

CALCULATING ZIPF'S LAW

(and building growth curves)

Marjolein van Egmond – M.vanEgmond1@uu.nl

Oct, 2013

Disclaimer: This protocol explains how *I* do it.
That might not be the most efficient way, so
if you know a different/better way to do it:
please let me know!

R's help functions:

```
> ?plot           #for help on a specific function
> ??plot         #for a general search on your search term
```



1. Getting started


At the end of this session I'd like to know what your results look like. Take a moment to make a (word) document with the following: the details of the text you are analysing (author, title, where you found it), the graphs of the alpha and beta distribution and the value of the slope and adj. R^2 . Send it to M.vanEgmond1@uu.nl. Not for a grade, but because I'm genuinely interested. Thanks!

You should have a text ready in .txt format. If not, you can download one here: <http://www.gutenberg.org/>. Download the plain text UTF-8 version and take a few seconds to open the file and manually remove the little bit of formal rubbish from the Gutenberg project at the beginning and end of the file.

In the examples I am using *Metamorphosis* by Franz Kafka (<http://www.gutenberg.org/ebooks/5200>). Please change 'kafka' with some short name for your own text.

During this session you will create some files, so it is probably easiest if you make a new folder for this session. Store your text file in this folder. Also store the file 'commands.R' that has been distributed to you in this folder.

Open R Studio. This program makes using R a little bit more user-friendly. You now see three or four windows. Go to the window on the bottom right. Select the tab "Files". Click on the button  on the top right of this window, on the right of the path name. A new window opens, where you can browse for folders. Select the folder where your text is stored and press OK. Now, go to  and click on "Set As Working Directory". Double click on the file 'commands.R' to open it in the top left window.

These analyses require some additional packages from R, which we will install along the way. One package will be needed right away, which is the tm-package. There are two ways to get it. Type the following, or set your cursor on the command in the 'commands.R' file in the top left window and press :

```
> install.packages("tm", repos = "http://cran.r-project.org")
> library(tm)
```

The second way: go to the bottom right window and select the tab 'Packages'. Check if tm is already in the list. If it is, tick the box. If not, click 'Install Packages' in the ribbon above the list (and below the tabs). In the box in the middle, type 'tm' and press 'Install'. Now it's in your list. Look it up and tick the box.

2. Preparing your text

You can read in your text using the command¹:

```
> kafka <- scan("metamorphosis.txt", what = "character")
```

Check what it looks like with `head(kafka)`.

For the coming analysis to work we first need to do a bit of cleaning up. Punctuation markers should be removed, and all capitals should be changed into lower case (otherwise words at the beginning of sentences will be considered different from words in the middle of the sentence)².

First the removal of punctuation (note: this function comes from the package `rm`).

```
> kafka <- removePunctuation(kafka)
```

Now changing it into lower case:

```
> kafka <- tolower(kafka)
```

Check again with `head(kafka)` if everything worked out all right.

Some additional commands that might be useful:

```
> kafka <- kafka[-1]           # remove the first word
> kafka <- kafka[-(1:4)]      # remove the first four words
> kafka[1] <- "anything"     # change the first word into "anything"
```

There is one problem with your file now. Quotes were originally read in as one word, even if they consisted of multiple words. We don't want that. We removed the quotation marks now, but they are still considered one word. The easiest way to solve this is by storing the file and opening it again. This works as follows:


Write the new version to a file:

```
> file <- file("kafka_plain.txt")
> writeLines(kafka, file, sep = " ")
> close(file)
```

And open it again:

```
> kafka <- scan("kafka_plain.txt", what = "character")
```

(There might be a more elegant way of doing this, but this method works.)

¹ This command and all the following commands are also given in 'commands.R'. You can run them without retyping them by putting your cursor on the command in the 'commands.R' file in the top left window in R Studio and pressing .

² Note that this is a choice. It is my choice that punctuation and capitals are not important, but you might want to make a different choice!

3. Making a word frequency list

What we need now is a list of word frequencies: for each word, you want to know how often it occurs in the text. We use the function `table()` to do this.

```
> kafka.table <- table(kafka)
> kafka.table <- sort(kafka.table, decreasing = TRUE)
> kafka.table[1:10]
kafka
  the  to  and  he  his  of  was  had  it  in
1148 753 642 576 550 429 409 352 352 348
```

I prefer to build a data frame for the rest of the analyses. I want a column with rank, one with frequency and one with the type (the word). So let's build it:

```
> a.kafka = data.frame(rank = c(1:length(kafka.table)), freq =
+ as.numeric(kafka.table), type = names(kafka.table))
```

The `a.` in front of the name is there because this is the list we need for the alpha analysis.

Word frequencies are usually displayed on logarithmic scales. To make this easier, add two extra columns:

```
> a.kafka$logfreq <- log2(a.kafka$freq)
> a.kafka$logrank <- log2(a.kafka$rank)
```

Now you have created a data frame like this:

```
> head(a.kafka, 4)
  rank freq type  logfreq logrank
1     1 1148  the 10.164907 0.000000
2     2  753   to  9.556506 1.000000
3     3  642  and  9.326429 1.584963
4     4  576   he  9.169925 2.000000
5     5  550  his  9.103288 2.321928
6     6  429   of  8.744834 2.584963
```

You might want to save this as a separate file, by typing

```
> write.table(a.kafka, "a.kafka.txt")
```

Which, when needed later, you can read in with:

```
> a.kafka <- read.table("a.kafka.txt", header = TRUE)
```

4. Calculating alpha

The alpha-formulation of Zipf's law is as follows: $f_k \propto k^{-\alpha}$. In words: the frequency of rank k is proportional to the power of minus alpha (where alpha is traditionally said to be 1, approximately). What you want to calculate is the value of alpha; rank and frequency are known. When you plot rank x word frequency on logarithmic scales, you will find approximately a straight line.

Make a plot to see if word frequencies in your text look anything like Zipf's law:

```
> plot(a.kafka$logrank, a.kafka$logf, main = "Alpha distribution  
+ Kafka", xlab = "log rank", ylab = "log frequency")
```

For the *Metamorphosis*, this results in the graph in figure 1 (we'll draw the line in later). Can you figure out why the steps are there for the lower ranks?³

Sidenote I: You can specify the x-axis range by using `xlim = c(0,8)`, and similarly for the y-axis with `ylim`. A different colour is given by `col = "red"`, a different line type is given by `lty = 2` (any number from 1 to 6).

Sidenote II: if you want multiple plots next to each other, use

```
> par(mfrow = c(1,2))
```

Where 1 specifies the number of rows and 2 the number of columns.

We can now apply linear regression to the frequency list to see how well the data points follow a straight line and to find out what the slope of this line would be. For this, you have to do simple linear regression on the log-values. But there's a problem with this kind of data: there are many more cases in the low frequency range than in the high frequency range, which attract your slope. To prevent this, you'll have to weigh your cases by frequency. This tells the formula to contribute relatively more importance to the high frequency values.

This is what you have to type in to do this:

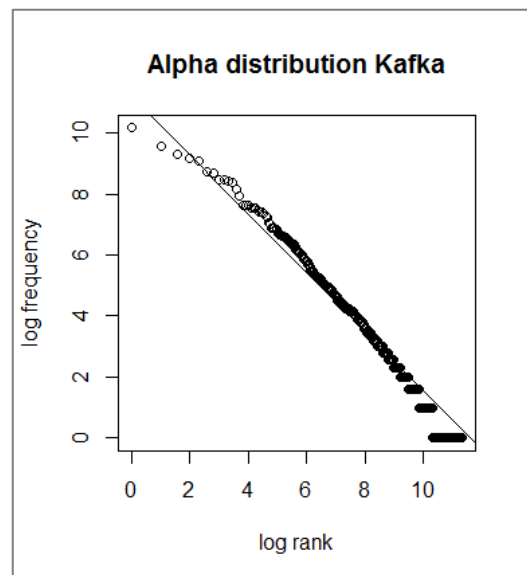



Figure 1. Alpha distribution of Franz Kafka's *Metamorphosis*

³ Think of the kind of data you're working with: it's count data. Words can only occur once, twice, three times, etc., not 1.74 times. A large number of words occurs only once, and $\text{Log } 1 = 0$, so that explains the step at log frequency = 0.

```
> a.kafka.lm <- lm(logfreq ~ logrank, weights = freq, data = a.kafka)
```

Now you can draw in the line of Zipf's law into your previous plot:

```
> abline(a.kafka.lm)
```

Please copy your graph to your document with results. Go to the window with your graph in it and click on . Click on 'Copy Plot to Clipboard...' and click 'Copy Plot'. You can now easily paste it into your document.

You can see the slope value and fit by typing in

```
> summary(a.kafka.lm)
```

Pay attention to the coefficient estimate for logrank and the adjusted R-squared value. The first value is usually close to 1 (although deviations from this typical value have been reported for individuals with impaired speech and other non-typical text genres); the second should be close to 1. In this case, this gives you (the values to report are colored yellow):

```
> summary(a.kafka.lm)
```

Call:

```
lm(formula = logfreq ~ logrank, data = a.kafka, weights = freq)
```

Weighted Residuals:

Min	1Q	Median	3Q	Max
-35.201	-0.806	-0.545	-0.330	9.095

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	11.203845	0.017900	625.9	<2e-16 ***
logrank	-0.960530	0.002787	-344.6	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.246 on 2618 degrees of freedom

Multiple R-squared: 0.9784, Adjusted R-squared: 0.9784

F-statistic: 1.188e+05 on 1 and 2618 DF, p-value: < 2.2e-16

So, you can see that these values have a slope of 0.960 (you can drop the minus sign since it is already in the formula), with an adjusted R^2 of 0.978. This means that 97.8% of all variability in your data is explained by the formula. This is a good fit, so we can conclude that Zipf's law applies.

Please copy these values to your results document.

5. Calculating Beta

The beta-formulation of Zipf's law is: $n_f \propto f^{-\beta}$. In words: the number of words whose frequency is f in a given text (n_f) is approximately equal to frequency f to the power of minus beta (where beta is traditionally said to be 2, approximately). You want to calculate beta.

This formula has some advantages over the alpha-formulation. First, you don't have to introduce an additional variable 'rank', the formula only needs the variable 'frequency'. And second, where in the alpha formulation the data *has* to follow a decreasing line due to the ranking, this is not the case in the beta-formulation. This renders this formulation more sensitive.

But this formulation also has some disadvantages. In large texts, the distribution that you get is accurate primarily for the frequency classes of the smaller frequencies. Simple linear regression therefore only works for the first approximately 15 points.⁴

Despite these problems it is insightful to see what this distribution looks like, so let's have a look.

For alpha, you made a count list per word. Now, you need a list of the size of frequency classes. In other words, how many words occur exactly once, how many words occur exactly twice, etc.

In R terms, what you need to do is count how often each frequency value occurs (try out `table(a.kafka$freq)` to see the result of this), and put that in a data frame with column names `freqclass` and `classsize`. You can add in extra columns by specifying them, to get all the required columns. This can look something like this:

```
> b.kafka = data.frame(freqclass =  
+ as.numeric(names(table(a.kafka$freq))), classsize =  
+ as.numeric(table(a.kafka$freq)))
```

Again, you need log-scales, so we might as well add those columns right away:

```
> b.kafka$logfreqclass <- log2(b.kafka$freqclass)  
> b.kafka$logclasssize <- log2(b.kafka$classsize)
```

⁴ For a detailed discussion of this problem and some solutions, see p. 19-24 of R.H. Baayen (2001), *Word Frequency Distributions*, Kluwer Academic Publishers Dordrecht/Boston/London.

Now, we have created the following data frame:

```
> head(b.kafka)
  freqclass classsize logfreqclass logclasssize
1         1     1318     0.000000     10.364135
2         2      375     1.000000     8.550747
3         3      211     1.584963     7.721099
4         4      129     2.000000     7.011227
5         5       79     2.321928     6.303781
6         6       63     2.584963     5.977280
```

You can see now that there are 1318 different types (words) that occur exactly once, 375 types that occur exactly twice, etc.

Again, you might want to save this as a separate file, by typing

```
> write.table(b.kafka, "b.kafka.txt")
```

You can now plot log frequency class x log class size:

```
> plot(b.kafka$logfreqclass, b.kafka$logclasssize, main = "Beta
+ distribution Kafka", xlab = "log frequency class", ylab = "log
+ class size")
```

The result for Kafka can be seen in Figure 2 (again, we'll draw in the lines later). You can easily see that the first few frequency classes form a straight line, while the higher frequency classes are much more messy.

Now, we can calculate the value of beta, similarly to how we calculated the value of alpha. There are two big differences though. The first is that we now don't need a weighting variable, because the values are much more evenly distributed over the x-axis. The second is, as mentioned above, that your linear regression only works if you only look at the first approximately 15 ranks.

Taking this all into account, you can type something like this:

```
> b.kafka15.lm <- lm(logclasssize[1:15] ~ logfreqclass[1:15], data =
+ b.kafka)
```

Add the regression line to your plot:

```
> abline(b.kafka15.lm)
```

If you like, you can check what the regression line looks like if you include all data points:

```
> b.kafka.lm <- lm(logclasssize ~ logfreqclass, data = b.kafka)
> abline(b.kafka.lm, lty = 2)
```


You can now inspect the outcome of the linear regression, and see what the value of Beta is (compare Beta and its adj. R^2 for the first 15 values and for the whole spectrum):

```
> summary(b.kafka15.lm)

Call:
lm(formula = logclasssize[1:15] ~ logfreqclass[1:15], data = b.kafka)

Residuals:
    Min       1Q   Median       3Q      Max
-0.83252 -0.05989  0.06045  0.15111  0.28607

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.40145    0.20040   51.90 < 2e-16 ***
logfreqclass[1:15] -1.72925    0.06919  -24.99 2.25e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2921 on 13 degrees of freedom
Multiple R-squared:  0.9796, Adjusted R-squared:  0.978
F-statistic: 624.6 on 1 and 13 DF,  p-value: 2.247e-12
```

The relevant value to look at is the estimate for logFreqClass, in this case slope = 1,723. This slope is relatively close to the traditionally reported $\beta = 2$.

The adj. R^2 in this example seems to be very high (0.978), but it is meaningless: it is only the fit of the model to the first 15 ranks and does not say anything about the model in general.

Please copy the value of beta and the graph to your results document.

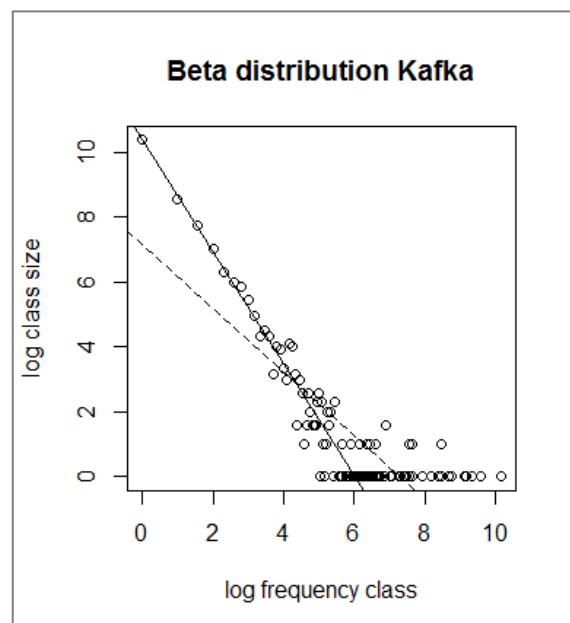


Figure 2. Beta distribution of Franz Kafka's *Metamorphosis*

You can now send your results to me at M.vanEgmond1@uu.nl. Thanks!

**Everything worked so far?
You've sent me your file?
Good!**

Then now the choice is up to you. You have two options:

Option 1:

Try out the analyses above on different kinds of texts. It has been claimed that Zipf's law applies to any text in any language, only the slope differs. But is that really the case? What happens in 'weird' texts, such as spontaneous speech (either normal or by people with speech or language disorders), texts written by multiple authors, texts from languages with a different kind of writing system (e.g. Chinese)... Try to think of texts as crazy as possible and try it out! Please let me know what you tried and send me results files of everything.

Option 2:

You can continue to work with the text that you were working with on another, related point, namely vocabulary growth curves (you don't have to send me the results on this). These analyses might be more interesting to you if you are a more advanced R user.

6. Growth curves

[The exercises below come from R.H. Baayen (2008), *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*. Cambridge University Press.)

If you read through a text, and at regular intervals keep note of how many different words (types) you have encountered, you find that this number increases. This in itself is not very interesting. What is interesting, though, is the rate at which this happens. First, every word is new, so the number of new types is quickly increasing. But once you have seen a large number of types, you will only rarely encounter a new one. We can visualize this process in what is called a 'growth curve'.

You'll need an additional package for this analysis to work, namely `languageR`. Again, there are two ways to get it. Type this:

```
> install.packages("languageR", repos = "http://cran.r-project.org")
> library("languageR")
```

Or go to the bottom right window and select the tab 'Packages'. Check if `languageR` is already in the list. If it is, tick the box. If not, click 'Install Packages' in the ribbon above the list and below the tabs. In the box in the middle, type 'languageR' (it will automatically find it for you after you've typed the first few letters) and click 'Install'. Now it's in your list. Look it up and tick the box.

We are going to calculate some lexical measures at fixed intervals. But what intervals would make sense?

Check what the number of words in your text is:

```
> length(kafka)
[1] 22017
```

What works well for this number is 50 chunks with 440 words in each. Determine what would work for you (choose chunks of about 400-600 words).

Make a growth object with these numbers (the default values are `size = 646` and `nchunks = 40`):

```
> kafka.growth = growth.fnc(text = kafka, size = 440, nchunks = 50)
```

(For large texts, this might take a while.)

Inspect what you have now:

```
> head(kafka.growth)
Chunk Tokens Types HapaxLegomena DisLegomena TrisLegomena
1      1    440   253           194           32           13
2      2    880   402           270           64           26
3      3   1320   533           345           83           42
4      4   1760   626           394          101           46
5      5   2200   703           425          120           50
6      6   2640   790           475          130           59

      Yule      Zipf TypeTokenRatio      Herdan  Guiraud
1  90.80579 -0.6017242      0.5750000 0.8365160 12.06130
2  85.53719 -0.6676721      0.4568182 0.7768991 13.55142
3  82.04775 -0.7066615      0.4037879 0.7433030 14.67034
4  92.60718 -0.7551366      0.3556818 0.7266270 14.92169
5  98.42149 -0.7816206      0.3195455 0.7083707 14.98801
6 100.82071 -0.8032514      0.2992424 0.6879187 15.37535

      Sichel Lognormal
1  0.1264822 0.2667653
2  0.1592040 0.3916670
3  0.1557223 0.4407961
4  0.1613419 0.4808600
5  0.1706970 0.5240188
6  0.1645570 0.5399684
```

These columns should be interpreted as standing next to each other. The first three columns list the indices of the chunks, the corresponding (cumulative) number of tokens, and the count of the number of types up to that point in the text. The next three columns list the number of hapax, dis and tris legomena, the number of words that occur exactly once, two times or three times in the text. The remaining columns list various measures of lexical richness, of which Zipf's law (alpha, in this case) will be familiar.

You can now easily make a plot of these different variables. For w , give the name of any of the columns seen above. If you don't specify w you will get plots for types and for the five measures of lexical richness.

```
> plot(kafka.growth, w = "types")
```

You can find the result of this in Figure 3. Try some different plots and try to interpret what you see. What does that curve mean?

You now have the growth object, but if you want to use the values in there you will need to extract them. You can extract the whole data frame that is internal to `kafka.growth` as follows:

```
> kafka.g <- kafka.growth@data$data
```

Now you can more easily make your own custom plots. Like the one in Figure 4, which was constructed as follows:

```
> plot(kafka.g$Tokens, kafka.g$HapaxLegomena, xlab = "Tokens", ylab =  
+ "Legomena", ylim = c(50,1400), pch = 3)  
> points(kafka.g$Tokens, kafka.g$DisLegomena, pch = 4)  
> points(kafka.g$Tokens, kafka.g$TrisLegomena, pch = 8)
```

One problem that you will always encounter if you work with this kind of data is that your outcomes are dependent on text size. This is due to the large number of low probability elements. Such distributions are called LNRE (Large Number of Rare Events) distributions. No matter how big your sample is, if it grows you will find new words that did not yet occur in your sample. The joint probability of all these unseen words is usually so substantial that the relative frequencies in the sample become inaccurate estimates of the real probabilities. Relative frequencies sum up to 1, so they do not leave any space for unseen types in the population. So you have to correct for this. One way of doing this is through the growth rate of the vocabulary.

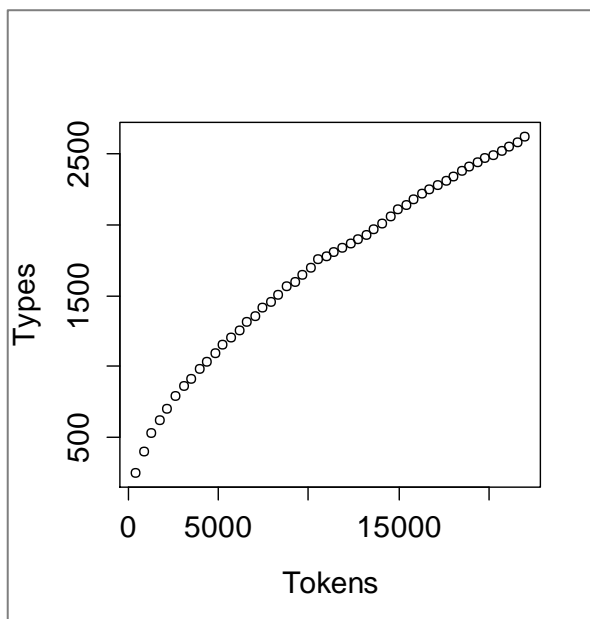


Figure 3. Type growth curve of Franz Kafka's *Metamorphosis*

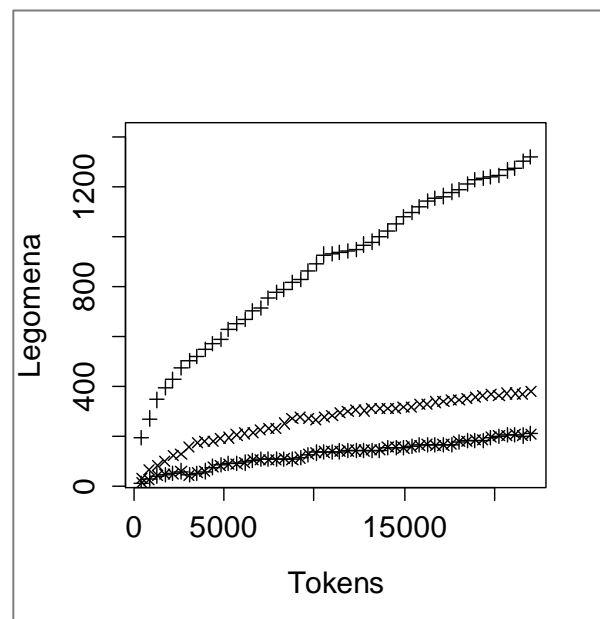


Figure 4. Growth curves for hapax legomena (+), dis legomena (x) and tris legomena (*) of Franz Kafka's *Metamorphosis*

We are going to use yet another package for this, the zipfR package. Install and load this package. You should know by now how to do that.

We are going to construct a `spc` object now, because that is what zipfR works with. This is basically the same as the data frame that you constructed for the calculation of beta. The reason why we first constructed that data frame and not a `spc` object is because that data frame contained more information, in a more transparent way, than a `spc` object does. But now we want to use zipfR so we have to. Luckily, it is not difficult. We start from the original text, and apply `table` twice.

```
> kafka.table <- table(table(kafka))
> head(kafka.table)
      1      2      3      4      5      6
1318  375  211  129   79   63
```

This shows that there are 1318 hapax legomena, 375 dis legomena, etc. We already saw these values in `b.kafka`.

Now, we're actually going to construct the `spc` object.⁵ We use the function `spc()` for this. Its first argument, `m`, specifies the word frequency classes, its second argument, `Vm`, specifies the frequency class size (so how many words occur with the frequency of that frequency class). Type the following:

```
> kafka.spc <- spc(m = as.numeric(names(kafka.table)), Vm =
as.numeric(kafka.table))
> kafka.spc
      m  Vm
1    1 1318
2    2  375
3    3  211
4    4  129
5    5   79
6    6   63
7    7   57
8    8   44
9    9   31
10  10   20
      ...
      N   V
22017 2620
```

⁵ This method should only be applied to texts or corpora with less than a million words, otherwise it will take forever.

These objects have a summary method, which lists the first ten elements together with the total number of tokens N and total number of types V .

If you like, you can verify these values by looking at `b.kafka`.

We can now fit a LNRE model to this text, using the function `lnre()`. This function takes two arguments, the type of model and an `spc` object. For details, see

```
> ?lnre
```

For now, we are using the method `gigp`, Generalized Inverse Gauss-Poisson model.

```
> kafka.lnre.gigp = lnre("gigp", kafka.spc)
```

```
> kafka.lnre.gigp
```

```
Generalized Inverse Gauss-Poisson (GIGP) LNRE model.
```

```
Parameters:
```

```
Shape:          gamma = -0.6705846
```

```
Lower decay:    B = 0.02137325
```

```
Upper decay:    C = 0.02686677
```

```
[ Zipf size:    Z = 37.2207 ]
```

```
Population size: S = 8747.398
```

```
Sampling method: Poisson, with exact calculations.
```

```
Parameters estimated from sample of size N = 22017:
```

	V	V1	V2	V3	V4	V5	
Observed:	2620.00	1318.00	375.00	211.00	129.00	79.00	...
Expected:	2609.44	1308.27	422.09	201.43	119.48	80.11	...

```
Goodness-of-fit (multivariate chi-squared test):
```

X2	df	p
29.25494	13	0.006021914

We skip the technical details about the model, which are listed first. LNRE models take the mass of unseen types into account, which allows them to give an estimate of the population number of types. This estimate represents the number of words that Kafka would have considered appropriate to use when writing more books like the *Metamorphosis*. In this case, that is an estimated 8747 words.

The next thing that this summary gives you is a comparison between the observed number of types, number of hapax legomena, dis legomena, etc., and the expected number in these categories according to the LNRE model. This comparison can be plotted:

```
> plot(kafka.spc, lnre.spc(kafka.lnre.gigp, 22017))
```

You can find the plot for Kafka's *Metamorphosis* in Figure 5.

The last thing in the summary is the goodness-of-fit of this model for these data. For a good fit, χ^2 should be low and the corresponding p-value large and preferably well above 0.05. Clearly, this model is not that good. You can see this discrepancy between model and reality also from the bar plot, where you see that the expected number of dis legomena is somewhat larger than the observed number. This discrepancy is probably due to the statistical theory underlying the model that was used. LNRE models assume that words are used at random and independently of each other in texts. This is of course a simplification, and likely the reason for the poor fit.

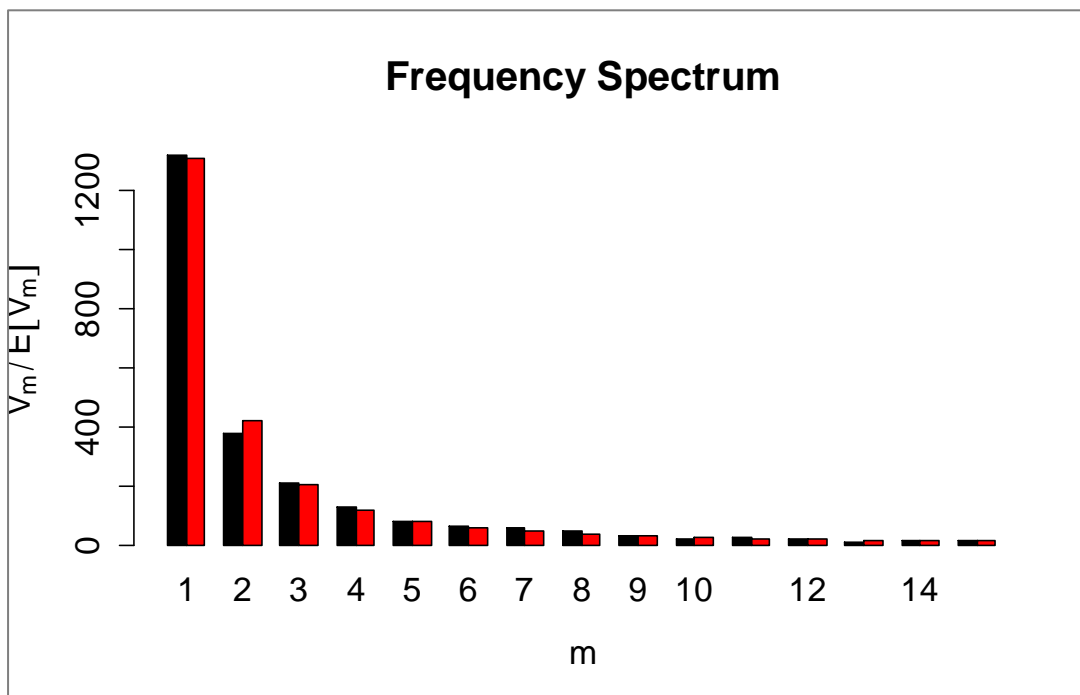


Figure 3. Observed (black) and expected (red) counts for hapax legomena, dis legomena, etc.

For now, we are going to continue with this model anyway. Once a LNRE model has been fitted to a frequency spectrum, the model can be used to obtain expected values for the vocabulary size and frequency class sizes both at smaller samples (interpolation) and at larger sample sizes (extrapolation). We can build nice graphs that visualize the expected vocabulary growth curves for us.

For this, we have the function `lnre.vgc()`, for building vocabulary growth objects. First we are going to do the extrapolation. The function `lnre.vgc()` takes as arguments a fitted model (the one we just built), a sequence of sample sizes (`N()` is used to access sample sizes in `vgc` objects, and we take 20 sample sizes from the size of the sample up to twice the size of the sample), and the number of frequency classes for which you want curves (here, 3).

```
> kafka.ext.gigp <- lnre.vgc(kafka.lnre.gigp, seq(N(kafka.lnre.gigp),
  N(kafka.lnre.gigp)*2, length = 20), m.max = 3)
```


Next, we are going to do the interpolation, in a similar fashion. The difference is that we now take sample sizes within the original sample size, so 20 samples from 0 up to the original sample size:

```
> kafka.int.gigp <- lnre.vgc(kafka.lnre.gigp, seq(0,
+ N(kafka.lnre.gigp), length = 20), m.max = 3)
```

We now have `vgc` objects for the interpolation and extrapolation, but we would also like to add the original data to the graph. For this, we need to convert the growth object that we had into a vocabulary growth object:

```
> kafka.vgc <- growth2vgc.fnc(kafka.growth)
```

Now we can build the plot that can be seen in Figure 6. This is what you should type:

```
> plot(kafka.int.gigp, kafka.ext.gigp, kafka.vgc, add.m = 1:3, main =
+ "Vocabulary Growth")
> mtext(c(" V3", " V2", " V1", " V"), side = 4, at = c(290, 600,
+ 1600, 3600), las = 1)
```

The thick line on top gives you the vocabulary growth. It is composed from the interpolation (the black line, coming from `kafka.int.gigp`), the extrapolation (the red line, coming from `kafka.ext.gigp`) and the original data (the green dashes, coming from `kafka.vgc`). The first thin curve below the thick one gives you the growth curve of the hapax legomena. The second curve gives you the growth curve of the dis legomena, and the last curve that of the tris legomena. You can plot only the vocabulary growth curve by leaving out `add.m = 1:3` from the plot command. (If you don't need the curves for the hapax, dis and tris legomena at all, you can also leave out `m.max = 3` from the extra- and interpolation command.)

The `mtext` command gives you the labels at the end of the lines. The argument `side` tells it to use the margin on the right, `at` tells it where to place the labels, and `las` specifies the reading direction. Try playing around with this to see what it does.

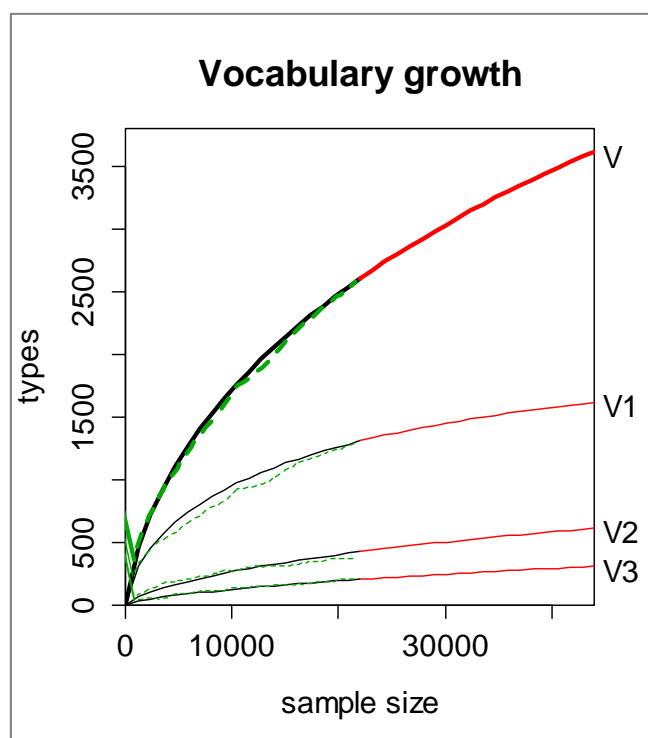


Figure 4. Vocabulary growth curves of Franz Kafka's *Metamorphosis*