Scientific Programming - Assignment 1

Contents

1 What is this?

This document is the work of **a random student**, completed as an assignment for the Scientific Programming module as part of MPhil Computational Biology at DAMTP.

2 Results:

2.1 Analysing Words

2.1.1 Q: How many unique words are there (ignoring cases)?

```
## A: 97723
```

2.1.2 Q: How many words contain an apostrophe(')?

words.match_idx.
A'S
AA'S
AB'S
ABM'S
AC'S
ACTH'S
AI'S
AIDS'S
AM'S
AOL'S

```
## A: 25751
## 71972 words left after filtering
```

2.1.3 Q: How many words contain non-ASCII chars? Save the filtered words as 'database'.

words.match_idx.
ASUNCIÓN
ATATÜRK
BARTÓK
BOGOTÁ
BOÖTES
BUÑUEL
CONCEPCIÓN
DVORÁK
DÜRER
DÜSSELDORF

```
## A: 159
## 71813 words left after filtering, saved to "words_data"
```

2.1.4 Q: Find all the words as the two related form XOG and XOGUE, for example CATALOG and CATALOGUE.

```
## 10 pairs of words were found to conform "XOG"->"XOGUE" projection
```

Table 3: Showing best 10 hits below:

XOG	XOGUE
ANALOG	ANALOGUE
CATALOG	CATALOGUE
DEMAGOG	DEMAGOGUE
DIALOG	DIALOGUE
EPILOG	EPILOGUE
MONOLOG	MONOLOGUE
PEDAGOG	PEDAGOGUE
PROLOG	PROLOGUE
SYNAGOG	SYNAGOGUE
TRAVELOG	TRAVELOGUE

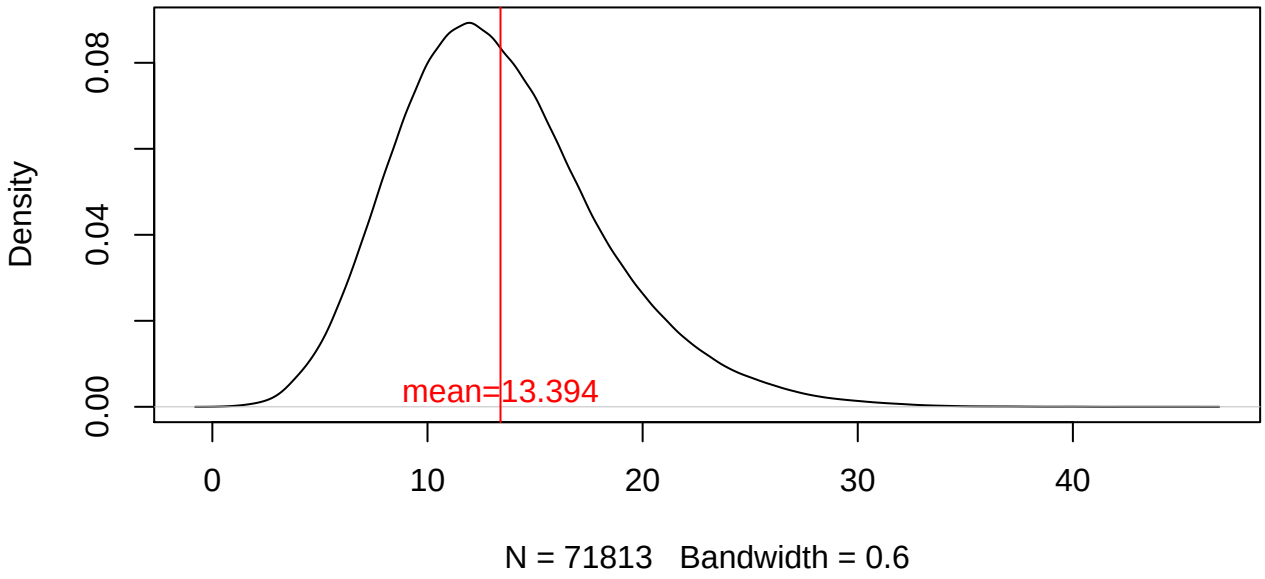
2.1.5 Q: Load scrabble scores for the alphabet from "scarabble.txt", store to "scores"

Table 4: Showing best 10 hits below:

	scores
Q	10
Z	10
J	8
X	8
K	5
F	4
H	4
V	4
W	4
Y	4

2.1.6 Q: Compute the scrabble score for each word in the database. Plot the distribution of scores. What is the highest-scoring word

(Summation of score is assumed)




```
## Highest score is achieved by: PIZZAZZ , Score: 45
```

2.1.7 Q: Find all words W where both W and its reverse complement are both in the database.

```
## We found 68 entries, out of which 16 are reverse-complement to themselves , Showing longest 10 entrie
```

Table 5: Showing best 10 hits below:

smaller_word	larger_word
BOORISH	SHRILLY
HOVELS	HOVELS
WIZARD	WIZARD
BOORS	HILLY
HOOFS	HULLS
ARIZ	ARIZ
BEVY	BEVY
BIRD	WIRY
BOIL	ORLY
GILL	OORT

2.1.8 Q: Using "F A L U Y P L N I", how many words of four or more letters can you find that are in the database AND all contain the letter A?

```
## We found 39 hits according to query "LLFAUYPNI" in "words_data"",
##  (Option: min_len=4, special=A).
```

Table 6: Showing best 20 hits below:

	words_res	words_len	words_score
44850	PAINFULLY	9	9

	words_res	words_len	words_score
23582	FINALLY	7	7
44840	PAILFUL	7	7
44847	PAINFUL	7	7
47295	PLAINLY	7	7
47419	PLAYFUL	7	7
25465	FULANI	6	6
23572	FINAL	5	5
23863	FLAIL	5	5
32408	INLAY	5	5
45662	PAULI	5	5
46955	PILAF	5	5
46963	PILAU	5	5
47289	PLAIN	5	5
1332	AINU	4	4
1705	ALLY	4	4
22746	FAIL	4	4
22753	FAIN	4	4
22807	FALL	4	4
23063	FAUN	4	4

2.2 Examination Marking

2.2.1 Importing and Scoring:

```
## [DEBUG]:There are 100 questions from file:crib.dat
```

Table 7: Result of 12 students in this exam, sorted by percent score in descending order

student_idx	score	percent	grade	rank
7	29	96	A	1.0
2	24	80	A	2.0
6	23	76	A	3.0
9	22	73	A	4.0
5	20	66	B	5.0
1	19	63	B	6.0
12	18	60	B	7.0
4	17	56	C	8.5
10	17	56	C	8.5
3	14	46	D	10.0
11	9	30	F	11.0
8	7	23	F	12.0

```
## Stats were calculated for the rounded percent score for 12 students in the exam
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   23.00   53.50   61.50   60.42   73.75   96.00
```

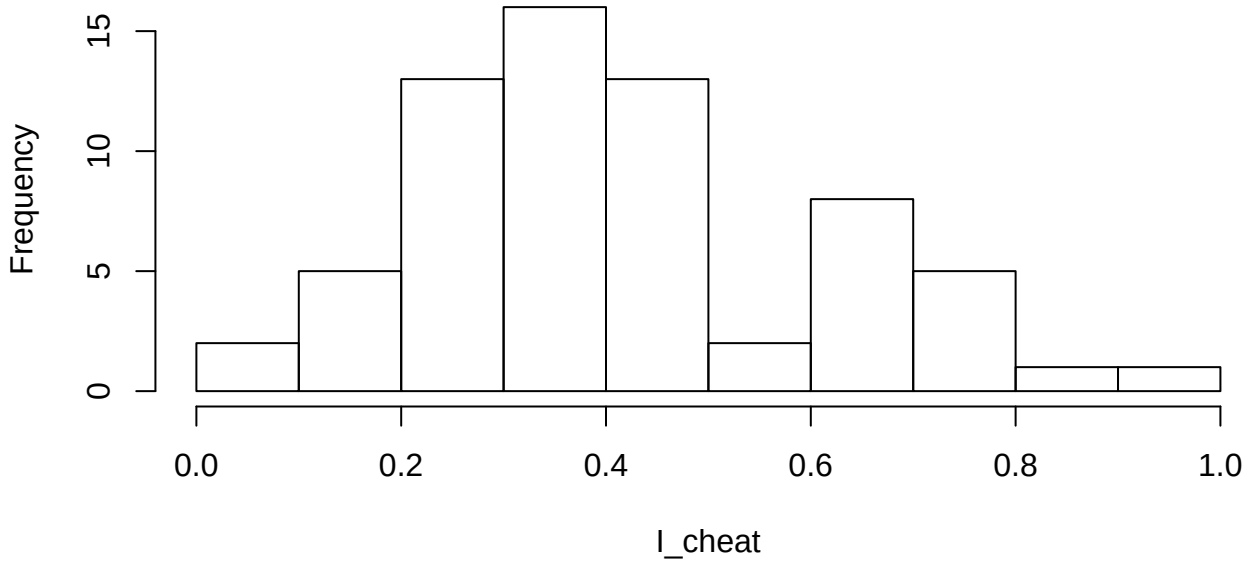


Figure 1: Distribution of I_{cheat}

2.2.2 Detecting cheating

2.2.2.1 Naive approach

We define cheat index between two students, as the number of questions with a same answer, divided by the product of overlapped questions.

$$I_{cheat}^{(a,b)} = \frac{\sum_i \delta_{A_i^a, A_i^b}}{\sum_i \delta_{Q_i^a, Q_i^b}}$$

where a, b denotes index of students, A_i^a denotes answer to question i by student a, Q_i^a denotes whether question i is answered by student a, $\delta_{x,y}$ defined as

$$\delta_{x,y} = \begin{cases} 0 & x \neq y \\ 1 & x = y \end{cases}$$

Specifically, if any of the A_i^a, A_i^b is undefined, then $\delta_{A_i^a, A_i^b} = 0$.

Normalisation against overlapped questions count is important, because students can give the same answer by chance even without cheating. However, the calculated cheat index appear to follow a normal distribution, without a significant outlier. (See Figure 1)

2.2.2.2 Diagnostic approach

To obtain a more solid result, we plotted the numerator of I_{cheat} against its denominator, allowing for a careful examination of the relation between overlapped answers and overlapped questions. We then fitted this relation with linear regression $N_{answer} = a \cdot N_{question} + \epsilon$, estimated the standard deviation of the resultant residual $\sigma(\epsilon)$. The estimated standard deviation is then used to normalise the residual distribution, and to calculate the corresponding P-value, indicating our confidence to reject the null hypothesis that the point is sampled from this distribution (Figure 2).

From a visual inspection, point 15 comparison is obviously an outlier to the other clustered points. The normalised stats can be found at Table 8. We conclude student 2 and student 6 (graded A and B respectively) are highly likely to have cheated (with a 99.99994% confidence), giving identical answers to all 27 quesions that were allocated to both of them.

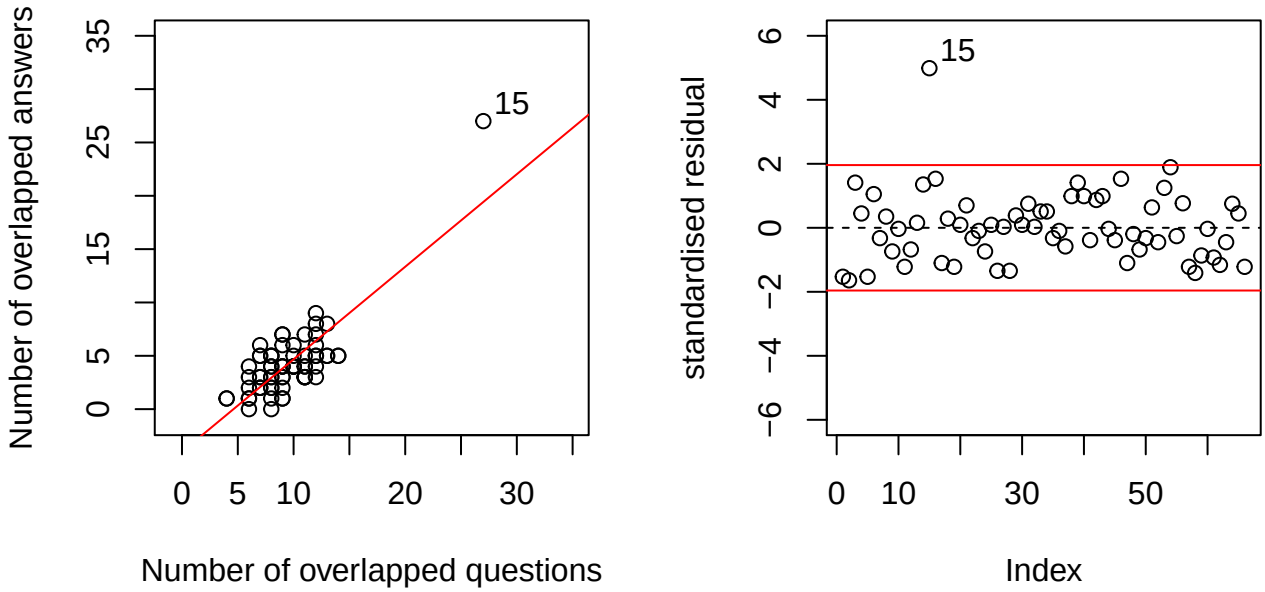


Figure 2: Diagnostic plots for finding the outlier/cheater(Using 95.000 % confidence (2 tailed) as threshold)

We also detected other students sharing a large number of answers, but associated with less significant z-score and are subject to further discussion. , we need to account for the fact that some questions are so easy that every student get it right, thus giving the same answer. The difficulty can be measured with Shannon entropy, and can be used to score the observed conincidence, instead of simple scoring by counting.

We refrain from a thorough treatment given the size limit of this report.

Table 8: Showing best 10 hits below:

	stud_A	grad	stud_B	grad.1	ques_same	ans_same	Confidence_to_reject_NULL
15	2	A	6	B	27	27	0.9999994
54	7	B	10	D	7	6	0.9415630
16	2	A	7	B	9	7	0.8743686
46	6	B	7	B	9	7	0.8743686
3	1	A	4	A	7	5	0.8417347
39	5	B	6	B	7	5	0.8417347
14	2	A	5	B	6	4	0.8240609
53	7	B	9	C	12	9	0.7882032
6	1	A	7	B	9	6	0.7073157
38	4	A	12	F	8	5	0.6779376

2.3 Treatment to Chapter 10 from "Dynamical Models in Biology" [1]

2.3.1 Introduction

In ecology, people study population growth from a spatial perspective. This could be done by discretising space into a lattice of cells, and specify a recursion formula to account for both the constitutive geometric growth the spatial dispersal. Here we consider a such model from [1] where the population $N(s,t)$ is a function of both space s and time t. Two phenomena are included in this model:

2.3.1.1 Geometric growth

(1)

$$M(s,t) = \lambda(s) \cdot N(s,t)$$

where M is an intermediate variable to be transformed back into N.

2.3.1.2 Spatial dispersal

(2)

$$N(s,t+1) = \begin{bmatrix} M(s-1,t), & M(s,t), & M(s+1,t) \end{bmatrix} \cdot \begin{bmatrix} d \\ 1-2d \\ d \end{bmatrix}$$

This is equivalent to convolve signal $M_t(s)$ with filter $D(i)$

$$N_{t+1}(s) = \sum_i M_t(s+i) \cdot D(i)$$

where

$$D(i) = \begin{cases} d & i \in \{-1, +1\} \\ 1-2d & i = 0 \\ 0 & for\ other\ i \end{cases}$$

Importantly, this filter implies a weighted sum of nearest neighbors, for we have

$$\sum_i D(i) = (d) + (1-2d) + (d) = 1$$

this is clearer if we set $K = s + i$ to obtain

(3)

$$\begin{aligned} \sum_s N_{t+1}(s) &= \sum_s \left(\sum_i M_t(s+i) \cdot D(i) \right) \\ &= \sum_s \left(\sum_K M_t(K) \cdot D(K-s) \right) \\ &= \sum_K \left(M_t(K) \cdot \sum_s D(K-s) \right) \\ &= \sum_K M_t(K) \cdot 1 \\ \sum_s N_{t+1}(s) &= \sum_K M_t(K) \end{aligned}$$

,which implies application of the spatial dispersal rule perserves the overall population. Since we are running finite simulation on $s \in [1, L]$ ($[a, b]$ means integers $a, a+1, ..., b$ if not otherwise specified), special treatment is required at border, namely $M(s-1)_{|s=0} = M(s)_{|s=0}$ and $M(s+1)_{|s=L} = M(s)_{|s=L}$ where $s-1, s+1$ are undefined. It's easy to verify eq(3) stays true for this finite $\{s\}$ under such treatment,by constructing an infinite array of $[1, L]$ and exploiting its symmetry.

2.3.2 Introducing tools for visualising

Given that spatial dispersal is merely a re-distribution of existing population, we reason it is merely adding a twist to the constitutive geometric growth process, as described by eq(1). The existence of dispersal ensures

$$N_{t+1}(s) \neq \lambda(s) \cdot N_t(s)$$

, which will be true if geometric growth is the only process. Nevertheless, we can still exploit this formalism to describe the growth process, by defining

$$N_{t+1}(s) = \mu_t(s) \cdot N_t(s) \mu_t(s) = \frac{N_{t+1}(s)}{N_t(s)}$$

However, $\mu_t(s)$ is undefined if $N_t(s) = 0$. To resolve this issue,we transform the $N(s)$ with a modified version of lograthmic function eq(4), use of which also allows comparison between population across different scales.

(4)
$$f(x) = \log_{10}(1 + 10 \cdot x)$$

and the logarithm of $\mu_t(s)$ is estimated with $\nu_t(s)$

(5)
$$\begin{aligned} log_{10}(\mu_t(s)) &= \log_{10}(\frac{N_{t+1}(s)}{N_t(s)}) \\ &= \log_{10}(N_{t+1}(s)) - \log_{10}(N_t(s)) \\ &= \log_{10}(10 \cdot N_{t+1}(s)) - \log_{10}(10 \cdot N_t(s)) \\ &\sim \log_{10}(1 + 10 \cdot N_{t+1}(s)) - \log_{10}(1 + 10 \cdot N_t(s)) \\ \nu_t(s) &= f(N_{t+1}(s)) - f(N_t(s)) \end{aligned}$$

To illustrate, we setup a simulation with initial condition and graph $f(N_t(s))$, $\nu_t(s)$, $f(\sum_s(N_t(s))$ accordingly for $t \in [1, 50]$:

(6)
$$\begin{cases} N_1(s) = 5, s \in [1, 20] \\ d = 0.1 \\ \lambda(s) = \begin{cases} 0.9 & s \in [1, 10] \\ 1.1 & s \in [11, 20] \end{cases} \end{cases}$$

The obseved linear trends also justify our use of $\nu(s, t)$ eq(5), which essantially results from the application of abackward difference operator along the time axis, in other words:

$$\nu_t(s) = D_t(N_t(s))$$

where D_t is the backward difference operator so that

$$D_t(f_t) = f_t - f_{t-1}$$

2.3.2.1 The steady global growth rate λ_{end}

The linear tail at $t \to \infty$ indicates the total population settles into a steady growth state, namely $\lambda_{global}(t) = \frac{E_s(N_t(s))}{E_s(N_{t-1}(s))} \to 1.098$ and $10^{E_s(\nu_t(s))} \to 1.098$ as $t \to \infty$ (see figure 4). Define:

$$\lambda_{end} = \lim_{t \to \infty} \lambda_{global}(t)$$

Given our finite simulation size, we can only approximate this limit by extending the simulation time. Throughout the analysis, we record $\lambda_{end} = E_t[\lambda_{global}(t)]$ from 20 consecutive $\lambda_{global}(t)$ if $\sigma_t[\log_{10}(\lambda_{global}(t))] < 10^{-5}$.

Definition of λ_{end} is also justified by the observation that $\sigma_s(\nu(s)) \to 0$ as $t \to \infty$, that is, $\nu(s)$ freezes into a constant independent of the position: $\nu(s) = \nu_{end}$. Because $\nu_t(s)$ approximates $\mu_t(s)$, we assume $\mu_t(s)$ is constant, aka:

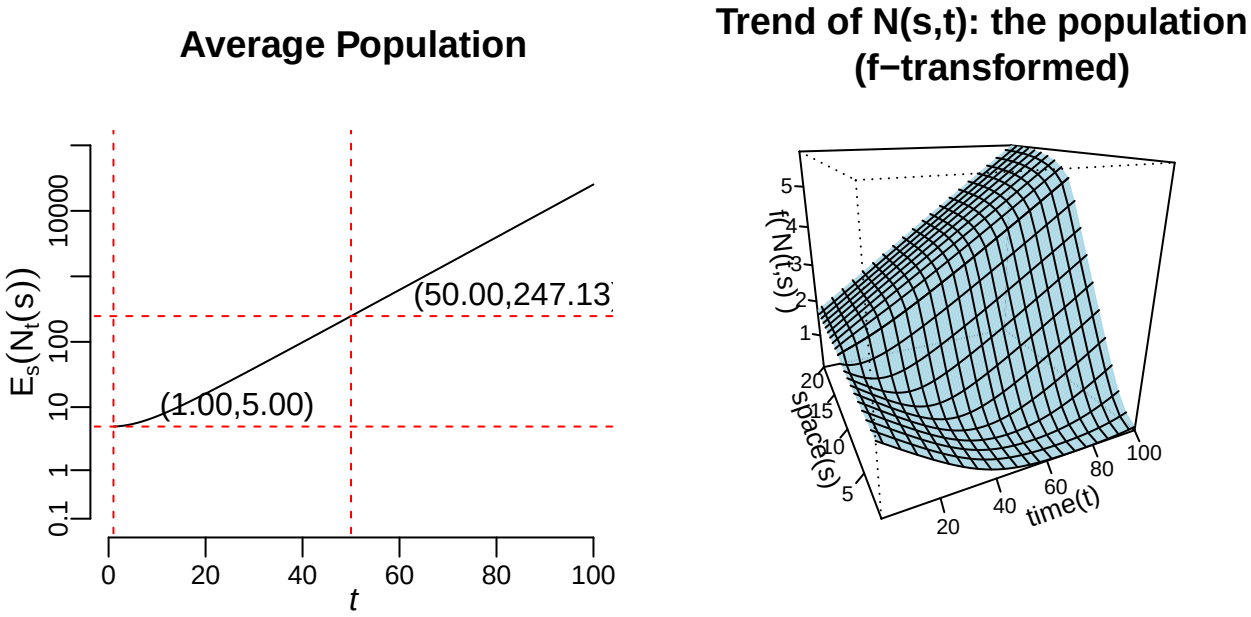


Figure 3: Simple visualisation without taking differential

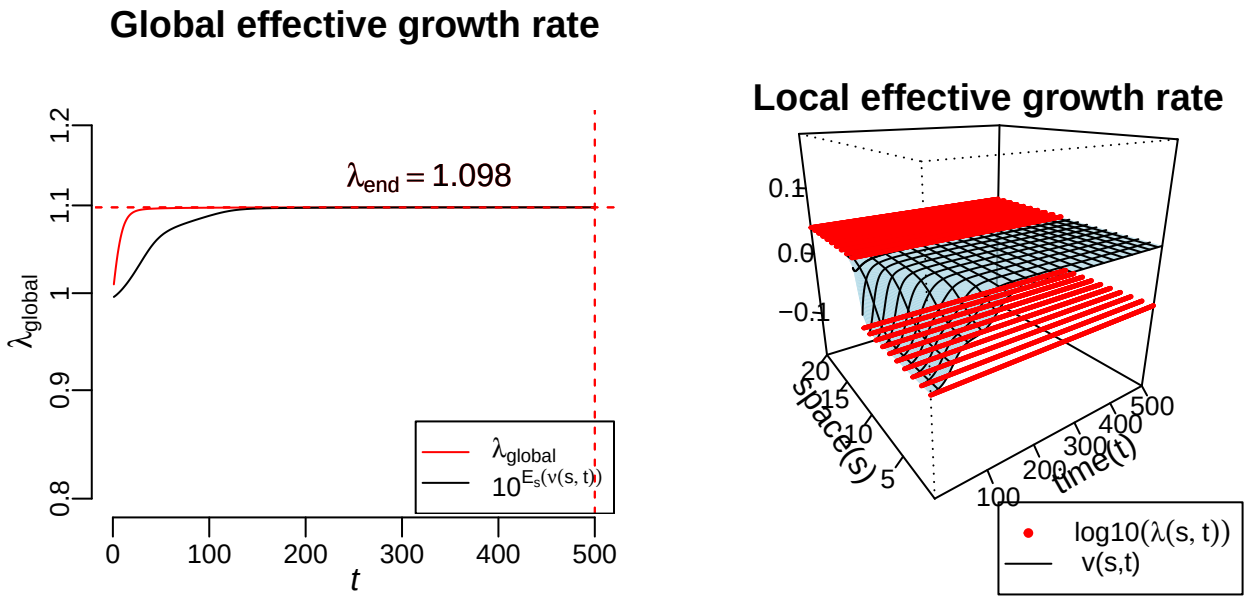


Figure 4: Visualisation after taking differential w.r.t. time

$$\nu_t(s) = \nu_{end}, but \quad \nu_t(s) \sim \mu_t(s)$$
$$\mu_t(s) = \mu_{end}$$

Then by definition of $\mu_t(s)$, we can deduce that global population also grows at a constant rate:

$$\begin{aligned}
\mu_{end} &= \log_{10}(\frac{N_{t+1}(s)}{N_t(s)}) \\
10^{\mu_{end}} &= \frac{N_{t+1}(s)}{N_t(s)} \\
E_s\left[N_{t+1}(s)\right] &= E_s\left[10^{\mu_{end}} \cdot N_t(s)\right] \\
E_s\left[N_{t+1}(s)\right] &= 10^{\mu_{end}} E_s[N_t(s)] \\
\log_{10}\left(\frac{E_s[N_{t+1}(s)]}{E_s[N_t(s)]}\right) &= \mu_{end} \\
\lambda_{end} &= \mu_{end}
\end{aligned}$$

2.3.3 Factors that affect λ_{end}

Note: We make the empirical assumption that λ_{end} is independent of $N_1(s)$ (see appendix figure 8)

We then ask what factors determine λ_{end}. We know it is not a simple average of $\lambda(s)$ since $E_s(\lambda(s)) = 1.0 \neq 1.098 = \lambda_{end}$ from eq(6).

2.3.3.1 Limit case: No spatial dispersal

In the limit case without dispersal ($d = 0.0$), we anticipate the global growth to be dominated by the patch with the greatest $\lambda(s)$, whose population would eventually take over given a sufficiently long period. Formally:

(7)
$$\lambda_{end} = max_s(\lambda(s)), when\ d = 0.00$$

Although this no longer holds for $d > 0$, we can still anticipate a proportionality that $\lambda_{end} = A \cdot max_s(\lambda(s))$, where factor A is determined from spatial configuration $\lambda(s)$. In other words, the equivalent max_s operator becomes more complex because of effect of spatial configuration on cluster growth.

To understand this effect, we keep the magitude of $\lambda(s)$ unchanged, but shuffle it spatially instead. We observe variation in λ_{end} under such shuffling and attempted some modelling (See 2.3.4). We give a brief analysis here before attempting the actual statistical test.

2.3.3.2 Effect of the reflective edge

Consider an edged cluster of good patches $s_{good} \in [1, 8]$ of length 8 on a strip $\{s\} = [1, 20]$ of length 20, where $\lambda(s) = \lambda_{good} = 1.1$. If we reflect the whole strip onto $\{s\} = [-19, 0]$, so that $\lambda(a - 0.5) = \lambda(-a - 0.5)$ and $N_t(a - 0.5) = N_t(-a - 0.5)$, then the spatial dispersal eq(2) is apparently restored in $s \in [0, 1]$, because $M_t(0) = \lambda_t(0) \cdot N_t(0) = \lambda_t(1) \cdot N_t(1) = M_t(1)$

We run a simulation to confirm edge cluster of length 8 on a strip of length 20 give a same λ_{end} as a central cluster of length 16 on a strip of length 40. (See figure 5). Given the identical $\lambda_{global}(t)$, we conclude an edged cluster is equivalent to a non-edged(central) cluster of a doubled length.

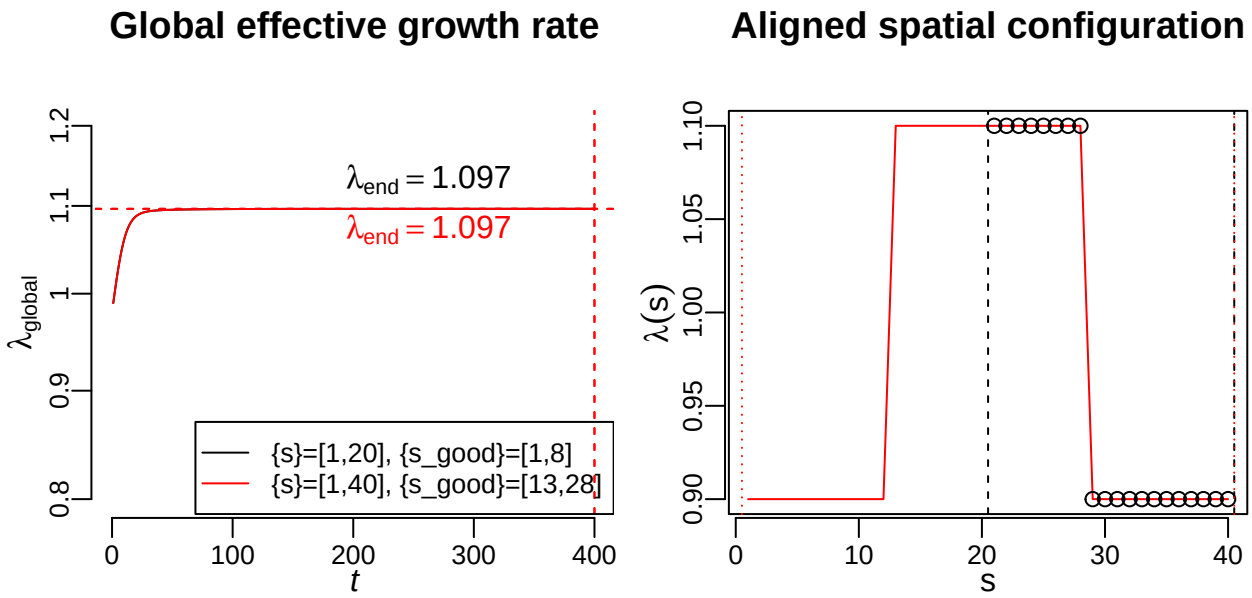


Figure 5: Left: λ_{end} stays invariant under reflection operation;Right: dotted-line indicates edge of {s}

2.3.3.3 Collective growth of closely separated clusters

If we do the same reflective operation with a good cluster separated by k bad patches from the edge, then we shall obtain central clusters separated by $K = 2k$ bad patches, which is a subcase of $K \in [0, \infty]$. We then measure λ_{end} for $K \in [0, 10]$, for equally sized unedged clusters on a strip of length 100, with total number of good patches ($L_{good} = |\{s_{good}\}|$) that are power of 2. Use of central clusters avoids confounding with the edge reflection issue. We use $\lambda_{good} = 1.2$ for better resolution.

Results (figure 6 left) show that at small separations, growth of two small clusters enhance each other and can be collectively treated as a cluster of larger size. But this enhancement decays very quickly as spearation increases, seemingly at an exponential rate. At large separations, there is so little enhancement that λ_{end} is same as that of a single cluster. By analogy to eq(7), we assert λ_{end} is dominated by the largest cluster for sufficiently large K.

To further understand the effect of enhancement at small separations, we simulate $K = 1$ cases for unequally-sized clusters. Two clusters are placed in the center of a strip of length 400. Combinations of cluster sizes (L_{left}, L_{right}) of $(0, 0) \sim (8, 8)$ are considered (figure 6 right). To our surprise, big cluster receives impairment rather than enhancement from a smaller cluster. In other words, a small cluster is more capable of slowing down the larger cluster than mere bad patches, counter-intuitively. In summary, collective growth at $K = 1$ is more like an average of two clusters. Because of its complexity, this effect is excluded from 2.3.4

2.3.4 Predict λ_{end} from maximal cluster size

Because the λ_{end} from two equally-sized clusters is well approximated with that from a single cluster for $K > 1$, we reason the biggest cluster ($max(cluster\_size)$) should dominate λ_{end}. To test this hypothesis, we sampled random spatial configurations at $L = 80$, $L_{good} = 30$, $\lambda_{good} = 1.2$, $\lambda_{bad} = 0.8$, $d = 0.1$.This is done by taking a random permutation of length 80 and rearrange the patches accordingly, so that each permutation is equally likely to be sampled. From (figure 7), $max(cluster\_size)$ seems to predict the average λ_{end} accurately, although the exact value varies for each configuration. This is confirmed with an ANCOVA analysis ($P < 2.2 \times 10^{-16}$). On a sidenote, $max(cluster\_size)$ itself follows a non-uniform distribution, with 3,4,5 being the most popular values. We conclude that maximal cluster is a good predictor of λ_{end}, and that λ_{end} is indeed dominated by the biggest cluster, though subject to correction from an account of interaction between closely separated clusters (see 2.3.3.3)

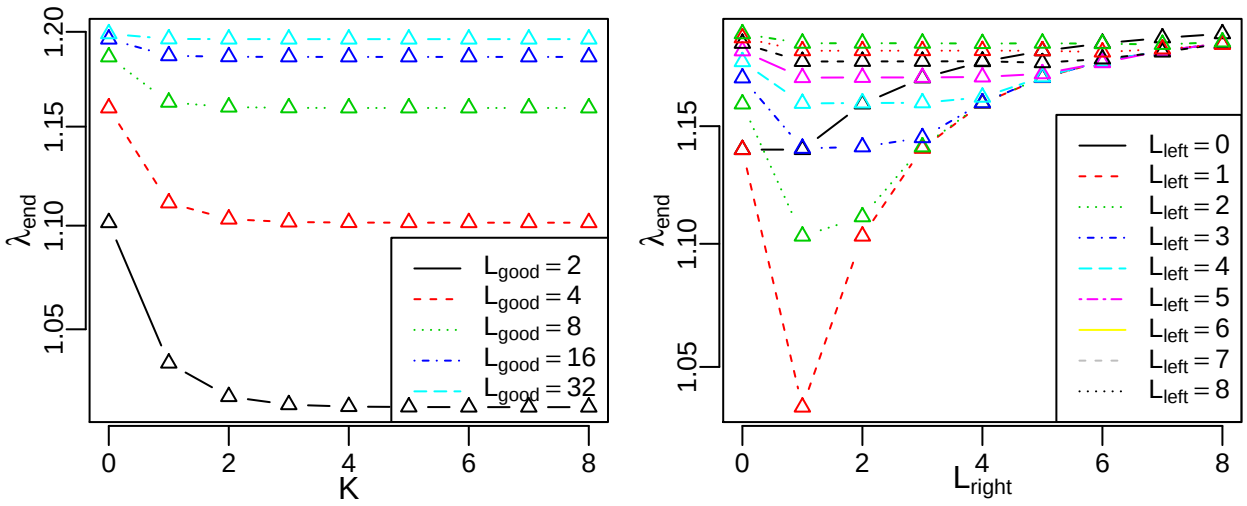


Figure 6: Left: Equally sized clusters at different separation. Right: Unequally sized clusters at $K = 1$

```
## Loading required package: grid
```

```
## Analysis of Variance Table
##
## Response: lambda_end
##                            Df  Sum Sq Mean Sq F value    Pr(>F)
## as.numeric(max_cluster)     1 1.89003 1.89003   41707 < 2.2e-16 ***
## Residuals                9998 0.45308 0.00005
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.3.5 Conclusion

We have investigated the growth-dispersal model using final-steady-growth-rate λ_{end} as a proxy, with the simple case where $\lambda(s)$ can only take two values, λ_{good} or λ_{bad}. Under such condition, we observe that λ_{end} is dominated by the biggest cluster of good patches. More generally, separation $K = 0$ and $K = 1$ are the most interesting cases, where cluster interaction produces the complex outputs. Thus it may be possible to model the λ_{end} by convolving $\lambda(s)$ with a local filter and then taking the maximum of the resultant function. The exact form, however, is yet to be defined.

3 References:

[1] Ellner SP, Guckenheimer J. An introduction to R for dynamic models in biology Department of Mathematics. 2011.

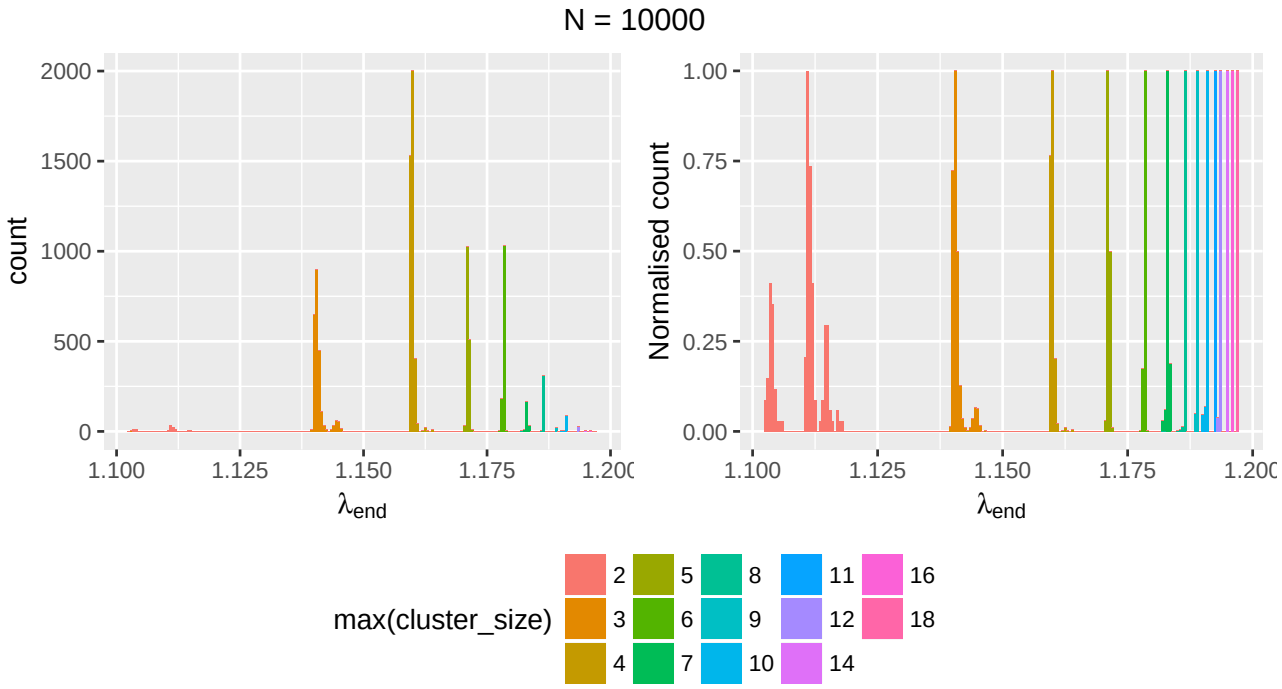


Figure 7: λ_{end} for randomly sampled spatial configurations. Left: Plotted with raw count; Right: Count is normalised for each $max(cluster\_size)$ so that highest bar measures 1.0

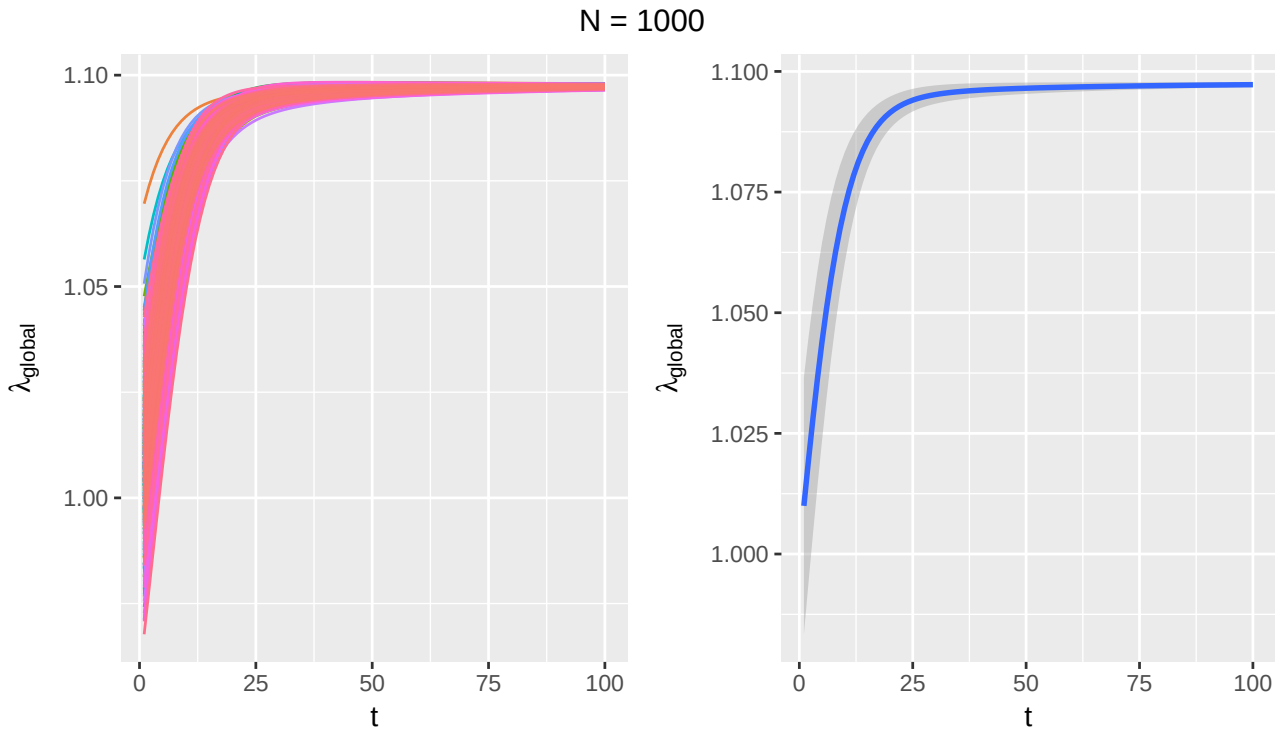


Figure 8: $\lambda_{global}(t)$ for $N_1(s) \sim unif(0, 100)$

4 Appendix:

4.1 Figures

4.2 R-scripts used:

4.2.1 "index.R": the main script that generates the document

```
## ----word1, echo=F, inclue = T-----------------------------------------

#### Define utility function
'%f%' <- function(x,f) { do.call(what = f,
                                 args = list(x)) }

#### Read data from file
f = file('rpc2017/a1/usr-share-dict-words','r')
words  <-  readLines(f)
close(f)

#### Convert to uppercase and count unique number
words <- toupper(words)
words <- unique(words)
sprintf('A: %d', length(words))%f%cat;

## ----word2, echo=F-----------------------------------------------------
buf <- c()
match_idx <- grep('\'',words)
buf <- c(buf,sprintf('A: %d ', length(match_idx)))

######### Print hits
hlen = 10
df = data.frame( words[match_idx])
caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = caption
)
words <- words[-match_idx]
buf <- c(buf,sprintf('\n%d words left after filtering', length(words)))
buf%f%cat

## ----word3, echo=F-----------------------------------------------------
buf <- c()
match_idx <- grep('[^ -~]',words)
buf <- c(buf,sprintf('A: %d ', length(match_idx)))

######### Print hits
hlen = 10
df = data.frame( words[match_idx])
caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
```

```
  booktabs = TRUE,
  caption = caption
)



words <- words[-match_idx]
words_data <- words
buf <- c(buf,sprintf('\n%d words left after filtering, saved to "words_data"', length(words)))
buf%f%cat

## ----word4, echo=F-------------------------------------------------------
words_data <- sort(words_data)
putative <- words_data[
  endsWith(words_data, 'OG')
  ]
target <- vapply( putative
                  , FUN = {function (x) paste0(x,'UE')}
                  ,FUN.VALUE='char'
                  ,USE.NAMES=T)
XOGUE <- words_data[match(target,words_data)]
XOG   <- putative
result <- data.frame(XOG,
                     XOGUE,
                     row.names = NULL)
colnames(result)<- c("XOG","XOGUE")
result <- result[complete.cases(result),]
rownames(result) <- 1:nrow(result)
sprintf('%d pairs of words were found to conform "XOG"->"XOGUE" projection', nrow(result))%f%cat
# print(result,  row.names = T)

######### Print hits
hlen = 10
df = result
caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = caption,
  align = 'l'
)



## ----word5,echo=F--------------------------------------------------------
fname = 'rpc2017/a1/scrabble.txt'
f = file(fname,'r')
buf = f%f%readLines
buf = strsplit(buf,' ')
f%f%close
buf = t(array(unlist(buf),dim = c(length(buf[[1]]),length(buf) )))
buf <- buf[order(buf[,1]),]
```

```
l = length(unique(buf[,1]))
if (l != 26){
  msg = paste0('[WARNING]: scrabble scores incomplete, expected 26, actual ',l)
  warning(msg)
}
# data.frame( char = buf[,1], score = buf[,4]) %f%print
scores <- buf[,ncol(buf)] %f% as.integer
names(scores) <- buf[,1]
hlen = 10
df = data.frame(scores = sort(scores,decreasing = T))

caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = caption,
  align = 'l'

)


## ----word6,echo = F-------------------------------------------------------
x = words_data
scoreit = {function (x){
                      sum(
                          unlist(
                            lapply( strsplit(x,''),
                            FUN = {function(i) scores[i]}
                            )
                          )
                      )
                    }
          }
y <- sapply( x,
                     FUN = scoreit,
                    # FUN.VALUE='integer',
                    USE.NAMES=T)
dy <- density( y, bw = 0.6)
# par(mar=c(3,3,3,3))
plot( dy,
      main='Distribution of scrabble scores')
my = mean(y)
abline( v = my, col=2 )
text(my, median(dy$y),sprintf('mean=%.3f',my), col = 2)

#### DEBUG stats
if (debug){dy%f%print}

cat(
  "Highest score is achieved by:",
  names(y)[idx<-which.max(y)],
  ", Score:",
  y[idx]
```

```
)

## ----word7----------------------------------------------------------------
######################################################################
#### compile reverse complement and define functions ##############
######################################################################
dict = rev(LETTERS)
names(dict) <- LETTERS

rev_comp = function(x){
  paste0(
    rev(dict[unlist(strsplit(x,''))])
  ,collapse = "")
}


######################################################################
#### Carry out search
######################################################################

idx = match(
  sapply(words_data, FUN=rev_comp),
  words_data
)%f%na.omit

######################################################################
#### linking  WORDS to their reverse complements #################
#### Sort and format
######################################################################
rv_0 = words_data[idx]
rv_1 = sapply( rv_0, FUN = rev_comp )

res <- data.frame(
  rv_0,
  rv_1)
res <-  apply(res, MARGIN = 1, FUN = sort)%f%t
dup_idx <- duplicated(res[,1])
res <- res[!dup_idx,]
res <- res[ order(nchar(res[,1]), decreasing = T),]

colnames(res) = c('smaller_word','larger_word')
rownames(res) <- 1:nrow(res)
sprintf('We found %d entries, out of which %d are reverse-complement to themselves , Showing longest 10

######### Print hits
hlen = 10
df = res
caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = caption,
  align = 'l'
```

```
)

## ----word8-----------------------------------------------------------------
count = function(lst){
  sapply(lst, {function(x){
    if (x[1]==-1){
      0
    }else{
      length(x)
    }
    }} )
}


##### Define generic query function
search_perm <- function (query_orig_str = 'LLFAUYPNI',
                         min_len = 4,
                        database =  words_data,
                        special  = c('A')
                        ){

  #### Preprocess the given string (vectorizing etc.)
  query_orig_str <- query_orig_str
  query_orig = strsplit(query_orig_str,'')[[1]]

  #### Obtain the unqiue set of chars
  query_uni = query_orig%f%unique

  #### Count occurrence of chars in the query
  qcount = sapply( query_uni,
          {function(subq)
            {count(
              gregexpr(subq, query_orig_str)
              )}})

  #### Counting occurence of chars for each word in database
  raw = sapply( query_uni,
          {function(subq)
            {count(
              gregexpr(subq, database)
              )}})

  if(debug){
    raw0 <- raw
    raw <- raw0
  }

  ### add index to initial dataset
  rownames(raw) = 1:nrow(raw)


  #### Exclude results that use more letters than specified
  exidx = apply(
```

```
    sweep(raw,qcount,MARGIN=2,FUN={function(x,y) x>y}),
    MARGIN=1,
    FUN = any )
  raw <- raw[!exidx,]

  #### Exclude result that does not contain special chars
  exidx <- apply(
    sweep(raw==0, query_uni %in% special,
          MARGIN = 2,
          FUN = '&'),
        MARGIN = 1,
        FUN = any)
  raw <- raw[!exidx,]

  ###### Compile into a data.frame
  words_res = database[rownames(raw)%f%as.integer]
  words_score = apply( raw,MARGIN = 1, sum)

  output = data.frame(
    words_res,
    words_len = nchar(words_res),
    words_score
    )

  ######################################################################
  #### Filtering out words that require additional characters ###############

  #### Option 1: Do not allow extra chars
  inidx <-  output$words_len==output$words_score & output$words_score>=min_len

  #### Option 2: Do allow extra chars
  # inidx <-  output$words_score>=min_len

  output <- output[inidx,]

  odx = order(output$words_score,decreasing = T)
  output <- output[odx,]
}


### Actual evaluating the demanded query
query_orig_str = 'LLFAUYPNI'
min_len = 4
database =  words_data
special = c('A')

output <- search_perm(
    query_orig_str = query_orig_str,
    min_len = min_len,
    database =  database,
    special = special
)
```

```
#### Check correctness
if (nrow(output) != 39)
{
  warning("[WARNING]:Number of result is different from the cached one")
}


sprintf('We found %d hits according to query \"%s\" in \"words_data\"", \n (Option: min_len=%d, special=%
# sprintf('Showing best %d hits below:',hlen) %f% cat

######### Print hits
hlen = 20
df = output
caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = caption,
  align = 'l'
)


## ----exam1----------------------------------------------------------------
#### Read data
setwd('rpc2017/a1/grading/')


##### read correct answers into "ref"
fname = "crib.dat"
f = file( fname,'r')
ref  <-  readLines(f)
close(f)
qnum <- length(ref)
qlevel = unique(ref)%f%sort
qdict <- 1:length(qlevel)
names(qdict) <- qlevel

sprintf("[DEBUG]:There are %d questions from file:%s",qnum,fname)%f%cat

####### Loading grading scheme and transform it to a vector
grade_crit <- read.table(file = 'grade.txt',header = T)
grade_dict <- rep(0,101)
for (i in 1:nrow(grade_crit)){
  ROW <- grade_crit[i,]
  MIN = ROW[[1]]
  MAX = ROW[[2]]
  grade = ROW[[3]]
  grade_dict[ MIN:MAX + 1] <- grade
}
grade_dict <- factor(grade_dict, labels = grade_crit$grade%f%levels)
graduate <- function(x){grade_dict[x+1]}

##### Test
```

```
if ( any(!graduate(c(39,40,49,50,59,60,69,70)) == c('F','D','D','C','C','B','B','A')) ){
  warning("[WARNING]:Cannot safely define \"graduate()\"")
}

##### Read results for each student
buf = list()
resp = list()
for (fname in list.files(getwd())){
  if ( startsWith(fname,'student') ){

  ##### Extract student number from filename and read data
  idx = regmatches( fname,gregexpr("[0-9]+", fname))%f%as.integer
  f = file(fname,'r')
  ttbl <- read.table(f, header = T,stringsAsFactors = F)
  close(f)
  ##### score answers and cache answers to "resp"
  score = sum(ref[ttbl$qn] == ttbl$response)
  buf[[idx]] = list( student_idx = idx,
                     score = score,
                     qnum = nrow(ttbl)
                     )
  ttbl$response <- qdict[ttbl$response]
  resp[[idx]] = ttbl
  }
}

#### Compile data frame
df <- data.frame(matrix(unlist(buf), nrow=length(buf), byrow=T))
colnames(df) <- names(buf[[1]])
df$percent <- floor(df$score / df$qnum * 100)
df$grade <- df$percent%f%graduate
df$rank <- rank(-df$percent)
df <- df[order(-df$score),]
if (sd(df$qnum)!=0){
  warning("Question answered is not homogeneous")
}

df <- subset(df,select = -c(qnum)) #### Drop columns for better printout
hlen <- nrow(df)
caption = sprintf('Result of %d students in this exam, sorted by percent score in descending order', nro
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = caption,
  row.names = F
  # align = 'l'
)
sprintf("Stats were calculated for the rounded percent score for %d students in the exam", nrow(df))%f%c
summary(df$percent)%f%print
res_exam <- df


## ----exam_naive,fig.cap=" \\label{fig:exam_naive} Distribution of $I_{cheat}$"----
```

```
idx_xy <- t(combn(1:nrow(res_exam),2))
res_exam <- res_exam[order(res_exam$student_idx),]

# df <- res_exam
cindex <- function(sa,sb,df = resp){
  sumA <- intersect(df[[sa]]$qn,df[[sb]]$qn) %f% length
  sumB <- intersect(data.frame(t(df[[sa]]))
             ,data.frame(t(df[[sb]]))
               )%f% ncol
  c(sumA,sumB)
  }

overlap <- apply(idx_xy, MARGIN = 1, FUN = {function(x)
  cindex( x[1],x[2])
}
)%f%t
I_cheat <- overlap[,2]/overlap[,1]
hist(I_cheat,main = NULL);


## ----exam_diag, fig.cap= cap-----------------------------------------------

cap <- "\\label{fig:exam_diag} Diagnostic plots for finding the outlier/cheater"

##### Calculate naive cheat index
idx_xy <- t(combn(1:nrow(res_exam),2))
res_exam <- res_exam[order(res_exam$student_idx),]

cindex <- function(sa,sb,df = resp){
  sumA <- intersect(df[[sa]]$qn,df[[sb]]$qn) %f% length
  sumB <- intersect(data.frame(t(df[[sa]]))
             ,data.frame(t(df[[sb]]))
               )%f% ncol
  c(sumA,sumB)
  }


#### take two students and find overlap in choic of questions
overlap <- apply(idx_xy, MARGIN = 1, FUN = {function(x)
  cindex( x[1],x[2])
}
)%f%t
question_overlap <- overlap[,1]
answer_overlap <- overlap[,2]
x <- question_overlap
y <- answer_overlap

#### regression
mdl <- lm(y~x)
mdl$stdres <- rstandard(mdl)

###### Thresholding outliers
conf <-  0.975 #### 1-tail p-value
```

```
cap <- paste0(cap,
sprintf("\\newline (Using %.3f %% confidence (2 tailed) as threshold)", (2 * conf - 1 )*100)
              )
zlim <- qnorm(conf)
outliers <- which( abs(mdl$stdres) > zlim)

##### Print plots
par(mfrow=c(1,2))
plot(x,y,
     xlab = "Number of overlapped questions",
     ylab = "Number of overlapped answers",
     xlim = c(-1,35),
     ylim = c(-1,35)
    )

text( x[outliers], y[outliers], outliers,adj = c(-.3,-.3))
abline(mdl,col = 2)

plot(mdl$stdres,ylab='standardised residual',
     ylim = c(-6,6))
abline(h = zlim, col=2)
abline(h =-zlim, col=2)
abline(h = 0,lty=2)
text( outliers, mdl$stdres[outliers], outliers,adj = c(-.3,-.3))

#### compile data frame
cheats<-data.frame(
  stud_A = idx_xy[,1],
  grad   = df$grade[idx_xy[,1]],
  stud_B = idx_xy[,2],
  grad   = df$grade[idx_xy[,2]],
  ques_same = question_overlap,
  ans_same = answer_overlap,
  zscore = mdl$stdres,
  Confidence_to_reject_NULL = 2 * pnorm(abs(mdl$stdres)) - 1
)

cheats <- cheats[order(-(cheats$zscore)),]

######### Print hits
hlen = 10
df <- cheats
df <- subset(df,select = -c(zscore)) #### Drop columns for better printout
caption = sprintf('Showing best %d hits below:',hlen)
knitr::kable(
  head(df,hlen),
  booktabs = TRUE,
  caption = paste0('\\label{tab:exam_diag}',caption)
)



## ----DMB_log, fig.cap= cap,eval=T----------------------------------------
```

```r
source('DMB.R')
################################################
### simple visulisation of N(s,t) ###############
################################################
cap <- " \\label{fig:DMB_log} Simple visualisation without taking differential "
L=20
pop = rep(5,L)
# pop <- rnorm(L,mean = 1, sd = 4 )
d<- 0.1
rate <- rep( c(0.9,1.1),c( L/2, L/2)) # %f% shuffle
# rate <- rep( c(0.9,1.1),c( L/2, L/2)) %f% shuffle

s0 <- list(    pop = pmax(0,pop),
               rate = pmax(0,rate),
               filter= c(d, 1-2*d, d),
               nhis = 500
               )
s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
s0$L <- length(s0$pop)
s1<-s0
s1 <- kickoff(s1,100)
# f <- function(x){log10(1+10*x)}
simple_vis(s1)

## ----DMB_logdiff,fig.cap = cap,eval = T----------------------------------
#####################################################################
### visulisation of N(s,t) after taking differential w.r.t. time
#####################################################################
cap <- " \\label{fig:DMB_logdiff} Visualisation after taking differential w.r.t. time"
s1<-s0
s1 <- kickoff(s1,500)
diff_vis(s1)


## ----reflect_sep,fig.cap=cap---------------------------------------------
cap <- " \\label{fig:reflect_sep} Left: $\\lambda_{end}$ stays invariant under reflection operation; \\ne

source('DMB.R')
init_row21()


L=20
pop = rep(5,L)
d<- 0.1
bad = 0.9
good = 1.1
rate <- rep(bad,L)
s0 <- list(    pop = pmax(0,pop),
               rate = pmax(0,rate),
               filter= c(d, 1-2*d, d),
               nhis = 500
               )
s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
```

```
s0$L <- length(s0$pop)

rate[1:8] <- good
rate_ref<-rate
s0$rate<-rate
s1 <- kickoff(s0,400)
diff_vis_init(s1)
par(xpd = F)
diff_add(s1,adj = c(1.5,-.5))


L=40
pop = rep(5,L)
d<- 0.1
bad = 0.9
good = 1.1
rate <- rep(bad,L)
s0 <- list(    pop = pmax(0,pop),
               rate = pmax(0,rate),
               filter= c(d, 1-2*d, d),
               nhis = 400
               )
s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
s0$L <- length(s0$pop)
rate[1:16 + 12] <- good
s0$rate<-rate
s1 <- kickoff(s0,400)
diff_add(s1,adj = c(1.5, 1.2),col = 2)

legend(
  # x = 245, y = -.11
          ,cex = 0.8
    ,x="bottomright"
    # ,x="bottomleft"
          ,legend=c(
            # sprintf('{%s}'),expression(s))
             # substitute(t[0]==t0)
            # parse(text ="'{s}'=='[0,20]''{s_good}'=='[0,8]'" )
            '{s}=[1,20], {s_good}=[1,8]'
            ,'{s}=[1,40], {s_good}=[13,28]'
            # expression(s="["0,10"]")
            # ,expression(10^{E[s](nu(s,t))})
                  )
          ,col=c(1,2), lwd=1, lty=c(1,1),
          pch=c(NA,NA) )
plot(s0$rate,type = 'l',col=2
     ,xlab = 's'
     ,ylab = expression(lambda(s))
     ,main = 'Aligned spatial configuration')
points(c(rep(0,20),rate_ref),col = 1)
abline(v = c(20.5,40.5),col = 1,lty=2)
abline(v = c(0.5,40.5),col = 2,lty=3)
```

```
## ----data__lend_K,eval = F-------------------------------------------------
## ######################################################################
## ### Compiling data for equally-sized clusters at different separations
## ######################################################################
## Ks = 0:8
## bad = 0.8
## good = 1.2
##
##
##
##
## ##### Initialise grid
## L=200
## pop = rep(5,L)
## d<- 0.1
## rate <- rep(bad,L)
## s0 <- list(    pop = pmax(0,pop),
##                rate = pmax(0,rate),
##                filter= c(d, 1-2*d, d),
##                nhis = 20
##                )
## s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
## s0$L <- length(s0$pop)
##
##
## L_goods = c(2,4,8,16,32)
##             # ,32)
##
## ################################################
## ### Initialise OUTPUT matrix and collect data ###
## ################################################
## OUTPUT_M = matrix(0,length(L_goods), length(Ks),
##                   dimnames = list(L_goods,Ks))
##
## for (L_i in 1:length(L_goods)){
##   L_good = L_goods[L_i]
##   n_good = L_good/2
## ?matrix
## OUTPUT = matrix(0,length(Ks),s0$nhis-1,
##                 dimnames = list(Ks,NULL))
##   for (i in 1:length(Ks)){
##     K = Ks[i]
##     offset = floor(K/2)
##     rate[1:L] = bad
##     rate[ c((L/2-offset-n_good+1):(L/2-offset),
##             (L/2-offset+K+1):(L/2-offset+K+n_good))] <- good
##     s0$rate<-rate
##     s1 <- s0
##     max_iter = 7
##     num_steps = 200
##     for (num in 1:max_iter){
##     s1 <- kickoff(s1,num_steps)
##     zs = diff(log10(s1$psum))
```

```
##     if (sd(zs)<1E-8){
##       break
##     }else if(num==max_iter){
##       warning(
##         sprintf("[WARNING]: lambda_global does not converge after %d steps",num*num_steps)
##       )
##     }
##
##     }
##     OUTPUT[i,] = zs
##   }
##
##   MEAN = apply(OUTPUT,MARGIN=1,mean)
##   OUTPUT_M[L_i,] = MEAN
## }
## ### save output
## save( L_goods,Ks,OUTPUT_M, file = "lend_K_dp1.dat")

## ----data__lend_K1,eval = F-----------------------------------------------
## ########################################################################
## ### Compiling data for unequally-sized clusters at K = 1 ##########
## ########################################################################
## # Ks = 0:8
## bad = 0.8
## good = 1.2
##
## ##### Initialise grid
## L=400
## pop = rep(5,L)
## d<- 0.1
## rate <- rep(bad,L)
## s0 <- list(    pop = pmax(0,pop),
##                rate = pmax(0,rate),
##                filter= c(d, 1-2*d, d),
##                nhis = 20
##                )
## s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
## s0$L <- length(s0$pop)
##
## # n_goods = c(2,4,8,16,32)
## n_goods = 0:12
##             # ,32)
##
##
## OUTPUT_M = matrix(0,length(n_goods), length(n_goods),
##                   dimnames = list(n_goods,n_goods))
##
## for (n_i in 1:length(n_goods)){
##   n_goodi = n_goods[n_i]
##   OUTPUT = matrix(0,length(n_goods),s0$nhis-1,
##                   dimnames = list(n_goods,NULL))
##   for (n_j in 1:length(n_goods)){
##     n_goodj = n_goods[n_j]
```

```
##     rate[ 1:L ] = bad
##     rate[c((L/2-n_goodi):(L/2-1),
##            (L/2+1):(L/2+n_goodj)
##            )] <- good
##     s0$rate<-rate
##     s1 <- s0
##     max_iter = 5
##     num_steps = 200
##     for (num in 1:max_iter){
##       s1 <- kickoff(s1,num_steps)
##       zs = diff(log10(s1$psum))
##       if (sd(zs)<1E-5){
##         break
##       }else if(num==max_iter){
##         warning(
##         sprintf("[WARNING]: lambda_global does not converge after %d steps",num*num_steps)
##       )
##     }
##     }
##     # zs
##     OUTPUT[n_j,] = zs
##   }
##   MEAN = apply(OUTPUT,MARGIN=1,mean)
##   OUTPUT_M[n_i,] = MEAN
## }
##
## ### save output
## save( OUTPUT_M, file = "lend_K1_dp1.dat")

## ----vis__lend_K_dp1, fig.cap = cap-------------------------------------
######################################################################
### Plotting data for clusters at separations   ####################
######################################################################
cap <- "\\label{fig:vis__K} Left: Equally sized clusters at different separation. Right: Unequally sized
init_row21()

load( file = "lend_K_dp1.dat")
p1 <- matplot(x = colnames(OUTPUT_M), t(10^OUTPUT_M),log='y',type= 'b',pch=2,
              xlab = expression(K),
              ylab = expression(lambda[end])
              ,cex = 0.8)
legend(x = 'bottomright',cex = 0.8
       , legend = sapply( rownames(OUTPUT_M),function(x){
            as.expression(
            bquote(L[good] ==.(x))
            )}
          )
       # , legend =  paste0(expression(L[good]),"=",rownames(OUTPUT_M))
       , col = 1:dim(OUTPUT_M)[2]
       , lty = 1:dim(OUTPUT_M)[2]
       , lwd = 1)

load( file = "lend_K1_dp1.dat")
```

```
OUTPUT_M <- OUTPUT_M[1:9,1:9]
p2 <- matplot(x = rownames(OUTPUT_M), 10^OUTPUT_M,log='y',type= 'b',pch=2,
              xlab = expression(L[right]),
              ylab = expression(lambda[end])
              ,cex = 0.8)
legend(x = 'bottomright',cex = 0.8
       , legend = sapply( rownames(OUTPUT_M),function(x){
           as.expression(
            bquote(L[left] ==.(x))
           )}
         )
        , col = 1:dim(OUTPUT_M)[1]
       , lty = 1:dim(OUTPUT_M)[1]
       , lwd = 1)

## ----data__max_cluster,eval = F----------------------------------------
## max_cluster <- function(rate, good = NULL){
##   if (is.null(good)){
##   good = max(rate)
##   }
##   N = length(rate)
##   max_vct = vector(mode='integer',length=N)
##   rbools = rate==good
##   edge = 1
##   MAX  = integer(1)
##   for (i in 1:N){
##     rbool = rbools[i]
##     MAX = (MAX * rbool) + rbool
##     if (!rbool){
##       edge = 0
##       }else if(i==N){
##       edge = 1
##     }
##   max_vct[i] = MAX * (edge + 1)
##   }
##   max(max_vct)
## }
##
##
## ##### Gathering Data
## for (L_good in c(30,35)){
##
## set.seed(0)
## L=80
## # L_good = 30
## pop = rep(5,L)
## d<- 0.1
## good = 1.2
## rate <- rep(bad,L)
## s0 <- list(    pop = pmax(0,pop),
##                rate = pmax(0,rate),
##                filter= c(d, 1-2*d, d),
##                nhis = 20
```

```
##                 )
## s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
## s0$L <- length(s0$pop)
## rate[1:L_good] <- good
## N = 10000
## OUTPUT = list()
## OUTPUT$lend = matrix(0,N,s0$nhis-1)
## for (i in 1:N){
##     # rate <- shuffle(rate0)
##     s0$rate <- shuffle(rate)
##     s1 <- s0
##     max_iter = 5
##     num_steps = 200
##     for (num in 1:max_iter){
##       s1 <- kickoff(s1,num_steps)
##       zs = diff(log10(s1$psum))
##       if (sd(zs)<1E-5){
##         break
##       }else if(num==max_iter){
##         warning(
##           sprintf("[WARNING]: lambda_global does not converge after %d steps \n",num*num_steps)
##         )
##       }
##     }
##   OUTPUT$lend[i,] = zs
##   OUTPUT$max_cluster[i] <- max_cluster(s0$rate)
## if (i %% 100==0){
##   sprintf("[PROGRESS]:%d of %d \n",i,N)%f%cat
## }
## }
## fname = sprintf("L%d_Good%d_N%d.dat",L,L_good,N)
## save(OUTPUT,file = fname)
## }
##
## ''

## ----vis__max_cluster,fig.cap=cap----------------------------------------
source("plotter.R") #### Utils for ggplot2
######################################################################
### Plotting lambda_{end} for randomly sampled lambda(s)    #######
######################################################################

cap <- " \\label{fig:vis__max_cluster} $\\lambda_{end}$ for randomly sampled spatial configurations.  Le
fname = "L80_Good30_N10000.dat"
load(fname)
N = gsub(".*N(\\d+).*","\\1", fname)


##### Modify data frame
OUTPUT$lend_mean = apply(OUTPUT$lend,MARGIN=1,mean)
OUTPUT$stdev = apply(OUTPUT$lend,MARGIN=1,mean)
bw = 0.0002
OUTPUT$lend_mean <- 10^OUTPUT$lend_mean
```

```r
bw = 0.0005
OUTPUT$lend = NULL
df<- data.frame(OUTPUT)
df$max_cluster <- factor(round(df$max_cluster))


plot1 <- ggplot(df) + geom_histogram( aes(x = lend_mean, y = ..count..
                                  ,fill =  max_cluster
                                  # ,colour = max_cluster
                                  ),binwidth = bw
                                ) +
  xlab(expression(lambda[end])) + labs(fill='max(cluster_size)')
plot2 <- ggplot(df) + geom_histogram( aes(x = lend_mean, y = ..ncount..
                                  ,fill =  max_cluster
                                  # ,colour = max_cluster
                                  ),binwidth = bw
                                ) +
  xlab(expression(lambda[end])) + ylab("Normalised count")

grid_arrange_shared_legend(plot1, plot2,ncol=2, top = sprintf("N = %s",N))

## ----aov__max_cluster-------------------------------------------------------
########################################
############# ANCOVA analysis  ###########
########################################
df$lambda_end <- df$lend_mean
df.lm <- lm( lambda_end~as.numeric(max_cluster),data  = df)
res = anova( df.lm)
print(res)

## ----data__N1,eval = F------------------------------------------------------
## ####################################################################
## ### Compile data of lambda_global from randomly sampled N(s)###
## ####################################################################
##
## pop = rep(5,L)
## d<- 0.1
## bad = 0.9
## good = 1.1
## rate <- rep(bad,L)
## s0 <- list(    pop = pmax(0,pop),
##                rate = pmax(0,rate),
##                filter= c(d, 1-2*d, d),
##                nhis = 300
##                )
## s0$his <- do.call(rbind, rep(list(pop), s0$nhis))
## s0$L <- length(s0$pop)
##
## rate[1:L/2] <- good
## s0$rate<-rate
## N = 1000
## set.seed(0)
## par(xpd = F)
```

```
## s1 <- kickoff(s0,500)
## # diff_vis_init(s1)
## OUTPUT <- vector("list",N)
## for (i in 1:N){
##   s0$pop = pmax(0,runif(L,0,100))
##   s1 <- kickoff(s0,300)
##   OUTPUT[[i]] = s1
## }
## fname = sprintf('traj_%d.dat',N)
## save(OUTPUT,file = fname)
## cat(
##   sprintf("[DONE]:saved to %s",fname)
## )

## ----vis__N1,fig.cap=cap--------------------------------------------------
#####################################################
### plot lambda_global for randomly sampled N(s)###
#####################################################
cap <- " \\label{fig:vis__N1} $\\lambda_{global}(t)$ for $N_1(s)\\sim unif(0,100)$"
load(file = "traj_1000.dat")
l_global<-sapply(OUTPUT,function(x){diff(log10(x$psum[1:101]))})
mat<- l_global
N = dim(mat)[2]
ts<-matrix(rep(1:dim(mat)[1],dim(mat)[2]),dim(mat)[1],dim(mat)[2])
idx<-matrix(rep(1:dim(mat)[2],dim(mat)[1]),dim(mat)[1],dim(mat)[2], byrow =  T)
df = data.frame( l_global = as.vector(l_global),
                 t = as.vector(ts),
                 idx = as.factor(idx))
require(ggplot2)
require(gridExtra)
d = ggplot(df,aes(x=t,y=10^l_global))
p1 <- d +
  geom_line(aes(group=idx,colour=idx) , show.legend = F) +
  ylab(expression(lambda[global]))
# +
#   scale_y_log10(breaks=c(1,2,5,10,25))
p2 <- d + stat_summary(fun.data ="mean_sdl",  geom = "smooth") +
  ylab(expression(lambda[global]))

grid.arrange(p1,p2,ncol = 2,top = sprintf("N = %d", N))
```

4.2.2 "DMB.R" : helper functions for 2.3

```
#############################################################
### A modifier logarithmic function  ########################
#############################################################
f <- function(x){log10(1+10*x)}
f_i <- function(x) { (10^x - 1)/10}


########## shuffle your vector using a random permutation ###############
shuffle <- function(vct){
```

```
  sample(vct, length(vct))
}

#################################################################
### A circular shift function ###################################
#################################################################
shifter <- function(x, n = 1) {
  if (n == 0) x else c(tail(x, -n), head(x, n))
}

#########################################################################
### Apply both geometric growth and spatial dispersal ###################
#########################################################################
forward <- function ( s, method = 1){
  growed <- s$rate * s$pop;
  if (method == 1){
    ##### naive implementation #####
    tpop  <- c(
      head(growed,1),
      growed,
      tail(growed,1)
    )

    neighbored <-rbind(
      shifter(tpop,1),
      tpop,
      shifter(tpop,-1)
    )
    pop <- apply(
      sweep(neighbored, MARGIN=1, s$filter,FUN = '*')
      , MARGIN=2 , sum)
    pop <- pop[2:(length(pop)-1)]

  }else if(method==2){
    ##### FFT implementation
    pop <-  filter( c(growed,rev(growed)), s$filter , sides = 2 ,circular = 1,
                    # method='recursive'
                    method='convolution'
    )[1:s$L]
  }
  # pop <-pmax(0,pop)
  s$hist = rbind(tail(s$hist, s$nhis-1),pop)
  s$pop <- pop
  return(s)
}

#################################################################
########### wrapper to apply "forwrad" iteratively ############
#################################################################
kickoff<- function (s0,tmax=1){

  # t0 = Sys.time() #### timing
  s <- s0
```

```
  s$tmax <- tmax
  for (t in 1:tmax){
    s <- forward(s, method = 2 );
  }
  # t1 = Sys.time() #### timing
  # paste(t1-t0)    #### timing
  rownames(s$hist) <- 1:dim(s$hist)[1]
  s$psum<- apply(s$hist,MARGIN=1,FUN=mean)
  return(s)
}



## ------------- Plotting helpers functions-----------------------------------------------------------------

##############################################
###  mfrow = c(1,2) with extra params #######
##############################################
init_row21 <- function(){
  par(mfrow=c(1,2)
      ,mar=c(5.1,4.1,1.6,.0)
      ,mai=c(0.6,0.6,0.9,0.)
      ,mgp=c(1.2,0.4,0.1)
      ,cex.axis = 0.8
      ,tcl=-0.5
      # ,family="serif"
      ,omi=c(0.2,0.0,0.5,0.)
      ,oma=c(1,1,1,1)
      ,xpd=F
  )

}

##############################################
###  plotting N(s,t)                  #######
##############################################
simple_vis <-function(s1){
  init_row21()
  par(xpd = T)
  ts = 1:s1$tmax
  ys <- f(s1$psum)
  xlab = expression(
    italic('t')
  )
  ylab = expression(
    E[s](N[t](s))
  )

  ytk = 10^(-1:5)
  ytk_act = f(ytk)


  plot(ts, ys,type='l',ylim =range(ytk_act),
```

```
      ylab = ylab,
      main = 'Average Population',
      xlab = xlab,
      axes = F
  )
  axis(2, at=f(ytk), labels=ytk)
  axis(1)

  par(xpd = F)
  x = 50;y=ys[50];
  y_act = f_i(y)
  text(x,y,sprintf("(%.2f,%.2f)",x,y_act),adj = c(-.3,-.5))
  abline(v = x,h= y,lty=2, col='red')

  x = 1;y=ys[1];
  y_act = f_i(y)
  text(x,y,sprintf("(%.2f,%.2f)",x,y_act),adj = c(-.3,-.5))
  abline(v = x,h= y,lty=2, col='red')

  xs <- 1:s1$tmax
  ys <- 1:s1$L
  zs <- f(s1$hist)
  surf <- persp(1:s1$tmax,
                1:s1$L,
                f(s1$hist),
                xlab = 'time(t)',
                ylab = 'space(s)',
                main = 'Trend of N(s,t): the population \n (f-transformed)',
                # zlab = expressionas indicated in (4).as indicated in (4).
                ,cex.axis = .7
                ,cex.lab = .8
                ,ticktype =
                  "detailed"
                ,zlab = 'f( N(t,s) )'
                ,theta = -25
                ,phi = 20
                ,col = 'lightblue'
                ,border= NA
  )


  lnum = 20
  for(i in floor(seq( 1, length(ys), length=lnum+1)[1:lnum]+length(ys)/2/lnum)) lines(trans3d(xs, ys[i],
  lnum = 20
  for(i in floor(seq( 1, length(xs), length=lnum+1)[1:lnum]+length(xs)/2/lnum)) lines(trans3d(xs[i], ys,
}

##############################################
###  plotting nu(s,t)                  ######
##############################################

##############################################
###  Adding a line on lambda_gloabl    ######
```

```
##############################################

diff_add <- function(s1,yname = "lglobal",addtext = T,...){
  ts = 2:s1$tmax
  kwargs<-list(...)
  kwargs$x <- switch (
    yname,
    "lglobal"= diff(f(s1$psum)),
    "Ev" = apply(diff(f(s1$hist)),MARGIN=1,FUN=mean)
  )
  adj = if (is.null(kwargs$adj)){ c(1.5,-.5)} else{kwargs$adj}
  kwargs$adj = NULL
  # kwargs$x = ys
  # kwargs$x = ts
  do.call("lines", args = kwargs )

  par(xpd = F)
  x = tail(ts,1);y=tail(kwargs$x,1);
  y_act = 10^(y)
  if (addtext){text(x,y,parse(text = sprintf("lambda[end]==%.3f", y_act)), adj = adj,col = kwargs$col)}
  # text(x,y,parse(text = "lambda[end]\"=0.075\""),adj = c(1.5,-.5))
  abline(v = x,h= y,lty=2, col='red')

}



diff_vis_init <- function(s1 , main = 'Global effective growth rate'){
  ts = 2:s1$tmax
  ys <- diff(log10(s1$psum))
  xlab = expression(
    italic('t')
  )
  ylab = expression(
    lambda[global]
    # f(E[s](N[t](s))) - f( E[s](N[t-1](s)))
  )

  ytk = 10^(-1:5)
  ytk = seq(0.8,1.2, by = 0.1)
  ytk_act = log10(ytk)

  # plot(0,0
  #      ,ylim =range(ytk_act)
  #      ,ylab = ylab,
  #      main = 'Global effective growth rate',
  #      xlab = expression(italic('t')),
  #      axes = F
  # )
  plot(ts, -1+ts*0,
       ,type='l'
       ,ylim =range(ytk_act)
       ,ylab = ylab,
```

```
      main = main,
      xlab = expression(italic('t')),
      axes = F
      ,col = 'red'
  )
  axis(2, at=log10(ytk), labels=round(ytk,3))
  axis(1)

}

diff_vis<-function(s1){
  init_row21()
  diff_vis_init(s1)
  # diff_add(s1)
  diff_add(s1,yname = "lglobal",col=2)
  diff_add(s1,yname = "Ev",col=1)

  #### Add legend
  legend(
    ,cex = 0.8
    ,x="bottomright"
    # ,x="bottomleft"
    ,legend=c(
      expression(lambda[global])
      ,expression(10^{E[s](nu(s,t))})
    )
    ,col=c("red","black"), lwd=1, lty=c(1,1),
    pch=c(NA,NA) )

  xs <- 2:s1$tmax
  ys <-1:s1$L
  zs <- diff(f(s1$hist))

  bad = 0.9
  good = 1.1
  zlim = c(bad,good)
  zlim = c(1/1.5,1.5)

  xs <- 2:s1$tmax
  ys <-1:s1$L

  par(xpd = T)
  ttl = 'Local effective growth rate'
  surf <- persp(
    xs,
    ys,
    zs,
    xlab = 'time(t)',
    ylab = 'space(s)',
    zlab = '',
    ,ticktype ="detailed"
    ,cex.axis = .8
    ,cex.lab = 1
```

```
    # ,zlab = 'f( N(t,s) )'
    ,zlim = log10(zlim)
    ,theta = -35
    ,phi = 20
    # ,border="blue"
    ,col = 'lightblue'
    ,border= NA
  )
  lnum = 20
  for(i in floor(seq( 1, length(ys), length=lnum+1)[1:lnum]+length(ys)/2/lnum)) lines(trans3d(xs, ys[i],
  lnum = 20
  for(i in floor(seq( 1, length(xs), length=lnum+1)[1:lnum]+length(xs)/2/lnum)) lines(trans3d(xs[i], ys,

  title( ttl, line = 0)

  mxs <- outer(xs,ys, {function(x,y)x}) %f% as.vector
  mys <- outer(xs,ys, {function(x,y)y}) %f% as.vector
  zs = log10(matrix(rep(s1$rate,s1$tmax-1,byrow =T),s1$L,s1$tmax-1)%f%t)
  mzs <- zs %f% as.vector

  pts <- trans3d( mxs, mys, mzs
                  ,pmat=surf)
  points(pts
         ,pch=20
         ,col=2
         ,cex=0.3
  )

  legend( .05,-.52
          ,cex = 0.8
          # ,x="bottomright"
          # ,x="bottomleft"
          ,legend=c(
            expression(log10(lambda(s,t)))
            ," v(s,t) "
            # ,"zlim : []"
          ),
          col=c("red","black"), lwd=1, lty=c(0,1),
          pch=c(20,NA) )
}
```

4.2.3 "plotter.R": Utility functions to use with "ggplot2"

```
###### Reference: ttps://stackoverflow.com/questions/35982640/
grid_arrange_shared_legend <- function(..., nrow = 1, ncol = length(list(...)), position = c("bottom", "
  require(ggplot2)
  require(gridExtra)
  require(grid)
  plots <- list(...)
  position <- match.arg(position)
  g <- ggplotGrob(plots[[1]] + theme(legend.position = position))$grobs
  legend <- g[[which(sapply(g, function(x) x$name) == "guide-box")]]
```

```
  lheight <- sum(legend$height)
  lwidth <- sum(legend$width)
  gl <- lapply(plots, function(x) x + theme(legend.position = "none"))
  gl <- c(gl, nrow = nrow, ncol = ncol)

  ltitle = 0.1
  usetitle = !is.null(top)
  ltitle = ltitle * (usetitle)
  # cat(ltitle)
  combined <- switch(position,
                     "bottom" = arrangeGrob(do.call(arrangeGrob, gl),
                                            legend,
                                            ncol = 1,
                                            heights = unit.c(unit(1.0 - ltitle, "npc") - lheight, lheigh
                                            ,top = top
                     ),
                     "right" = arrangeGrob(do.call(arrangeGrob, gl),
                                           legend,
                                           ncol = 2,
                                           widths = unit.c(unit(1.0, "npc") - lwidth, lwidth)
                                           ,top = top
                     )

  )
  grid.newpage()
  grid.draw(combined)
}
```