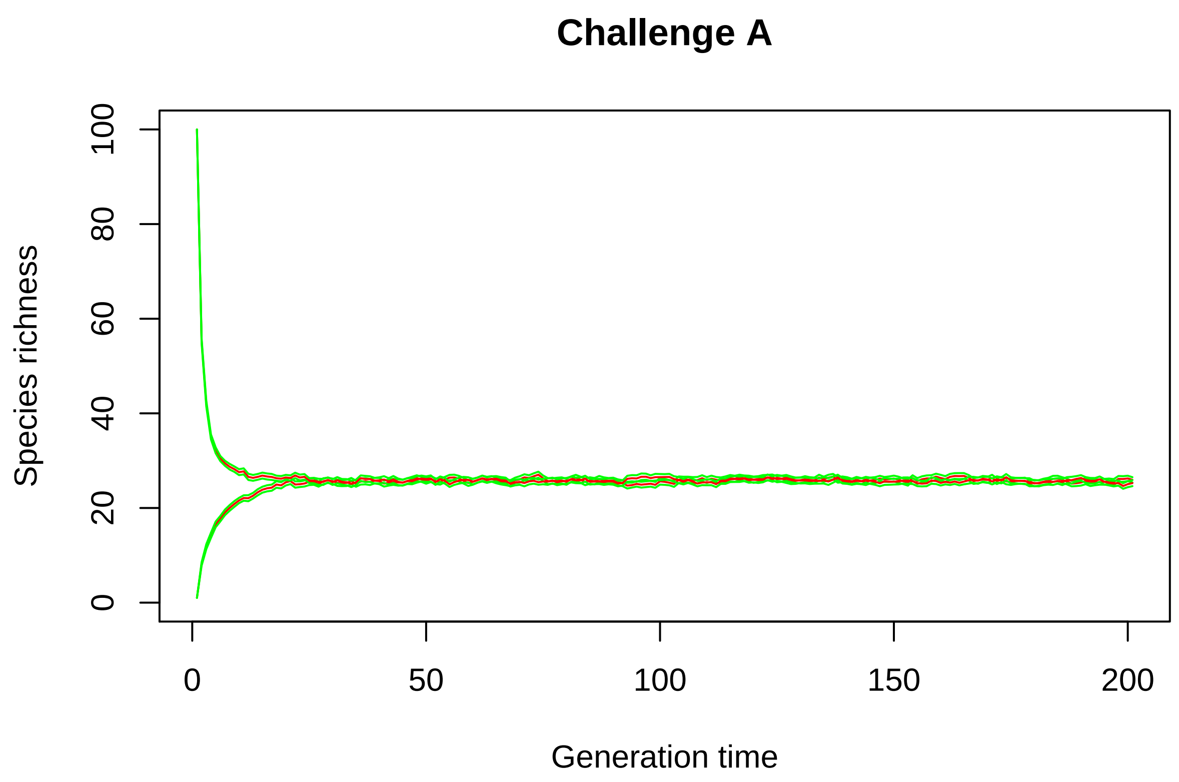


These plots show the average species abundance distribution (as octaves) after a burn in period of 200 generations.

The graph on the left has the original parameters, whereas the graph on the right had a different initial community. Despite this, the overall outcome has not significantly changed. This is because the speciation rate has the biggest impact, and so regardless of initial conditions, if the speciation rate stays the same, then the results after the burn in period will be very similar.

If a shorter burn in period was used, or less generations in total, then a bigger difference would be seen as it takes time for the impact of the initial conditions to be negligible, especially if extreme conditions were used.

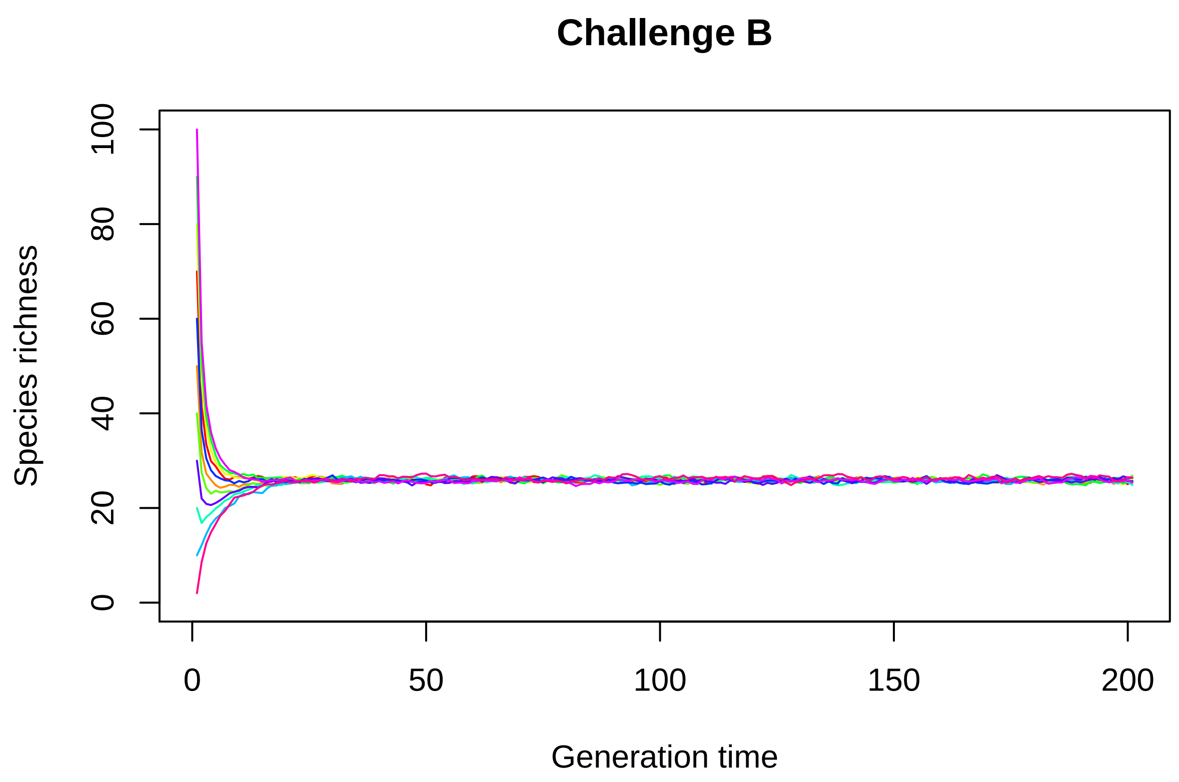
```
#Challenge A



multi_run = replicate(100, neutral_time_series_speciation(X, spec_rate, generation))
len = nrow(multi_run) # running simulation multiple times
multi_run2 = replicate(100, neutral_time_series_speciation(J, spec_rate, generation))
len = nrow(multi_run2)

total_average = function(mult){ #creating total average function,
  total_av = mean(mult[1,])    # couldn't figure out lappy with the replicate outcome
  for (i in 2:len){
    av = mean(mult[i,])
    total_av = c(total_av, av)
  }
  return(total_av)
}

total_sd = function(mult){ #total sd function
  total_std = sd(mult[1,])
  for (i in 2:len){
    standd = sd(mult[i,])
    total_std = c(total_std, standd)
  }
  return(total_std)
}
std1 = total_sd(multi_run)

CI = function(av, std){ # Confidence interval function
  error = qnorm(0.986) * (std[1]/sqrt(len))
  top = av[1] + error
  bottom = av[1] - error
  tot_top = top
  tot_bot = bottom
  for (i in 2:len){
    error = qnorm(0.986) * (std[i]/sqrt(len))
    top = av[i] + error
    bottom = av[i] - error
    tot_top = c(tot_top, top)
    tot_bot = c(tot_bot, bottom)
  }
  return(list(tot_top,tot_bot))
}


means1 = total_average(multi_run)
means2 = total_average(multi_run2)
std2 = total_sd(multi_run2)
CI2 = CI(means2, std2)
CI1 = CI(means1,std1)
question_challengeA = function(){
    plot(means1, xlab = "Generation time",
         ylab = "Species richness", main = "Challenge A", col = "red",
         cex = 0.2, type = "l", ylim = range(0,100))
  lines(CI1[[1]], col="green")
  lines(CI1[[2]], col ="green")
  lines(means2,  col = "red", cex = 0.2)
  lines(CI2[[1]], col="green")
  lines(CI2[[2]], col ="green")
}

question_challengeA()
```



This graph shows mean species richness over time, with two initial conditions, max and min diversity. The green lines show the 97.2% confidence intervals of both conditions. From this graph I would confidently say that to reach dynamic equilibrium a burn in period of 30 generations would need to be set.

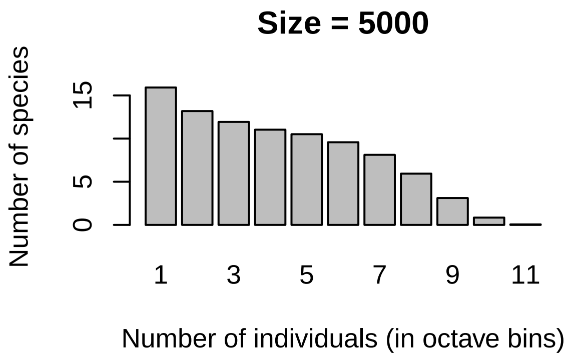
```
#Challenge_B


#replica = function(){
#   i = 1
#   for (i in 1:10){
#     run_list = c()
#     run = replicate(100, neutral_time_series_speciation(c
#                     (initialise_min(i * 10),initialise_max(100 - (i*10))), spec_rate, generation))
#     run_list = c(run_list, run)
#     i = i+1
#     return(run_list)
#   }
#}

#a = replica()


run1 = replicate(100, neutral_time_series_speciation(c(initialise_min(30)
                ,initialise_max(70)), spec_rate, generation))
run2 = replicate(100, neutral_time_series_speciation(c(initialise_min(50),
                 initialise_max(50)), spec_rate, generation))
run3 = replicate(100, neutral_time_series_speciation(c(initialise_min(20),
                 initialise_max(80)), spec_rate, generation))
run4 = replicate(100, neutral_time_series_speciation(c(initialise_min(60),
                 initialise_max(40)), spec_rate, generation))
```

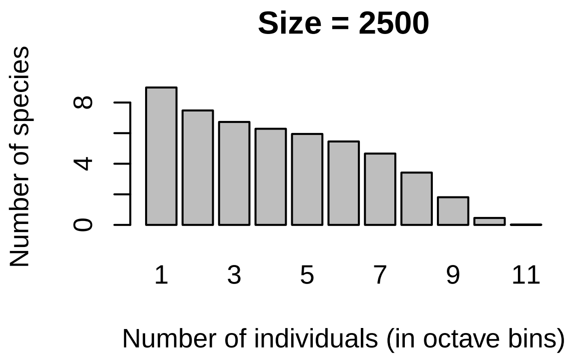
```r
run5 = replicate(100, neutral_time_series_speciation(c(initialise_min(10),
                initialise_max(90)), spec_rate, generation))
run6 = replicate(100, neutral_time_series_speciation(c(initialise_min(80),
              initialise_max(20)), spec_rate, generation))
run7 = replicate(100, neutral_time_series_speciation(c(initialise_min(90),
                initialise_max(10)), spec_rate, generation))
run8 = replicate(100, neutral_time_series_speciation(c(initialise_min(40),
                initialise_max(60)), spec_rate, generation))
run9 = replicate(100, neutral_time_series_speciation(c(initialise_min(70),
                initialise_max(30)), spec_rate, generation))
run10 = replicate(100, neutral_time_series_speciation(c(initialise_min(0),
                initialise_max(100)), spec_rate, generation))
run11 = replicate(100, neutral_time_series_speciation(c(initialise_min(100),
                initialise_max(0)), spec_rate, generation))

run1av = total_average(run1)
run2av = total_average(run2)
run3av = total_average(run3)
run4av = total_average(run4)
run5av = total_average(run5)
run6av = total_average(run6)
run7av = total_average(run7)
run8av = total_average(run8)
run9av = total_average(run9)
run10av = total_average(run10)
run11av = total_average(run11)

question_challengeB = function(){
  cl = rainbow(11)
  plot(run1av, xlab = "Generation time",
       ylab = "Species richness", main = "Challenge B", col = cl[1],
       cex = 0.2, type = "l", ylim = range(0,100))
  lines(run2av, col= cl[2])
  lines(run3av, col = cl[3])
  lines(run4av, col = cl[4])
  lines(run5av, col= cl[5])
  lines(run6av, col = cl[6])
  lines(run7av, col = cl[7])
  lines(run8av, col = cl[8])
  lines(run9av, col = cl[9])
  lines(run10av, col = cl[10])
  lines(run11av, col = cl[11])
}
question_challengeB()
```

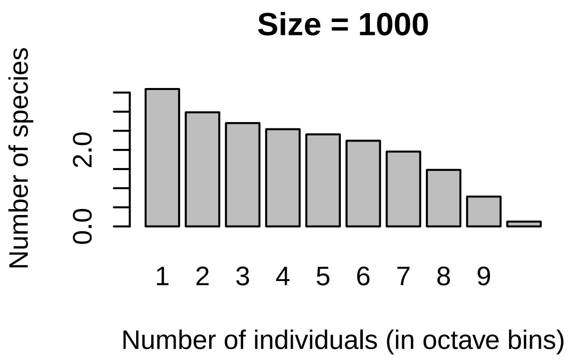


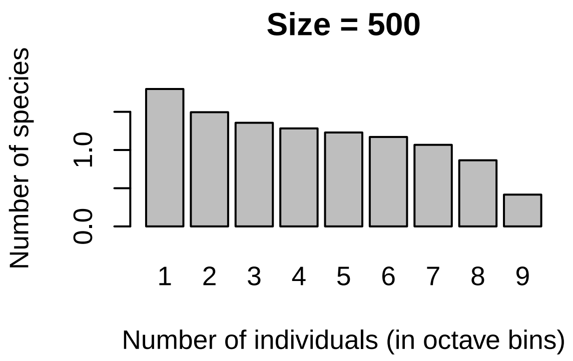
This graph shows many averaged time series, all with different starting conditions, ranging from initialise_min(100) to initialise_max(100). All time series had the same likelihood of taking any species identity.

```
#Q 20
names = c(1,2,3,4,5,6,7,8,9,10,11)
abundance_500 = calc(1)
abundance_1000 = calc(26)
abundance_2500 = calc(51)
abundance_5000 = calc(76)
par(mfrow = c(2,2))
plot_5000 <- barplot(abundance_5000, main = "Size = 5000",
     names.arg = names, xlab = "Number of individuals (in octave bins)", ylab = "Number of species")
plot_2500 <- barplot(abundance_2500, main =  "Size = 2500",
     names.arg = names, xlab = "Number of individuals (in octave bins)" ,ylab = "Number of species")
plot_1000 <- barplot(abundance_1000, main = "Size = 1000",
  names.arg = names[1:10], xlab = "Number of individuals (in octave bins)" ,ylab = "Number of species")
plot_500 <- barplot(abundance_500, main = "Size = 500",
  names.arg = names[1:9],  xlab = "Number of individuals (in octave bins)", ylab = "Number of species")
```





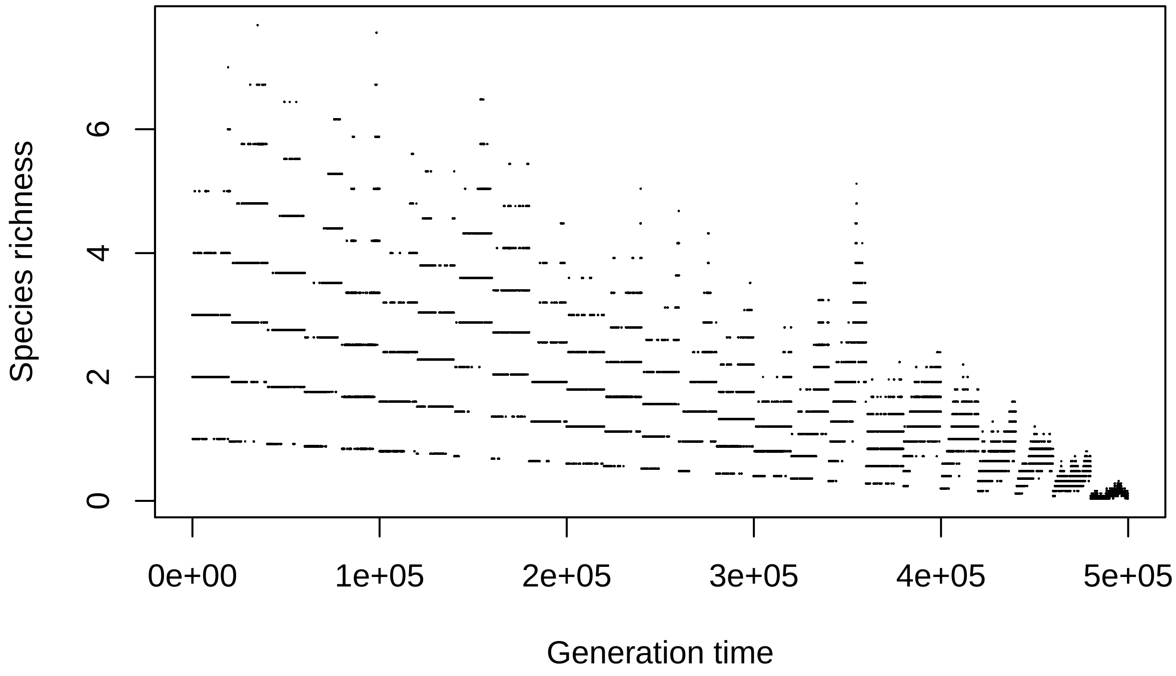




These figures show mean species abundances (in octaves), for all simulation sizes after the burn in period. The structure of all the graphs is similar; most species only have a few individuals, with no species reaching beyond 11 octave bins.

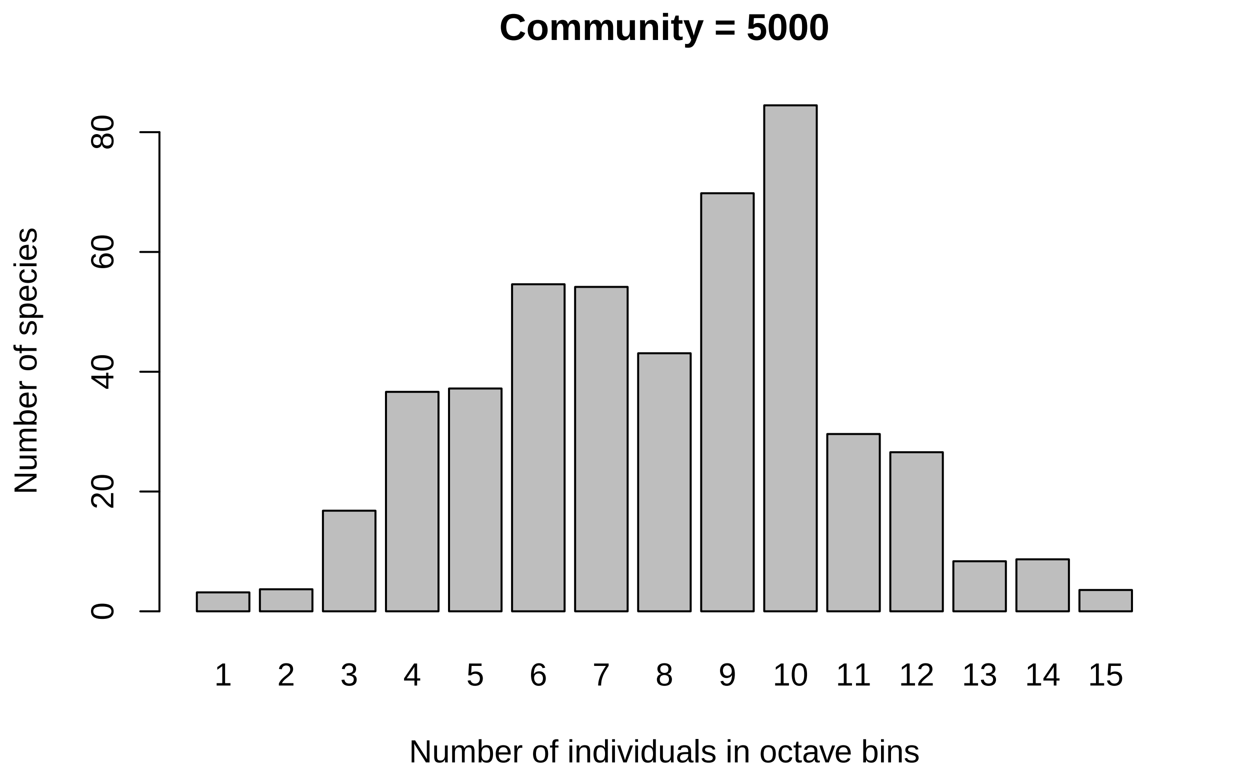
```
challenge_C = function(iterbegin){
  iter = iterbegin
  mean_store = c()
  vect = c()
  while (iter >= iterbegin & iter <= (iterbegin + 24)){
    file = paste("my_test_file", iter, ".rda", sep = "_")
    file = paste("../Results/", file, sep = "")
    load(file)
    n = length(spec_rich_store)/100
    start_no = 1
    it = start_no
    if (n > start_no){
      for (it in start_no:n){
        summing = length(unique(spec_rich_store[it:(it+99)]))
        vect = c(vect, summing)
        it = it + 99
      }
    }
    iter = iter + 1
    mean_store = sum_vect(mean_store, vect)
  }
  return(mean_store/25)
}

richness_500 = challenge_C(1)
x = 1:length(richness_500)
```

```r
plot(x =x, y =richness_500, xlab = "Generation time", ylab = "Species richness", cex = 0.01  )
```



The species richness begins to settle (between 0 - 2) at aroound 400000 generations, which could indicate that most of our HPC was wasted just burning in.

```r
################################################
#Challenge D
################################################

#Coalescence
lineages = function(j){
  return(rep(1, j))
}

challenge_D = function(N, iter ="2"){
  N = N
  a = as.numeric(proc.time())[3]    #Measuing time
  abundances = c()    # initialising empty abundance vector
  j = N   # original length, j will always stay the same
  lineage = lineages(j)     #initialising lineage
  O = abi_spec_rate*((j - 1) / (1 - abi_spec_rate))  #creating o
  index_j = round(runif(1, 1, length(lineage))) # picking random index
  randnum = runif(1) # picking random number
  while ( N > 1){
    if (randnum < (O/ (O + (N - 1)))){
      abundances = c(abundances,lineage[index_j])  # Adding lineage to abundance
    }
    else{
      index_i = round(runif(1, 1, length(lineage)))
      if(index_i != index_j){
        lineage[index_i] = lineage[index_i] + lineage[index_j] #Speciation
      }
      else{
        next
      }
    }
    lineage = lineage[-(index_j)]  #Removing the added lineage
    N = N -1 #same here
    randnum = runif(1) #initialising new random number
    index_j = round(runif(1, 1, length(lineage))) # and new index
  }
  final_time = (as.numeric(proc.time())[3] - a)/60
  abundances = c(abundances, lineage)
  abundances = octaves(species_abundance(abundances))
  save(abundances, final_time, file = paste("../Results/Challenge_D", iter, ".rda", sep = "_" ))
}



do_simulation_CD = function(){ #Similar to do_sim for HPC, allowing 100 simulations to run
  a = as.numeric(proc.time())[3]
  iter = 1
  while (iter <101) {
  if (iter < 26){
    N = 500
    set.seed(iter)
    challenge_D(N, iter)
    iter = iter + 1
  }
  else if (iter < 51){
    N = 1000
    set.seed(iter)
    challenge_D(N, iter)
    iter = iter + 1
  }
  else if (iter < 76){
    N = 2500
    set.seed(iter)
    challenge_D(N, iter)
    iter = iter + 1
  }
  else if (iter < 101){
    N = 5000
    set.seed(iter)
    challenge_D(N, iter)
    iter = iter + 1
  }
  }
  return(as.numeric(proc.time())[3] - a)
}


for_plotting = function(){
  abundances_v = c()
  iter = 1
  while (iter < 26){
    file = paste("../Results/Challenge_D", iter, ".rda", sep = "_" )
    load(file)
    abundances_v = sum_vect(abundances_v, abundances)
    iter = iter + 1
  }
  return(abundances_v)
}

plt = for_plotting()
barplot(plt/25, xlab = "Number of individuals in octave bins", ylab = "Number of species",
        main = "Community = 5000", names.arg = c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15))
```



The coalescence code above ran 100,000 times faster than the HPC code; do_simulation_CD() returns a time of approximately 7 seconds. This is due to their being no burn in period with coalescence, and working backwards means that there is less irrelevant code and data; only the present individuals' lineages are worked out.

The plot above is of species richness in octaves, in a community size of 5000. This graph is different to those of question Q20, as coalescence only produces the final species richness abundance, whereas Q20 graphs were produced by averaging abundances after burn in time was up until the end of the simulation.

```
######################
#Question 18#
######################

#x = a + ib

width1 <- 3
size1 <- 8
```

```r
#size1 = width1^x
#log(size1) = log(width1^x)
#log(size1) = x * log(width1)
x = log(size1)/log(width1)

#The first object has a dimension of 1.893

width2 <- 3
size2 <- 20

x = log(size2)/log(width2)

#Second dimension is 2.727
```

The code above was used to calculate the dimensions of the objects. The first object had was between 1 and 2 dimensional, and it had a width of 3 (3 mini boxes fit into it's width) and a size of 8(total mini boxes making up the main box). From that I used the size1 = width1ˆx equation to work out that the dimension of the object was 1.893. I used the same equation to calculate the 2-3d object, which had a width of 3 (3 mini cubes made the main cube) and a size of 20 (20 mini cubes to 1 big cube). This resulted in an estimate of 2.727 as the dimensioon.

```r
chaos_game = function(){
  graphics.off()
  a = c(0,0)
  b = c(3,4)
  c = c(4,1)
  to_samp = list(a,b,c)
  x0 = c(0,0)
  plot(0:5, 0:5, type = "n")
  #axis(side = 1)
  #axis(side = 2)
  points(x = x0[1],y= x0[2], cex = 0.2)
  i = 1
  while (i <= 10000) {
    samp = sample(c(1,2,3), 1)
    x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
    new_point = c(((x0[1] + x[1])/2), ((x0[2] + x[2])/2))
    points(x = new_point[1], y = new_point[2], cex = 0.2)
    x0 = new_point
    i = i +1
  }
}

chaos_game()
```

```r
challenge_E = function(){
  graphics.off()
  a = c(0,0)
  b = c(3,4)
  c = c(4,1)
  to_samp = list(a,b,c)
  x0 = c(2,4)
  plot(0:5, 0:5, type = "n")
  #axis(side = 1)
  #axis(side = 2)
  points(x = x0[1],y= x0[2], col = "green", pch = 23)
  i = 1
  n = 1
  while (i <= 10000) {
    while (n <= 100) {
      cl <- rainbow(100)
      samp = sample(c(1,2,3), 1)
      x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
      new_point = c(((x0[1] + x[1])/2), ((x0[2] + x[2])/2))
      points(x = new_point[1], y = new_point[2], col = cl[n], pch = 23)
      x0 = new_point
      i = i +1
      n = n+1
    }
    samp = sample(c(1,2,3), 1)
    x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
    new_point = c(((x0[1] + x[1])/2), ((x0[2] + x[2])/2))
    points(x = new_point[1], y = new_point[2], cex = 0.2)
    x0 = new_point
    i = i +1
  }
}




challenge_E2 = function(){
  graphics.off()
  a = c(0,0)
  b = c(2,4)
  c = c(4,0)
  to_samp = list(a,b,c)
  x0 = c(1,2)
  plot(0:5, 0:5, type = "n")
  #axis(side = 1)
  #axis(side = 2)
  points(x = x0[1],y= x0[2], col = "green", pch =14 )
  i = 1
  n = 1
  while (i <= 1000) {
    while (n <= 100) {
      cl <- rainbow(100)
      samp = sample(c(1,2,3), 1)
      x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
      new_point = c(((x0[1] + x[1])/2), ((x0[2] + x[2])/2))
      points(x = new_point[1], y = new_point[2], col = cl[n] , pch =14)
      x0 = new_point
      i = i +1
      n = n+1
    }
    samp = sample(c(1,2,3), 1)
    x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
```

```r
    new_point = c(((x0[1] + x[1])/2), ((x0[2] + x[2])/2))
    points(x = new_point[1], y = new_point[2], cex = 0.2)
    x0 = new_point
    i = i +1
  }
}





challenge_E3 = function(){
  graphics.off()
  a = c(0,0)
  b = c(0,4)
  c = c(4,0)
  d = c(4,4)
  to_samp = list(a,b,c,d)
  x0 = c(2,2)
  plot(0:5, 0:5, type = "n")
  #axis(side = 1)
  #axis(side = 2)
  points(x = x0[1],y= x0[2], col = "green")
  i = 1
  n = 1
  while (i <= 1000) {
    while (n <= 100) {
      cl <- rainbow(100)
      samp = sample(c(1,2,3,4), 1)
      x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
      new_point = c(((x0[1] + x[1])/3), ((x0[2] + x[2])/3))
      points(x = new_point[1], y = new_point[2], col = cl[n])
      x0 = new_point
      i = i +1
      n = n+1
    }
    samp = sample(c(1,2,3,4), 1)
    x = c((to_samp[[samp]][1]), to_samp[[samp]][2])
    new_point = c(((x0[1] + x[1])/2), ((x0[2] + x[2])/2))
    points(x = new_point[1], y = new_point[2], cex = 0.2)
    x0 = new_point
    i = i +1
  }
}
challenge_E()
```

```r
challenge_E2()
```

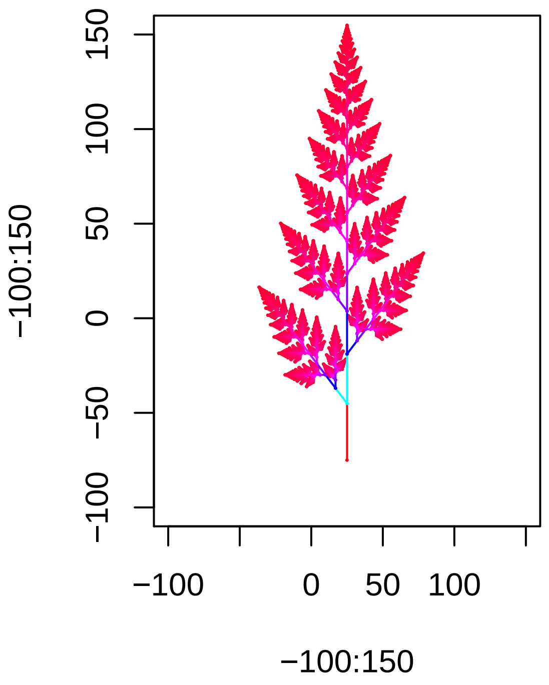
```r
challenge_E3()
```

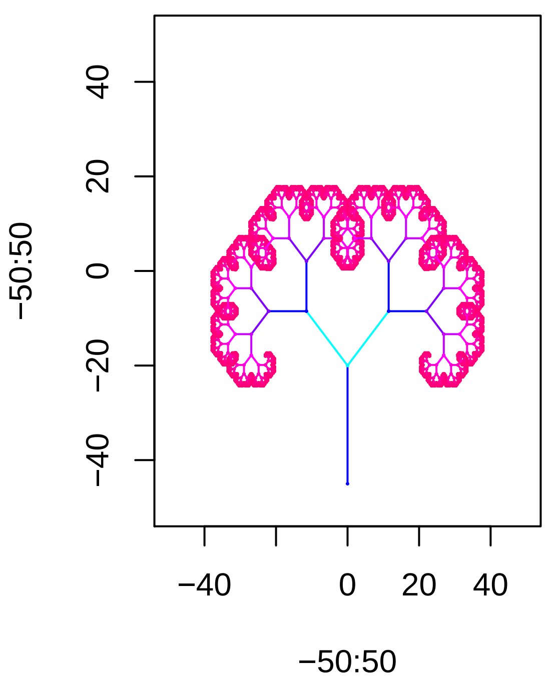
These graphs show variations of the chaos game. The first plot is using different initial positions. Despite the different initial conditions, the graph managed to realign itself and continue on as normal, this is due to the nature of moving halfway towards one of the 3 points, and with none of them being the starting point it resulted in minimal difference to the outcome.

The second graph shows a classic Sierpinski Gasket, similar to that of the chaos game, but with an equilateral triangle.

The final graph shows a graph with 4 initial points and a movement of 1/3 towards the new point. This graph doesn't look like the other fractals as it is not self similar, this is because it does not move halfway between points, recreating itself and producing repeatability. Instead, it truly is chaos.

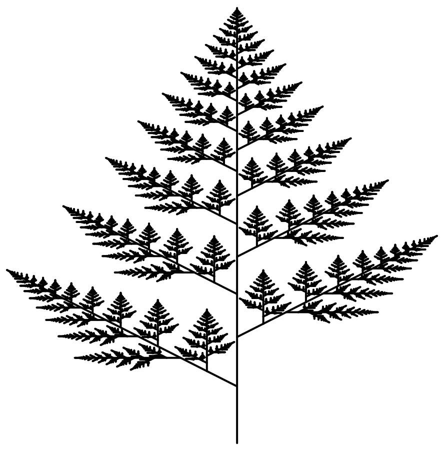
```
challenge_F = function(start, direction, len, dir, col = rainbow(n), n = 1){
  if (len > 0.1){
    if (dir == -1){
      a = turtle(start, direction, len, col = col[n])
      challenge_F(a, (direction-45), (0.38*(len)),-dir, n = n + 1)
      challenge_F(a, direction , (0.87*(len)), -dir, n = n + 1)
    }
    else{
      a = turtle(start, direction, len, col = col[n])
      challenge_F(a, (direction+45), (0.38*(len)),-dir, n = n + 1)
      challenge_F(a, direction , (0.87*(len)), -dir, n = n + 1)
    }
  }
}

par(mfrow = c(1,2))

plot(-100:150,-100:150, type="n")
challenge_F(c(25,-75), 90, 30, 1)


challenge_F2= function(start, direction, len, col = rainbow(n), n = 1){
  if (len > .1){
    a = turtle(start, direction, len, col[n])
    challenge_F2(a, (direction+45), 0.65*(len), n = n +1)
    challenge_F2(a, (direction-45), 0.65*(len), n = n +1)
  }
}


plot(-50:50, -50:50, type = "n")
challenge_F2(c(0,-45), 90, 25, "blue")
```





The image produced gets much clearer the longer it runs, but the time it takes gets expontentially longer, as each call of the function then calls 2 more, etc. This is why it is necessary to make it in a loop which cuts off at a certain length.

```
challenge_G <- function(start, direction, len, dir){
  #Tried changing argument names to make them shorter
   #But I just would't work and spent a few days on this now
   #So I'm happy with where it is now
  if (len > .1){
    challenge_G1 <- function(start, direction, len, dir){
        new_xy = c(((len * cos(direction))+start[1]), (len * sin(direction)+start[2]))
        x = c(start[1], new_xy[1])
        y = c(start[2], new_xy[2])
        lines(x,y)
        return(new_xy)
     }
      if (dir == -1){ d = direction - pi/4}
      else { d = direction + pi/4}
      a = challenge_G1
      b = a(start, direction,len, dir)
      challenge_G(b, d, .38*len, -dir)
      challenge_G(b, direction, .87*len, -dir)
  }
}


plot(-100:150,-100:150, type="n", axes=FALSE, ann=FALSE)
challenge_G(c(25,-75), pi/2, 30, 1)
```



This is the final challenge; the task was to see how short we could make the function with it still working. Despite my function not being short (over 4x the length of the shortest code), I did figure out how to create functions within functions and call them properly to produce the same result as the original fern. I am pleased with this outcome as I didn't think I would be able to do this at the beginning of the week, and also I believe it is a good skill to know despite not getting the shortest code.