GRCRiddles Documentation

Release 1a

Marko Manninen

Apr 13, 2018

Contents

1 Processing Greek corpora for the riddle solver **3**
- 1.1 Focus of the study . . . 3
  - 1.1.1 Natural language processing . . . 5
  - 1.1.2 Statistics . . . 8
- 1.2 Table data . . . 10
- 1.3 Bar chart . . . 10
- 1.4 Optional live chart . . . 11
  - 1.4.1 Unique words database . . . 11
- 1.5 Longest words . . . 14
- 1.6 Highest isopsephy . . . 17

2 Isopsephical riddles in Greek Pseudo Sibylline hexameter poetry **21**
- 2.1 Introduction . . . 21
  - 2.1.1 Riddle 1 . . . 23
  - 2.1.2 Riddle 2 . . . 23
  - 2.1.3 Riddle 3 . . . 24

3 Study of the results of the riddle solver **33**

4 Appendix 1 - Store database **35**

5 Appendix 2 - Solve riddles **37**

In the following chapters, I'm guiding through the process of solving some enigmatic alphanumeric riddles in the Greek Pseudo Sibylline oracles.

In the first chapter, I will download and preprocess freely available open source Greek corpora. In the second chapter, I will use the processed unique word database to solve some specific riddles in the Sibylline oracles. The third chapter is reserved more for the speculative analysis of the results.

The reader may download all material and code for the interactive exploration from the GitHub project: https://github.com/markomanninen/grcriddles

Contents:

CHAPTER 1

Processing Greek corpora for the riddle solver

Pseudo-Sibylline[1] oracles contain hexametric poems written in Ancient Greek. These *oracula* were mainly composed in 150BC - 700AD to twelve distinct extant books. They were circulating and quite famous among the Judaeo-Christian community at that time.

They shouldn't, however, be too much confused with the earlier Sibylline books[2]. Sibylline books contained religious ceremonial advice that were consulted by the selected priests and curators in the Roman state when it was in deep political trouble. The collection of the original Sibylline books were destroyed by different accidental events and deliberate actions in history.

Pseudo-Sibylline oracles, on the other hand, contain Jewish narrative of the human history contrasted with the Greek mythology and to the chronology of the other great ancient empires. Another intention of the oracles was to support evolving Christian doctrine and interpretation of the prophecies. Prophecies were mostly grounded in Jewish tradition, but surprisingly some pagan world events also came to be interpreted as signs of the coming Messiah. Sibyl, the acclaimed author of the prophecies as a woman prophetess, the daughter of Noah in the Pseudo-Sibylline lore, has a unique character crossing over the common borders in several ancient religions and mythology.

Good introductions to the Pseudo-Sibylline oracles can be found from these two books:

- Sibylline Oracles in ‘The Old Testament Pseudepigrapha, Volume I

<https://books.google.fi/books?id=TNdeolWctsQC>‘__[3] by J. J. Collins

- Part 1 in ‘The Book Three of the Sibylline Oracles and Its Social Setting

<https://books.google.fi/books?id=Zqh8ZQZqnWYC>‘__[4] by Rieuwerd Buitenwerf

1.1 Focus of the study

Some material in the Pseudo-Sibylline oracles contains cryptic puzzles, referring to persons, cities, countries, and epithets of God for example. These secretive references are often very general in nature, pointing only to the first letter

[1] https://en.wikipedia.org/wiki/Sibylline_Oracles
[2] https://en.wikipedia.org/wiki/Sibylline_Books
[3] https://books.google.fi/books?id=TNdeolWctsQC
[4] https://books.google.fi/books?id=Zqh8ZQZqnWYC

Fig. 1.1: Michelangelo's Delphic Sibyl, Sistine Chapel

of the subject and its numerical value. Solving them requires, not so much of mathematical or cryptographical skills in a modern sense, but a proper knowledge of the context, both the inner textual and the historical context.

Most of the alphanumeric riddles in the oracles can already be taken as solved by various researchers. See footnotes in [The Sibylline Oracles](http://www.sacred-texts.com/cla/sib/sib.pdf) by Milton S. Terry for example. But, some of the riddles are still problematic and open for better proposals. Better yet, few of these open riddles are more complex and specific enough so that one may try to solve them by modern programmable tools.

As an independent researcher not affiliated with any organization, the sole motivation and purpose of mine in this book is to provide a reusable and a testable method for processing and analyzing ancient corpora, especially detecting alphanumeric patterns in a digitalized text. Although the target language in this study is Ancient Greek, the method should be applicable to any language using alphabetic numerals.

1.1.1 Natural language processing

Programmatical approach to solve the riddles requires a huge Greek text corpora. Bigger it is, the better. I will download and preprocess available open source Greek corpora, which is a quite daunting task for many reasons. Programming language of my choice is Python[5] for it has plenty of good and stable open source libraries required for my work. Python is widely recognized in academic and scientific field and well oriented to the research projects.

I have left the most of the overly technical details of these chapters for the enthusiasts to read straight from the commented code in functions.py[6] script. By collecting the large part of the used procedures to the separate script maintains this document more concise too.

In the end of the task of the first chapter, I'll have a word database containing hundreds of thousands of unique Greek words extracted from the naturally written language corpora. Then words can be further used in the riddle solver in the second chapter.

Note: Note that rather than just reading, this, and the following chapters can also be run interactively in your local Jupyter notebook[7] installation if you prefer. That means that you may test and verify the procedure or alter parameters and try solving the riddles with your own parameters.

Your can download independent Jupyter notebooks for processing corpora[8], solving riddles[9], and analysing results[10].

You may also run code directly from Python shell environment, no problem.

Required components

The first sub task is to get a big raw ancient Greek text to operate with. I have implemented an importer interface with tqdm library to the Perseus[11] and the First1KGreek[12] open source data sources in this chapter.

I'm using my own Abnum[13] library to remove accents from the Greek words, remove non-alphabetical characters from the corpora, as well as calculating the isopsephical value of the Greek words. Greek accentuation[14] library is used to split words into syllables. This is required because the riddles of my closest interest contain specific information about

[5] http://python.org
[6] https://github.com/markomanninen/grcriddles/blob/master/functions.py
[7] https://jupyter.org
[8] https://github.com/markomanninen/grcriddles/blob/master/Processing%20Greek%20corpora%20for%20the%20isopsehical%20riddle%20solver.ipynb
[9] https://github.com/markomanninen/grcriddles/blob/master/Isopsephical%20riddles%20in%20the%20Greek%20Pseudo%20Sibylline%20hexameter%20poetry.ipynb
[10] https://github.com/markomanninen/grcriddles/blob/master/
[11] https://www.python.org/shell/
[12] https://github.com/tqdm/tqdm
[13] http://www.perseus.tufts.edu/hopper/opensource/download
[14] http://opengreekandlatin.github.io/First1KGreek/

the syllables of the words. Pandas[15] library is used as an API (application programming interface) to the collected database. Plotly[16] library and online infographic service are used for the visual presentation of the statistics.

You can install these libraries by uncommenting and running the next install lines in the Jupyter notebook:

```
import sys

#!{sys.executable} -m pip install tqdm abnum requests
#!{sys.executable} -m pip install pandas plotly pathlib
#!{sys.executable} -m pip install greek_accentuation
```

For your convenience, my environment is the following:

```
print("Python %s" % sys.version)
```

Output:

```
Python 3.6.1 | Anaconda 4.4.0 (64-bit) | (default, May 11 2017, 13:25:24)
[MSC v.1900 64 bit (AMD64)]
```

Note that *Python 3.4+* is required for all examples to work properly. To find out other ways of installing PyPI maintained libraries, please consult: https://packaging.python.org/tutorials/installing-packages/

Downloading corpora

I'm going to use *Perseus* and *OpenGreekAndLatin* corpora for the study by combining them into a single raw text file and unique words database.

The next code snippets will download hundreds of megabytes of Greek text to a local computer for quicker access. *tqdm* downloader requires a stable internet connection to work properly.

One could also download source zip files via browser and place them to the same directory with the Jupyter notebook or where Python is optionally run in shell mode. Zip files must then be renamed as *perseus.zip* and *first1k.zip*.

1. Download packed zip files from their GitHub repositories:

```
    from functions import download_with_indicator, perseus_zip_file, first1k_zip_file
# download from perseus file source
fs = "https://github.com/PerseusDL/canonical-greekLit/archive/master.zip"
download_with_indicator(fs, perseus_zip_file)
# download from first1k file source
fs = "https://github.com/OpenGreekAndLatin/First1KGreek/archive/master.zip"
download_with_indicator(fs, first1k_zip_file)
```

Output:

```
Downloading: https://github.com/PerseusDL/canonical-greekLit/archive/master.zip
71.00MB [04:15, 211.08KB/s]
Downloading: https://github.com/OpenGreekAndLatin/First1KGreek/archive/master.zip
195.00MB [09:15, 201.54KB/s]
```

2. Unzip files to the corresponding directories:

```
from functions import perseus_zip_dir, first1k_zip_dir, unzip
# first argument is the zip source, second is the destination directory
```

[15] https://github.com/markomanninen/abnum3
[16] https://github.com/jtauber/greek-accentuation

```
unzip(perseus_zip_file, perseus_zip_dir)
unzip(first1k_zip_file, first1k_zip_dir)
```

3. Copy only suitable Greek text xml files from *perseus_zip_dir* and *first1k_zip_dir* to the temporary work directories. Original repositories contain a lot of unnecessary files for the riddle solver which are skipped in this process.

```
from functions import copy_corpora, joinpaths, perseus_tmp_dir, first1k_tmp_dir
# important Greek text files resides in the data directory of the repositories
for item in [[joinpaths(perseus_zip_dir,
               ["canonical-greekLit-master", "data"]), perseus_tmp_dir],
              [joinpaths(first1k_zip_dir,
               ["First1KGreek-master", "data"]), first1k_tmp_dir]]:
    copy_corpora(*item)
```

Output:

```
greek_text_perseus_tmp already exists. Either remove it and run again, or
just use the old one.

Copying greek_text_first1k_tmp -> greek_text_first1k
```

Depending on if the files have been downloaded already, the output may differ.

Collecting files

When the files has been downloaded and copied, it is time to read them to the RAM (Random-Access Memory). At this point file paths are collected to the *greek_corpora_x* variable that is used on later iterators.

```
from functions import init_corpora, perseus_dir, first1k_dir
# collect files and initialize data dictionary
greek_corpora_x = init_corpora([[perseus_tmp_dir, perseus_dir], [first1k_tmp_dir,␣
↪first1k_dir]])
print(len(greek_corpora_x), "files found")
```

Output:

```
1708 files found
```

Actual files found may differ by increasing over time, because Greek corpora repositories are constantly maintained and new texts are added by voluteer contributors.

Processing files

Next step is to extract Greek content from the downloaded and selected XML source files. Usually this task might take a lot of effort in NLP (natural language processing). Python NLTK[17] and CLTK[18] libraries would be useful at this point, but in my case I'm only interested of Greek words, that is, text content encoded by a certain Greek Unicode letter[19] block. Thus, I'm able to simplify this part by removing all other characters from source files except Greek characters. Again, details can be found from the functions.py script.

Extracted content is saved to the *corpora/author/work* based directories. Simplified uncial conversion is also made at the same time so that the final data contain only plain uppercase words separated by spaces. Pretty much in a format written by the ancient Greeks, except they didn't even use spaces to denote individual words and phrases.

[17] http://pandas.pydata.org
[18] https://plot.ly
[19] https://www.nltk.org/

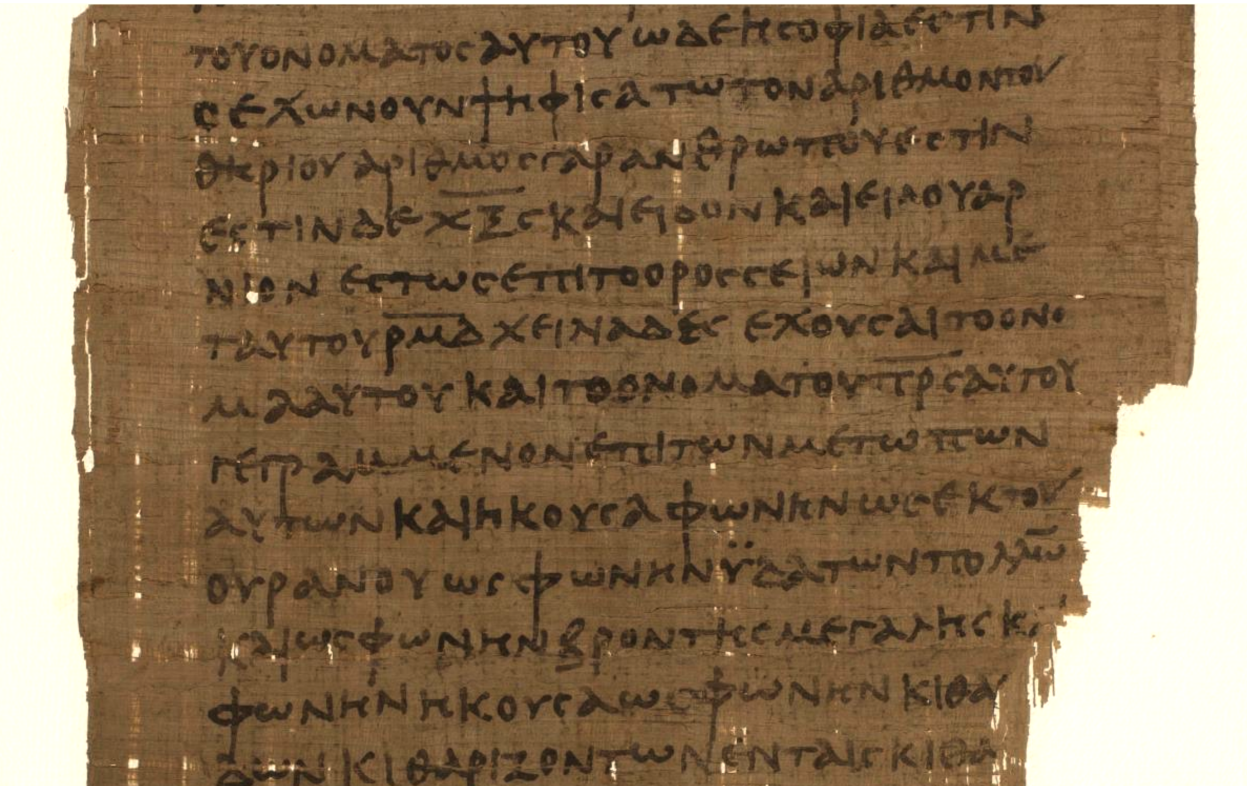

Fig. 1.2: Papyrus 47, Uncial Greek text without spaces. Rev 13:17-

Next code execution will take several minutes depending on if you have already run it once and have the previous temporary directories available. Old processed corpora files are removed first, then they are recreated by calling *process_greek_corpora* function.

```
from functions import remove, all_greek_text_file, perseus_greek_text_file,\
                      first1k_greek_text_file, process_greek_corpora
# remove old processed temporary files
try:
    remove(all_greek_text_file)
    remove(perseus_greek_text_file)
    remove(first1k_greek_text_file)
except OSError:
    pass
# process and get greek corpora data to the RAM memory
greek_corpora = process_greek_corpora(greek_corpora_x)
```

1.1.2 Statistics

After the files have been downloaded and preprocessed, I'm going to output the size of them:

```
from functions import get_file_size

print("Size of the all raw text: %s MB" % get_file_size(all_greek_text_file))
print("Size of the perseus raw text: %s MB" % get_file_size(perseus_greek_text_file))
print("Size of the first1k raw text: %s MB" % get_file_size(first1k_greek_text_file))
```

Output:

```
Size of the all raw text: 347.76 MB
Size of the perseus raw text: 107.41 MB
Size of the first1k raw text: 240.35 MB
```

Then, I will calculate other statistics of the saved text files to compare their content:

```
from functions import get_stats

ccontent1, chars1, lwords1 = get_stats(perseus_greek_text_file)
ccontent2, chars2, lwords2 = get_stats(first1k_greek_text_file)
ccontent3, chars3, lwords3 = get_stats(all_greek_text_file)
```

Output:

```
Corpora: perseus_greek_text_files.txt
Letters: 51411752
Words in total: 9900720
Unique words: 423428

Corpora: first1k_greek_text_files.txt
Letters: 113763150
Words in total: 23084445
Unique words: 667503

Corpora: all_greek_text_files.txt
Letters: 165174902
Words in total: 32985165
Unique words: 831308
```

Letter statistics

I'm using *DataFrame* class from *Pandas* library to handle tabular data and show basic letter statistics for each corpora and combination of them. Native *Counter* class in Python is used to count unique elements in the given sequence. Sequence in this case is the raw Greek text stripped from all special characters and spaces, and elements are the letters of the Greek alphabet.

This will take some time to process too:

```
from functions import Counter, DataFrame
# perseus dataframe
df = DataFrame([[k, v] for k, v in Counter(ccontent1).items()])
df[2] = df[1].apply(lambda x: round(x*100/chars1, 2))
a = df.sort_values(1, ascending=False)
# first1k dataframe
df = DataFrame([[k, v] for k, v in Counter(ccontent2).items()])
df[2] = df[1].apply(lambda x: round(x*100/chars2, 2))
b = df.sort_values(1, ascending=False)
# perseus + first1k dataframe
df = DataFrame([[k, v] for k, v in Counter(ccontent3).items()])
df[2] = df[1].apply(lambda x: round(x*100/chars3, 2))
c = df.sort_values(1, ascending=False)
```

The first column is the letter, the second column is the count of the letter, and the third column is the percentage of the letter contra all letters.

```
from functions import display_side_by_side
# show tables side by side to save some vertical space
display_side_by_side(Perseus=a, First1K=b, Perseus_First1K=c)
```

1.2 Table data

Perseus			FirstK1			Both		
Letter	Count	Percent	Letter	Count	Percent	Letter	Count	Percent
A	4182002	10.96	A	26817705	10.76	A	30999707	10.79
E	3678672	9.64	O	23687669	9.50	O	27351703	9.52
O	3664034	9.61	I	22665483	9.09	I	26279145	9.14
I	3613662	9.47	E	22498413	9.03	E	25909263	9.01
N	3410850	8.94	N	22121458	8.88	N	25800130	8.98
T	2903418	7.61	T	21698265	8.71	T	24601683	8.56
Σ	2830967	7.42	Σ	18738234	7.52	Σ	21569201	7.50
Υ	1776871	4.66	Υ	11384921	4.57	Υ	13161792	4.58
P	1440852	3.78	H	9776411	3.92	H	11217263	3.90
H	1392909	3.65	P	9268111	3.72	P	10661020	3.71
Π	1326596	3.48	K	8982955	3.60	K	10244628	3.56
K	1261673	3.31	Π	8290364	3.33	Π	9616960	3.35
Ω	1179566	3.09	Ω	7874161	3.16	Ω	9053727	3.15
M	1147548	3.01	M	7498489	3.01	M	1147548	3.01
Λ	1139510	2.99	Λ	6929170	2.78	Λ	8076718	2.81
Δ	932823	2.45	Δ	5757782	2.31	Δ	6690605	2.33
Γ	584668	1.53	Γ	4197053	1.68	Γ	4781721	1.66
Θ	501512	1.31	Θ	3440599	1.38	Θ	3942111	1.37
X	352579	0.92	X	2294905	0.92	X	2647484	0.92
Φ	325210	0.85	Φ	2115768	0.85	Φ	2440978	0.85
B	220267	0.58	B	1322737	0.53	B	1543004	0.54
Ξ	152971	0.40	Ξ	951076	0.38	Ξ	1104047	0.38
Z	75946	0.20	Z	559728	0.22	Z	635674	0.22
Ψ	51405	0.13	Ψ	375266	0.15	Ψ	426671	0.15
F	349	0.00	Ϝ	5162	0.00	Ϝ	5171	0.00
Ϝ	9	0.00	ϡ	259	0.00	F	505	0.00
ϡ	4	0.00	F	156	0.00	ϡ	263	0.00
	3 0	0.00 0.00	Ϙ	111 13	0.00 0.00	Ϙ	114 13	0.00 0.00

Greek corpora contains mathematical texts in Greek, which explains why the rarely used digamma (F/Ϝ = 6), qoppa (/Ϙ = 90), and sampi (ϡ = 900) letters are included on the table. You can find other interesting differences between *Perseus* and *First1k* corpora, like the occurrence of P/H, K/Π, and O/I/E which are probably explained by the difference of the included text genres in corpora.

1.3 Bar chart

The next chart will show visually which are the most used letters and the least used letters in the available Ancient Greek corpora.

Vowels with *N*, *S*, and *T* consonants pops up as the most used letters. The least used letters are *Z*, Ξ, and Ψ, if the exclusive numerals Ϛ, , and Ϟ are not counted.

1.4 Optional live chart

Uncomment the next part to output a new fresh graph from Plotly:

```
#import plotly
#plotly.offline.init_notebook_mode(connected=False)

# for the fist time set plotly service credentials, then you can comment
# next line
#plotly.tools.set_credentials_file(username='MarkoManninen', api_key='xyz')

# embed plotly graphs
#plotly.tools.embed("https://plot.ly/~MarkoManninen/8/")
```

1.4.1 Unique words database

Now it is time to collect unique Greek words to the database and show certain specialties of the word statistics. I'm reusing data from the *greek_corpora* variable that is in the memory already. Running the next code will take a minute or two depending on the processor speed of your computer:

```
from functions import syllabify, Abnum, greek, vowels
# greek abnum object for calculating isopsephical value of the words
g = Abnum(greek)
# count unique words statistic from the parsed greek corpora
# rather than the plain text file. it would be pretty hefty work to find
# out occurence of the all over 800000 unique words from the text file that
# is over 300 MB big!
unique_word_stats = {}
for item in greek_corpora:
```

```
    for word, cnt in item['uwords'].items():
        if word not in unique_word_stats:
            unique_word_stats[word] = 0
        unique_word_stats[word] += cnt
# init dataframe
df = DataFrame([[k, v] for k, v in unique_word_stats.items()])
# add column for the occurrence percentage of the word
# lwords3 variable is the length of the all words list
df[2] = df[1].apply(lambda x: round(x*100/lwords3, 2))
# add column for the length of the individual word
df[3] = df[0].apply(lambda x: len(x))
# add isopsephical value column
df[4] = df[0].apply(lambda x: g.value(x))
# add syllabified word column
df[5] = df[0].apply(lambda x: syllabify(x))
# add length of the syllables in word column
df[6] = df[5].apply(lambda x: len(x))
# count vowels in the word as a column
df[7] = df[0].apply(lambda x: sum(list(x.count(c) for c in vowels)))
# count consonants in the word as a column
df[8] = df[0].apply(lambda x: len(x)-sum(list(x.count(c) for c in vowels)))
```

Store database

This is the single most important part of the chapter. I'm saving all simplified unique words as a CSV file that can be used as a database for the riddle solver. After this you may proceed to the riddle solver Jupyter notebook document in interactive mode, if you prefer.

```
from functions import csv_file_name
# save dataframe to CSV file
df.to_csv(csv_file_name, header=False, index=False, encoding='utf-8')
```

Noteworth is that stored words are not stems or any base forms of the words but contain words in all possible inflected forms. Due to nature of machine processed texts, one should also be warned about corrupted words and other noise to occur in results. Programming tools are good for extracting interesting content and filtering data that would be impossible for a human to do because of its enormous size. But results still need verification and interpretation. Also, procedures can be fine tuned and developed in many ways.

Most repeated words

For a confirmation of the succesful task, I will show the total number of the unique words, and five of the most repeated words in the database:

```
# import display html helper function
from functions import display_html
# sort and limit words, select columns by index 1, 2, and 3
words = df.sort_values(1, ascending=False).head(n=5).iloc[:,0:3]
# label columns
words.columns = ['Word', 'Count', 'Percent']
# output total number of the words from df object
print("Total records: %s" % len(df))
# index=False to hide index column and output table by using to_html method
display_html(words.to_html(index=False), raw=True)
```

Total records: 833817

Word	Count	Percent
KAI	1781528	5.38
ΔE	778589	2.35
TO	670952	2.03
TΩN	487015	1.47
H	483372	1.46

KAI, the word denoting and-conjuction[20], is well known as the most repeated word in the Ancient Greek. Above statistics says that *KAI* word takes almost 5.4% of the all words.

This can be explained easily because *KAI* serves for many fundamental functions in text, such as an indicator of a new chapter or a paragraph, list copulative of two or more items, etc., basicly in a place, where we would use punctuation nowadays. From the other words, *H* stands for a paraphrase and ΔE for a disconjunction. All these three words characterises Ancient Greek as fundamentally based on logical constructors, one could argue. Maybe even early type of list processing structures have been developed in a form of natural language. It would be an interesting excurse to compare the propositional logic and the list processing features of the Ancient Greek rhetorics to the modern LISP language or similar programming paradigm, but that is definitely beyond the scope of the investigation of this study.

Naturally, articles and particles (*TO*, *TΩN*) belong to the most repeated words as well. One could use the knowledge of the certain word rate as one of the indicators of the text genre, or even quess the author of the text.

Longest words

For a curiosity, let's also see the longest words in the database:

```
from functions import HTML
# load result to the temporary variable for later usage
# sort by length, limit to 20 items
l = df.sort_values(3, ascending=False).head(n=20)
# take column index 0, 1, and 3. this is the second way of selecting
# certain columns. see iloc method in the previous example
l = l[[0, 1, 3]]
# label columns
l.columns = ['Word', 'Count', 'Length']
# output table without the index column
HTML(l.to_html(index=False))
```

A bit later I'm searching exact place of these words from the corpora, but lets first find out, what words have the biggest isopsephical value.

Biggest isopsephical value

So, which words have the biggest isopsephical value in the database? We can find it out by sorting words database by the fourth column, that is the isopsephical value of the word.

```
# sort by the isopsephy column and get the first 20 items
m = df.sort_values(4, ascending=False).head(n=20)
# select columns by indices
m = m[[0, 1, 4]]
# relabel selected columns
m.columns = ['Word', 'Count', 'Isopsephy']
# remove the index column and output table
HTML(m.to_html(index=False))
```

[20] https://github.com/cltk/cltk

These are very rare words, as was the case with the longest words too, but as it can be seen, the longest and the biggest isopsephical words are just partly overlapping. Isopsephical value of the word is not depending of the length of the word, but it is depending on the fact, how many times the latter part of the letters in the alphabet occus in the word. In ΛΕΟΝΤΑΤϒΦΛΩΣΩΝΣΚΩΛΩΨΔΕΤΟϒ letters *T*, Φ, Ω, and Σ are repeated several times so that the sum of the alphabetic numerals in the word, i.e. the isopsephical value, is 6865. The value gap between the first and the second word is rather big. Results like these are interesting because they may tell deliberate construction of the words, which I want to detect from the vast sample of coincidental hits.

Before going to the last useful procedure of spotting the location of the words, lets see a special statictic about the frequency of the words.

Word frequency

So, I already know that there are certain words repeating very often, for different reasons. But then there are words repeating once or few times only. Thus, it is relevant to ask, how many percent of the whole word base, the least repeated words actually take? For the task I'm using *groupby* and *count* methods of the *Dataframe* object in *Pandas*.

```
# length of the words database. taken to a variable to prevent unnecessary
# repeatition in the next for loop
le = len(df)
# group words by occurrence and count grouped items, list the first 10 items
for x, y in df.groupby([1, 2]).count()[:10].T.items():
    print("words repeating %s time(s): " % x[0], round(100*y[0]/le, 2), "%")
```

Output:

```
words repeating 1 time(s):  44.95 %
words repeating 2 time(s):  15.86 %
words repeating 3 time(s):  7.48 %
words repeating 4 time(s):  4.84 %
words repeating 5 time(s):  3.32 %
words repeating 6 time(s):  2.5 %
words repeating 7 time(s):  1.92 %
words repeating 8 time(s):  1.59 %
words repeating 9 time(s):  1.28 %
words repeating 10 time(s):  1.11 %
```

Almost 45% of the wodrds in database occurs only once in a corpora. That looks pretty high number which reason I have yet to resolved. Words that repeat 1-4 times fills roughly 70% of the whole corpora.

Detect source texts

Stats are nice, but it wouldn't be so useful, if there was no routine to find out words from corpora, where they actually occur.

The last part of the chapter one is to specify the procedure to find out the exact places of the given words in the corpora. This is going to be useful on the next chapters too. I have provided a *search_words_from_corpora* function to simplify this task. You may find the code from *functions.py* and alter it for your use.

1.5 Longest words

```
from functions import search_words_from_corpora
# I'm collecting the plain text words from the already instantiated l variable
```

```
words = list(y[0] for x, y in l.T.items())
search_words_from_corpora(words, [perseus_dir, first1k_dir])
```

Output:

```
+ Aristophanes, Lysistrata (tlg0019.tlg007.perseus-grc2.xml) =>

----- ΣΠΕΡΜΑΓΟΡΑΙΟΛΕΚΙΘΟΛΑΧΑΝΟΠΩΛΙΔΕΣ (1) -----
ὦ ξύμμαχοι γυναῖκες ἐκθεῖτ ἔνδοθεν ὦ σπερμαγοραιολεκιθολαχανοπώλιδες ὦ␣
↪σκοροδοπανδοκευτριαρτοπώλιδες

+ Aristophanes, Wasps (tlg0019.tlg004.perseus-grc1.xml) =>

----- ΟΡΘΡΟΦΟΙΤΟΣΥΚΟΦΑΝΤΟΔΙΚΟΤΑΛΑΙΠΩΡΩΝ (1) -----
ς ἀκούειν ἡδἔ εἰ καὶ νῦν ἐγὼ τὸν πατέρ ὅτι βούλομαι τούτων ἀπαλλαχθέντα τῶν␣
↪ὀρθροφοιτοσυκοφαντοδικοταλαιπώρων τρόπων ζῆν βίον γενναῖον ὥσπερ Μόρυχος αἰτίαν ἔχω ταῦτα␣
↪δρᾶν ξυνωμότης ὢν καὶ φρονῶν

+ Athenaeus, Deipnosophistae (tlg0008.tlg001.perseus-grc3.xml) =>

----- ΠΥΡΒΡΟΜΟΛΕΥΚΕΡΕΒΙΝΘΟΑΚΑΝΘΟΥΜΙΚΤΡΙΤΥΑΔΥ (1) -----
τις ἃ Ζανὸς καλέοντι τρώγματ ἔπειτ ἐπένειμεν ἐνκατακνακομιγὲς πεφρυγμένον␣
↪πυρβρομολευκερεβινθοακανθουμικτριτυαδυ βρῶμα τοπανταναμικτον ἀμπυκικηροιδηστίχας␣
↪παρεγίνετο τούτοις

+ Athenaeus, TheDeipnosophists (tlg0008.tlg001.perseus-grc4.xml) =>

----- ΠΥΡΟΒΡΟΜΟΛΕΥΚΕΡΕΒΙΝΘΟΑΚΑΝΘΙΔΟΜΙΚΡΙΤΡΙΑΔΥ (1) -----
ἐπεί γ ἐπένειμεν ἐγκατακνακομιγὲς πεφρυγμένον πυροβρομολευκερεβινθοακανθιδομικριτριαδυ␣
↪βρωματοπαντανάμικτον ἄμπυκι καριδίᾳ στιχὰς παρεγίνετο τούτοις σταιτινοκογχομαγὴς

+ Plato, Laws (tlg0059.tlg034.perseus-grc2.xml) =>

----- ΤΕΤΤΑΡΑΚΟΝΤΑΚΑΙΠΕΝΤΑΚΙΣΧΙΛΙΟΣΤΟΝ (1) -----
πεφευγότος ἀμφοτέρωθεν πρός τε ἀνδρῶν καὶ πρὸς γυναικῶν κληρονόμον εἰς τὸν οἶκον τοῦτον τῇ␣
↪πόλει τετταρακοντακαιπεντακισχιλιοστὸν καταστῆσαι βουλευομένους μετὰ νομοφυλάκων καὶ␣
↪ἱερέων διανοηθέντας τρόπῳ καὶ λόγῳ τοιῷδε ὡς οὐδεὶς

+ Plato, Republic (tlg0059.tlg030.perseus-grc2.xml) =>

----- ΕΝΝΕΑΚΑΙΕΙΚΟΣΙΚΑΙΕΠΤΑΚΟΣΙΟΠΛΑΣΙΑΚΙΣ (1) -----
τοῦ τυράννου ἀφεστηκότα λέγῃ ὅσον ἀφέστηκεν ἐννεακαιεικοσικαιεπτακοσιοπλασιάκις ἥδιον αὐτὸν␣
↪ζῶντα εὑρήσει τελειωθείσῃ τῇ πολλαπλασιώσει τὸν δὲ τύραννον ἀνιαρότερον τῇ αὐτῇ ταύτῃ

+ AlexanderOfAphrodisias, InAristotelisMetaphysicaCommentaria (tlg0732.tlg004.opp-
↪grc1.xml) =>

----- ΟΥΝΙΚΑΝΩΣΠΕΡΙΑΥΤΩΝΗΜΙΝΕΝΤΟΙΣΠΕΡΙ (1) -----
οιησά αενο τ ιστεύσομεν ρ Φ τεθεώρηται μὲν οὐνὶκανῶςπερὶαὐτῶνἡμῖνἐντοῖςπερὶ φύσεως ἱκαὶἱκανῶς␣
↪φησί περὶτῶ ν ἀρχῶν τῶν φυσικῶν ἐν τοῖς περὶ φύσεως

+ AlexanderOfAphrodisias, InAristotelisTopicorumLibrosOctoCommentaria (tlg0732.tlg006.
↪opp-grc1.xml) =>

----- ΟΤΙΤΟΥΜΗΔΙΑΠΡΟΤΕΡΩΝΟΡΙΖΕΣΘΑΙΤΡΕΙΣ (1) -----
Τοῦ δὲ μὴ ἐκπροτέρων τρεῖς εἰσι τρόποι Ὅτιτοῦμὴδιὰπροτέρωνὁρίζεσθαιτρεῖς εἰσι τρόποι πρῶτοςμὲν εἰ␣
↪διὰ τοῦ ἀντικειμένου τὸ ἀντικείμενον ὥρισται ἅμ γὰρ τῇ φύσει τὰ ἀντικείμ
```

```
+ ApolloniusDyscolus, DeAdverbiis (tlg0082.tlg002.1st1K-grc1.xml) =>

----- ΠΑΡΕΓΕΝΟΜΕΝΟΜΕΝΟΣΗΝΚΑΙΕΤΙΕΚΤΗΣΛΕΣΒΟΥΟΥΦΑΜΕΝ (1) -----
τῆ Λέβου τη εκ εκ Λεβο παρεγενόμην καὶ ἔτι οὗ φαμεν παρεγενομενομενοηνκαιετιεκτηλεβουουφαμεν␣
↪Α εκ τη Λεβου ἔτι οὐ

+ ApolloniusDyscolus, DeConstructione (tlg0082.tlg004.1st1K-grc1.xml) =>

----- ΚΑΙΤΟΝΑΡΙΣΤΑΡΧΟΝΑΣΜΕΝΩΣΤΗΝΓΡΑΦΗΝΤΟΥ (1) -----
ἠλογῆθαι φα δὲ καίτὸνἈρίταρχονἀμένωτὴνγραφὴντοῦ Δικαιάρχουπαραδέξαθαι ἐνγὰρἁπάαι ν τὸ εὲῆ ἐν␣
↪πατρίδι γαί ὑπολαβόντα τὸ ἑαυτῆ νοεὶθαι ἐκ το

----- ΑΡΣΕΝΙΚΩΝΟΝΟΜΑΤΩΝΣΤΟΙΧΕΙΑΕΣΤΙΠΕΝΤΕ (1) -----
τ τὸ ᾶ τελικόν ἐτιν κτλ Τελικὰ ἀρενικῶνὀνομάτωντοιχεῖἀὲτιπέντε θηλυκῶνδὲ ὀκτώ ᾶη ωνξΒ ψ οὐδετέ␣
↪ρων δὲ ὲ ῦ εραίαν

----- ΑΡΙΣΤΑΡΧΟΣΚΑΙΟΙΑΠΟΤΗΣΣΧΟΛΗΣΦΑΣΙΝ (1) -----
αὐτῇ θτή ει Β καθότ καθ ὁ Ἀρίταρχοκαὶοἰἀπὸτῆχολῆφαιν οἰ οὐ υγκαταθετέον ε φαίν οὐκ ὀρθῶ

+ Artemidorus, Onirocriticon (tlg0553.tlg001.1st1K-grc1.xml) =>

----- ΑΥΤΟΜΑΤΟΙΔΕΟΙΘΕΟΙΑΠΑΛΛΑΣΣΟΜΕΝΟΙ (1) -----
ς μεγάλας σημαίνει οἱ γὰρ ἐν μεγάλαις συμφοραῖς γενόμενοι καὶ τῆς πρὸς θεούς εὐσεβείας␣
↪ἀφίστανται αὐτόματοιδέοἱθεοὶἀπαλλασσόμενοι καὶ τὰ ἀγάλμιατα αὐτῶν συμπίπτοντα θάνατον τῷ␣
↪ἰδόντι ἤ τινι τῶν αὐτοῦ προαγορεύει θεο

+ JoannesPhiloponus, InAristotetelisMeteorologicorumLibrumPrimumCommentarium (tlg4015.
↪tlg005.opp-grc1.xml) =>

----- ΛΛΗΣΤΗΣΑΝΩΘΕΝΘΕΡΜΟΤΗΤΟΣΑΤΜΙΔΟΥΜΕΝΟΝΦΕΡΕΤΑΙ (1) -----
νῦν μενούσης ἀμεταβλήτου τὸ οὖν περὶ τὴν γῆν ὑγρόν φησίν ὑπὸ τῶν ἀκτίνων καὶ ὑπὸ τῆς ἀ␣
↪λληςτῆςἄνωθενθερμότητοςἀτμιδούμενονφέρεται ἄνω πῶς μὲν ἡ ἐκ τῶν ἀκτίνων γίνεται θερμότης␣
↪ἐδίδαξεν ὅτι ὁ ε ναπο λαμβαν

----- ΔΥΝΑΤΟΝΔΕΤΟΑΙΤΙΑΙΗΣΓΕΝΕΣΕΩΣΚΑΙΤΗΣΦΘΟΡΑΣ (1) -----
λὴ ἀνάλογόν ἐστι γενέσει ἡ δὲ τοῦμπαλιν τῶν κουφοτέρων εἰς τὰ βαρότεραφθορᾷ␣
↪δυνατὸνδὲτὸαἰτίαιῆςγενέσεωςκαὶτῆςφθορᾶς διὰ τὸ ἄρθρον μὴ καθολικῶς ἀκούειν πάσης γενέσεως καὶ␣
↪φθορᾶς ἀλλὰ ὑετοῦ χιόν

+ Libanius, Epistulae1-839 (tlg2200.tlg001.opp-grc1.xml) =>

----- ΕΜΟΥΟΙΑΠΕΦΕΥΓΑΧΕΙΡΑΣΛΥΠΗΣΑΣΜΕΝΟΥΔΕΝΑΟΥΔΕΝ (1) -----
δον κατηφῆ καὶ συνεοταλμὲνον καὶ δάκρυα πρὸ τῶν λόγωνἀφεὶς ἐγὼ μόλις τὰς τῶν παθόντων␣
↪ἐμοῦὁιαπέφευγαχεῖραςλυπήσαςμὲνοὐδέναοὐδέν ἡνίκα ἐξῆν μικρο δὲ διασπασθείς καὶ προσετίθει φυγὴν␣
↪ἀδελφοῦ καὶ γένους ὅλου πλάνην καὶ γῆν ἄσπ

----- ΚΑΙΙΚΕΛΗΧΡΥΣΗΑΦΡΟΔΙΤΗΚΑΙΟΙΣΕΚΟΣΜΗΣΕ (1) -----
ε γονεῦσιν αὐτῆς καὶ σοὶ συνη σθην τοῖς μέν οἕαν ἔφυσαν σοὶ δέ οἵαν ἔχεις Δήλῳ δή ποτε τοῖον␣
↪καὶἰκέληχρυσῆἈφροδίτηκαὶοἵςἐκόσμησε γυναῖκας Ὅμηρος πάντα ἂν δέξαιτο ἀναμιμν

----- ΚΑΝΤΩΝΕΠΙΤΑΙΣΔΥΝΑΜΕΣΙΠΑΡΑΒΑΙΝΗ (1) -----
ὅτι ὦ βασιλεῦ τῶν ἀδικούντων οὐδένα οὐόὲν ἀξίωμα ῥύσεται ἀλλὰ κἂν τῶν δικαζόντων τις␣
↪κἂντῶνἐπὶταὶςδυνάμεσιπαραβαίνη του ςνο μους οὐκἀνέζομαιἀμελεῖσθαι τα

+ Libanius, OratioI (tlg2200.tlg00401.opp-grc1.xml) =>

----- ΗΔΙΚΗΜΕΝΟΝΔΕΑΠΕΡΡΙΜΜΕΝΟΝΠΕΡΙΟΡΑΣ (1) -----
τέ τῶν μὲν ἐξέβαλες τὰ δὲοὐΙδίδως ἀλλ ὁ μὲν ἠπατηκὼς τρυφᾷ τὸν␣
↪ἠδικημένονδὲἀπερριμμένονπεριορᾶς τοι αυ τα με ν προ ς το ε δος πο ρ
```

```
+ Suda, SuidaeLexicon (tlg9010.tlg001.1st1K-grc1.xml) =>

----- ΟΡΘΟΦΟΙΤΟΣΥΚΟΦΑΝΤΟΔΙΚΟΤΑΛΑΙΠΩΡΩΝ (2) -----
Ὀρθοφοιτουκοφαντοδικοταλαιπώρων Ἀριτοφάνη ὁτιὴ βούλομαι τούτων ἀπαλλαχθέντα τῶν␣
↪ὀρθοφοιτουκοφα
οδικοταλαιπώρων Ἀριτοφάνη ὁτιὴ βούλομαι τούτων ἀπαλλαχθέντα τῶν␣
↪ὀρθοφοιτουκοφαντοδικοταλαιπώρων τρόπων ζῆν βίον γενναῖον ὥπερ Μόρυχο αἰτίαν ἔχων ταῦτα␣
↪δρᾶν

----- ΣΠΕΡΜΑΓΟΡΑΙΟΛΕΚΙΘΟΛΑΧΑΝΟΠΩΛΙΔΕΣ (1) -----
Ὦ περμαγοραιολεκιθολαχανοπώλιδε ὦ κοροδοπανδοκευτριαρτοπώλιδε οὐκ ἐξέλκετ οὐ παιήετ οὐκ
```

For a small explanation: Aristophanes was a Greek comic playwright and a word expert of a kind. Mathematical texts are also filled with long compoud words for fractions for example.

1.6 Highest isopsephy

```
# I'm collecting the plain text words from the already instantiated m variable
words = list(y[0] for x, y in m.T.items())
search_words_from_corpora(words, [perseus_dir, first1k_dir])
```

Output:

```
+ Appian, TheCivilWars (tlg0551.tlg017.perseus-grc2.xml) =>

----- ΣΥΝΥΠΟΧΩΡΟΥΝΤΩΝ (1) -----
καὶ ἡ σύνταξις ἤδη παρελέλυτο ὀξύτερον ὑπεχώρουν καί τῶν ἐπιτεταγμένων σφίσι
δευτέρων καὶ τρίτων συνυποχωρούντων μισγόμενοι πάντες ἀλλήλοις ἀκόσμως
ἐθλίβοντο ὑπὸ σφῶν καὶ τῶν πολεμίων ἀπαύστως αὐτοῖς ἐπικειμένων

+ Aristophanes, Wasps (tlg0019.tlg004.perseus-grc1.xml) =>

----- ΟΡΘΡΟΦΟΙΤΟΣΥΚΟΦΑΝΤΟΔΙΚΟΤΑΛΑΙΠΩΡΩΝ (1) -----
ς ἀκούειν ἡδέ εἰ καὶ νῦν ἐγὼ τὸν πατέρ ὅτι βούλομαι τούτων ἀπαλλαχθέντα τῶν
ὀρθροφοιτοσυκοφαντοδικοταλαιπώρων τρόπων ζῆν βίον γενναῖον ὥσπερ Μόρυχος
αἰτίαν ἔχω ταῦτα δρᾶν ξυνωμότης ὢν καὶ φρονῶν

+ Athenaeus, Deipnosophistae (tlg0008.tlg001.perseus-grc3.xml) =>

----- ΒΡΥΣΣΩΝΟΘΡΑΣΥΜΑΧΕΙΟΛΗΨΙΚΕΡΜΑΤΩΝ (1) -----
τῶν ἐξ Ἀκαδημίας τις ὑπὸ Πλάτωνα καὶ Βρυσωνοθρασυμαχειοληψικερμάτων πληγεὶς
ἀνάγκῃ ληψολιγομίσθῳ τέχνῃ σ

+ Athenaeus, TheDeipnosophists (tlg0008.tlg001.perseus-grc4.xml) =>

----- ΒΡΥΣΣΩΝΟΘΡΑΣΥΜΑΧΕΙΟΛΗΨΙΚΕΡΜΑΤΩΝ (1) -----
Βρυσωνοθρασυμαχειοληψικερμάτων πληγεὶς ἀνάγκῃ ληψιλογομίσθῳ τέχνῃ

+ AlexanderOfAphrodisias, InAristotelisMetaphysicaCommentaria (tlg0732.tlg004.opp-
↪grc1.xml) =>

----- ΟΥΝΙΚΑΝΩΣΠΕΡΙΑΥΤΩΝΗΜΙΝΕΝΤΟΙΣΠΕΡΙ (1) -----
οιησά αενο τ ιστεύσομεν ρ Φ τεθεώρηται μὲν οὐνὶκανῶςπερὶαὐτῶνἡμῖνἐντοῖςπερὶ
φύσεως ἱκαὶἱκανῶς φησί περὶτῶ ν ἀρχῶν τῶν φυσικῶν ἐν τοῖς περὶ φύσεως
```

```
+ ApolloniusDyscolus, DeConstructione (tlg0082.tlg004.1st1K-grc1.xml) =>

----- ΚΑΙΤΟΝΑΡΙΣΤΑΡΧΟΝΑΣΜΕΝΩΣΤΗΝΓΡΑΦΗΝΤΟΥ (1) -----
ἠλογῆθαι φα δὲ καίτὸνἈρίταρχονἀμένωτὴνγραφὴντοῦ Δικαιάρχουπαραδέξαθαι
ἐνγὰρἀπάαι ν τὸ εὲῆ ἐν πατρίδι γαί ὑπολαβόντα τὸ ἑαυτῆ νοεὶθαι ἐκ το

----- ΑΡΣΕΝΙΚΩΝΟΝΟΜΑΤΩΝΣΤΟΙΧΕΙΑΕΣΤΙΠΕΝΤΕ (1) -----
τ τὸ ᾶ τελικόν ἐτιν κτλ Τελικὰ ἀρενικῶνὀνομάτωντοιχεῖἀὲτιπέντε
θηλυκῶνδὲ ὀκτώ ᾶη ωνξΒ ψ οὐδετέ ρων δὲ ὲ ῦ εραίαν

----- ΑΡΙΣΤΑΡΧΟΣΚΑΙΟΙΑΠΟΤΗΣΣΧΟΛΗΣΦΑΣΙΝ (1) -----
αὐτῆ θτή ει Β καθότ καθ ὁ Ἀρίταρχοκαὶοίἀπὸτῆχολῆφαιν οὶ οὐ
υγκαταθετέον ϵ φαίν οὐκ ὀρθῶ

+ ApolloniusDyscolus, DePronominibus (tlg0082.tlg001.1st1K-grc1.xml) =>

----- ΩΡΙΣΜΕΝΩΝΠΡΟΣΩΠΩΝ (1) -----
ι καὶ τὰ ἀναφερύμενα γνῶιν ἐπαγγέλλεται προῦφετῶαν ὅ ἐτι πάλιν πρόωπον
ὡριμένον ὀρθῶ ἄρα ὡριμένωνπροώπων παρατατικὴ ἡ ἀντωνυμία

+ Aristotle, MagnaMoralia (tlg0086.tlg022.1st1K-grc1.xml) =>

----- ΤΩΟΡΘΩΕΚΑΣΤΑΘΕΩΡΩΝ (1) -----
καὶ μὴ διεψεῦσθαι τῷ λόγῳ ἔστιν δὲ καὶ ὁ φρόνιμός τοιοῦτος ὀτῷ λόγῳ
τῷὀρθῷἔκασταθεωρῶν πότερον δ ἐνδέχεταιτὸν φρόνιμον ἀκρατῆ εἶναι ἢ οὔ
ἀπορήσειε γὰρ ἄν τις τὰ εἰρημένα ἐὰν δὲ πα ρ

+ ChroniconPaschale, ChroniconPaschale (tlg2371.tlg001.opp-grc1.xml) =>

----- ΟΠΡΩΤΟΣΑΝΘΡΩΠΩΝΥΠΟΔΕΙΞΑΣ (1) -----
δείξας οὐρανοδρομεῖν ὁπρῶτοςἀνθρώπωνὑποδείξας ἀγγέλων καὶ ἀνθρώπων μίαν
ὁδόν ὁ τὴν γῆν λαχὼν οἰκητηριον καὶ τὸν οὐρανὸν

+ EvagriusScholasticus, HistoriaEcclesiastica (tlg2733.tlg001.1st1K-grc1.xml) =>

----- ΓΛΩΣΣΟΤΟΜΗΘΕΝΤΩΝΧΡΙΣΤΙΑΝΩΝ (1) -----
ιδ Περὶ Ὀνωρίχου τοῦ Βανδίλων ἄρχοντος καὶ τῶν γλωσσοτομηθέντωνΧριστιανῶν
παῤ αὐτοῦ ιε Περὶ Καβαώνου

----- ΕΠΙΣΚΟΠΩΚΩΝΣΤΑΝΤΙΝΟΥΠΟΛΕΩΣ (1) -----
ἐστιν ἐν τούτοις Ἐπιστολὴ ἤτοι δέησις ἀποσταλεῖσα Ἀκακίῳ
ἐπισκόπῳΚωνσταντινουπόλεως παρὰ τῶν τῆς Ἀσίας ἐπισκόπων Ἀκακίῳ τῷ ἁγιωτάτῳ
καὶ ὁσιωτάτῳ πατριάρχῃ

+ JoannesPhiloponus, InAristotetelisMeteorologicorumLibrumPrimumCommentarium (tlg4015.
↪tlg005.opp-grc1.xml) =>

----- ΛΛΗΣΤΗΣΑΝΩΘΕΝΘΕΡΜΟΤΗΤΟΣΑΤΜΙΔΟΥΜΕΝΟΝΦΕΡΕΤΑΙ (1) -----
νῦν μενούσης ἀμεταβλήτου τὸ οὖν περὶ τὴν γῆν ὑγρόν φησίν ὑπὸ τῶν ἀκτίνων καὶ
ὑπὸ τῆς ἀ λληστῆςἄνωθενθερμότητοςἀτμιδούμενονφέρεται ἄνω πῶς μὲν ἡ ἐκ τῶν
ἀκτίνων γίνεται θερμότης ἐδίδαξεν ὅτι ὁ ϵ ναπο λαμβαν

----- ΔΥΝΑΤΟΝΔΕΤΟΑΙΤΙΑΙΗΣΓΕΝΕΣΕΩΣΚΑΙΤΗΣΦΘΟΡΑΣ (1) -----
λὴ ἀνάλογόν ἐστι γενέσει ἡ δὲ τοὔμπαλιν τῶν κουφοτέρων εἰς τὰ βαρότεραφθορᾷ
δυνατὸνδὲτὸαἰτίαιῆςγενέσεωςκαὶτῆςφθορᾶς διὰ τὸ ἄρθρον μὴ καθολικῶς ἀκούειν
πάσης γενέσεως καὶ φθορᾶς ἀλλὰ ὑετοῦ χιόν
```

```
+ Libanius, Epistulae1-839 (tlg2200.tlg001.opp-grc1.xml) =>

----- ΕΜΟΥΟΙΑΠΕΦΕΥΓΑΧΕΙΡΑΣΛΥΠΗΣΑΣΜΕΝΟΥΔΕΝΑΟΥΔΕΝ (1) -----
δον κατηφῆ καὶ συνεοταλμὲνον καὶ δάκρυα πρὸ τῶν λόγωνἀφεὶς ἐγὼ μόλις τὰς
τῶν παθόντων ἐμοῦὁιαπέφευγαχεῖρασλυπήσασμὲνοὐδέναοὐδέν ἡνίκα ἐξῆν μικρο δὲ
διασπασθείς καὶ προσετίθει φυγὴν ἀδελφοῦ καὶ γένους ὅλου πλάνην καὶ γῆν ἄσπ

+ PhiloJudaeus, DeVitaMosisLibIIi (tlg0018.tlg022.opp-grc1.xml) =>

----- ΨΥΧΟΓΟΝΙΜΩΤΑΤΩΝ (1) -----
ν ἀπετελέσθησαν αἱ σωματικαὶ ποιότητες ἐφεὶς τῷ Μωυσέως ἀδελφῷ τὰς δ ἴσας
ἐξ ἀέρος καὶ πυρὸς τῶν ψυχογονιμωτάτων μόνῳ Μωυσεῖ μίαν δὲ κοινὴν ἀμφοτέροις
ἑβδόμην ἐπιτρέπει τρεῖς δὲ τὰς ἄλλας εἰς συμπ

+ Porphyrius, VitaPythagorae (tlg2034.tlg002.1st1K-grc1.xml) =>

----- ΤΟΥΤΟΥΣΛΕΓΟΝΤΕΣΩΣΠΡΟΣΤΗΝ (1) -----
οι τὰς δυνάμεις τῶν στοιχείων καὶ αὐτὰ ταῦτα βουλόμενοι παραδοῦναι
παρεγένοντο ἐπὶ τοὺςχαρακτῆρας τούτουςλέγοντεςὡςπρὸςτὴν πρώτην διδασκαλίαν
στοιχεῖα εἶναι ὕστερον μέντοι διδάσκου σιν ὅτι οὐχ οὗτοι στοιχεῖά εἰσιν οἱ
χαρ

+ Suda, SuidaeLexicon (tlg9010.tlg001.1st1K-grc1.xml) =>

----- ΟΡΘΟΦΟΙΤΟΣΥΚΟΦΑΝΤΟΔΙΚΟΤΑΛΑΙΠΩΡΩΝ (2) -----
Ὀρθοφοιτουκοφαντοδικοταλαιπώρων Ἀριτοφάνη ὁτιὴ βούλομαι τούτων
ἀπαλλαχθέντα τῶν ὀρθοφοιτουκοφα

οδικοταλαιπώρων Ἀριτοφάνη ὁτιὴ βούλομαι τούτων ἀπαλλαχθέντα τῶν
ὀρθοφοιτουκοφαντοδικοταλαιπώρων τρόπων ζῆν βίον γενναῖον ὥπερ Μόρυχο
αἰτίαν ἔχων ταῦτα δρᾶν

----- ΚΩΔΩΝΟΦΑΛΑΡΑΧΡΩΜΕΝΟΥΣ (1) -----
μετήνεκται οὕτω ψοφοῦντα ψοφοῦντε Κωδωνοφαλαραχρωμένου αὐτὰ Κώδων
Σοφοκλῆ Τυρρηνικῆ

+ ValeriusBabrius, FabulaeAesopeae (tlg0614.tlg001.1st1K-grc2.xml) =>

----- ΛΕΟΝΤΑΤΥΦΛΩΣΩΝΣΚΩΛΩΨΔΕΤΟΥ (1) -----
τι ποιήσω καὶ εἰπὼν ἐπέβαλε τοιχοδεχειρασεπεβαλετον λεοντατυφλωσωνσκωλωψδετου
τωυπονυχα υποδυνα κεκαδαιμωσδουστη σαρκοσεισδυσησηνυσε θ ποιων
```

So, that's all for the Greek corpora processing and basic statistics. One could further investigate, categorize, and compare individual texts, but for me it is time to jump to the second big task, that is defining procedures for the riddle solver.

CHAPTER 2

Isopsephical riddles in Greek Pseudo Sibylline hexameter poetry

2.1 Introduction

There are dozens of alphanumeric riddles in the Pseudo Sibylline books. Major part of them are simple, only referring to the first letter of the person, and the number of that letter. In Greek alphabet, letters happens to have a numerical value also. I'm calling this letter value substitution system with a name *isopsephy* simply because it was a name known for the Ancient Greeks. Nowadays, this literary device is maybe a bit too much mystified and known mostly by its Hebrew counterpart _gematria_. Although isopsephy was used for divination also, one should also consider it as a device used by poets and experts in different literary genres. Those people wanted to excel in the art they practiced. Isopsephy, along with the dactylic hexameter, affected to the external structure of the text. Both were concrete devices in the toolbox of the masters of the written text and rhetorics.

The most of the riddles in Sibylline Oracles are actually too simple to be solved by computer algorithms only. There are not enough parameters for processing, thus all the words meeting simple criteria are far too many for any sensical analysis. Take for example Sibyl Book I, verses 51-60:

> Then a great destroyer of pious men shall come, whom seven times ten shall point out clearly. But from him a son, whom the first letter of three hundred proves, shall take the power. And after him shall be a ruler, of the initial sign of four, a life-destroyer. Then a reverend man of the number fifty. Next, succeeding him Who has the first mark of the initial sign three hundred, shall a Celtic mountaineer.

In the footnotes of "The Sibylline Oracles" by Milton S. Terry, page 41 we find that *seven times ten* refers to the letter O (omicron), which numerical value is 70. Ouespasianos is the person refered here, because just before the verse 51 oracle talks about the three kings Galba, Otho, and Vitelius who shortly reigned in the Roman Empire in 69AD. The next emperor in Rome was the son of the former, namely Titus, which first letter T is equal to 300 in Greek numerals. Continuation of the puzzle is made clear with the next three emperors: Domitian (_initial sign of **four**_ meaning simply D = 4), Nerva (_reverend man of the number **fifty**_, N = 50), and lastly Trajan (_initial sign three **hundred**_, T = 300).

Similar puzzles can be found from:

- I : 30, 388
- III : 28
- V : 18, 118, 149, 253, 274, 321, 338, 351

Fig. 2.1: Domenichino's Cumaean Sibyl, Musei Capitolini

- VIII : 48, 88, 127, 233, 306, 321
- XII : 21, 38, 73, 122, 313

It is interesting that we can find a solution for the ealier kingdoms and emperors in the riddles, but we have somewhat lost the context on the younger riddles from 200AD to 600AD.

From the more than 800 000 words, the words starting with any of the 24 Greek letters would yield tens of thousands of hits. I don't have a categorization / morphology system for the words so that I could filter proper nouns. If there was a way to find out names from the words database, then task might be reasonable.

On the other hand, as we can see, most of these riddles conserning persons, cities, and Gods names can be taken as solved already. Context of the riddle is revealing a name behind the puzzle. It helps that the plot of each book is more or less chronologially structured. Sometimes names are already mentioned near by the riddle, like Rome in the [Riddle 3](#Riddle-3). In these samples it is obscure, why riddles were used in first hand. It is possible that texts were edited later to add more information, fulfil predictions, correct and manipulate the original ones to prove divine inspiration.

But then there are couple of other riddles very special in nature. They contain more information about letters, syllables, vowels, consonants, and the total value of the name or the title. One of them, the Riddle 1, is regarded as unsolved yet so far. [Riddle 2](#Riddle-2) is solved, but works as a good reference and check point for the first one. Riddle 3 is here for an example as a more difficult puzzle to solve programmically, nevertheless solved by contextual analysis.

I'll take these three riddles under the closer investigation in this study:

2.1.1 Riddle 1

Sibyl Book 1 lines 137 - 146:

> 137 εἰμὶ δ᾽ ἔγωγε ὁ ὤν, σὺ δ᾽ ἐνί φρεσὶ σῇσι νόησον· 138 οὐρανὸν ἐνδέδυμαι, περιβέβλημαι δὲ θάλασσαν, 139 γαῖα δέ μοι στήριγμα ποδῶν, περὶ σῶμα κέχυται 140 ἀὴρ ἠδ᾽ ἄστρων με χορὸς περιδέδρομε πάντη. 141 ἐννέα γράμματ᾽ ἔχω· τετρασύλλαβός εἰμι· νόει με· 142 αἱ τρεῖς αἱ πρῶται δύο γράμματ᾽ ἔχουσιν ἑκάστη, 143 ἡ λοιπὴ δὲ τὰ λοιπὰ καί εἰσιν ἄφωνα τὰ πέντε· 144 τοῦ παντὸς δ᾽ ἀριθμοῦ ἑκατοντάδες εἰσὶ δὶς ὀκτώ 145 καὶ τρεῖς, τρὶς δεκάδες σύν γ᾽ ἑπτά. γνοὺς δὲ τίς εἰμι 146 οὐκ ἀμύητος ἔσῃ τῆς παρ᾽ ἐμοὶ σοφίης.

Translation:

> And I am He who is, and in thy heart Do thou discern. I clothe me with the heaven, And cast the sea around me, and for me Earth is a footstool, and the air is poured Around my body; and on every side Around me runs the chorus of the stars. Nine letters have I; of four syllables I am; discern me. The first three have each Two letters, the remaining one the rest, And five are mates; and of the entire sum The hundreds are twice eight and thrice three tens Along with seven. Now, knowing who I am, Be thou not uninitiate in my lore.

2.1.2 Riddle 2

Sibyl Book 1 lines 324-402:

> 324 δὴ τότε καὶ μεγάλοιο θεοῦ παῖς ἀνθρώποισιν 325 ἥξει σαρκοφόρος θνητοῖς ὁμοιούμενος ἐν γῇ, 326 τέσσαρα φωνήεντα φέρων, τὸ δ᾽ ἄφωνον ἐν αὐτῷ 327 δισσόν· ἐγὼ δέ κέ τοι ἀριθμόν γ᾽ ὅλον ἐξονομήνω· 328 ὀκτὼ γὰρ μονάδας, τόσσας δεκάδας δ᾽ ἐπὶ ταύταις 329 ἠδ᾽ ἑκατοντάδας ὀκτὼ ἀπιστοκόροις ἀνθρώποις 330 οὔνομα δηλώσει· σὺ δ᾽ ἐνὶ φρεσὶ σῇσι νόησον 331 ἀθανάτοιο θεοῦ Χριστὸν παῖδ᾽ ὑψίστοιο.

Translation:

> Then also shall a child of the great God Come, clothed in flesh, to men, and fashioned like To mortals in the earth; and he doth hear Four vowels, and two consonants in him Are twice announced; the whole

sum I will name: For eight ones, and as many tens on these, And yet eight hundred will reveal the name To men insatiate; and do thou discern In thine own understanding that the Christ Is child of the immortal God most high.

2.1.3 Riddle 3

Sibyl Book 8 lines 143-150:

143 ὤλετο γὰρ Ῥώμης ἀρχὴ τότε τηλεθόωσα, 144 ἀρχαίη πολέεσσι περικτιόνεσσιν ἄνασσα. 145 οὐκέτι νικήσειε πέδον Ῥώμης ἐριθήλου, 146 ὁππόταν ἐξ Ἀσίης κρατέων ἔλθῃ σὺν Ἄρηι. 147 ταῦτα δὲ πάντ᾽ ἔρξας ἥξει κρηπισθὲν ἐς ἄστυ. 148 τρὶς δὲ τριηκοσίους καὶ τεσσαράκοντα καὶ ὀκτώ 149 πληρώσεις λυκάβαντας, ὅταν σοι δύσμορος ἥξῃ 150 μοῖρα βιαζομένη τεὸν οὔνομα πληρώσασα.

Translation:

For Rome's power perished then while in its bloom an ancient queen with cities dwelling round. No longer shall the land of fertile Rome prevail, when out of Asia one shall come to rule with Ares. And when he has wrought all these things, to the city afterwards shall he come. And three times three hundred and eight and forty shalt thou make complete, when, taking thee by force, an ill-starred fate shall come upon thee and complete thy name.

Using Greek Text Corpora

I have used the next notebook to prepare the Greek text corpora and unique words database for the solver: Processing Greek corpora for the isopsehical riddle solver.ipynb. You need to run it once or have unique words csv file in the same directory than this notebook if you wish to run cells interactively.

Setup of the system I'm using is:

`` `python import sys sys.version ` ``

'3.5.4 | Continuum Analytics, Inc.| (default, Aug 14 2017, 13:41:13) [MSC v.1900 64 bit (AMD64)]'

Confirm unique words database is available and get the size of it:

```python from functions import get_file_size, csv_file_name

print("Size of the unique words database: %s MB" % get_file_size(csv_file_name)) ```

Size of the unique words database: 57.14 MB

## Import database

Using *Pandas* library I will read and import csv file that contains all preprocessed unique greek words collected earlier. Constructed *Pandas DataFrame* is a nice data container to handle tabular data.

```python # read unique words stats if available try:

from pandas import read_csv df = read_csv(csv_file_name, header = None) # convert data types for columns. 0 = word # 1: how many times word occurs in texts df[1] = df[1].apply(lambda x: int(x)) # 2: percentage of all words df[2] = df[2].apply(lambda x: float(x)) # 3: how many characters in the word df[3] = df[3].apply(lambda x: int(x)) # 4: isopsephical value of the word df[4] = df[4].apply(lambda x: int(x)) # 5: word split to syllables df[5] = df[5].apply(lambda x: str(x).replace("'", "").replace("[", "").replace("]", "").split(", ")) # 6: how many syllables df[6] = df[6].apply(lambda x: int(x))

**except Exception as e:** print(e) print("Could not find unique words database. Please follow the procedure explained in: Processing Greek corpora for the isopsehical riddle solver.ipynb")

```

To confirm succesful import I will show the first 20 most common words:
```

`python print("Total records:  %s" % len(df)) # get the most repeated words by sort asc, head 20 df.sort_values(1, ascending=False).head(n=20) `

Total records: 826516

0	1	2	3	4	5	6
38	ΚΑΙ	3332509	45.51	3	31	[ΚΑΙ]	1
25	ΔΕ	1355091	18.51	2	9	[ΔΕ]	1
309	ΤΟ	1297764	17.72	2	370	[ΤΟ]	1
46	ΤΟΥ	933432	12.75	3	770	[ΤΟΥ]	1
2	ΤΩΝ	918946	12.55	3	1150	[ΤΩΝ]	1

Solve the riddles

Riddle 1

> Nine letters have I; of four syllables I am; discern me. The first three have each two letters, the remaining one the rest, and five are mates; and of the entire sum the hundreds are twice eight and thrice three tens along with seven.

Using the next parameters from the riddle, lets try to solve it by the brute computational force:

- knowing the length of the word: 9
- knowing other details of the syllables of the word
- knowing the count of the consonants of the word: 5
- knowing the isopsephical value: 1697
- comparing the context of the result

Isopsephy and letter count filter

I will apply filters to the words database step by step narrowing the result.

So lets see first, how many words there are with the isopsephical value 1697 and letters counting nine?

“‘python from functions import HTML

make a copy of the database to keep original safe a = df.copy() # filter by isopsephical value a = a[a[4] == 1697] # filter by word length a = a[a[3] == 9]

print(“Total records: %s” % len(a)) HTML(a.to_html(index=False)) “‘

Total records: 15

0	1	2	3	4	5	6
ΑΜΦΕΚΑΛΥΨ	1	0.0	9	1697	[ΑΜ, ΦΕ, ΚΑ, ΛΥΨ]	4
ΑΛΕΙΦΑΤΩΝ	2	0.0	9	1697	[Α, ΛΕΙ, ΦΑ, ΤΩΝ]	4
ΒΕΛΤΙΣΤΩΝ	66	0.0	9	1697	[ΒΕΛ, ΤΙ, ΣΤΩΝ]	3
ΑΦΕΛΩΝΤΑΙ	20	0.0	9	1697	[Α, ΦΕ, ΛΩΝ, ΤΑΙ]	4
ΠΑΡΑΧΩΡΕΙ	130	0.0	9	1697	[ΠΑ, ΡΑ, ΧΩ, ΡΕΙ]	4
ΚΥΒΕΥΣΟΥΣ	1	0.0	9	1697	[ΚΥ, ΒΕΥ, ΣΟΥΣ]	3
ΛΙΒΥΣΣΕΩΝ	2	0.0	9	1697	[ΛΙ, ΒΥΣ, ΣΕ, ΩΝ]	4

ΙΣΤΟΡΗΣΘΩ | 9 | 0.0 | 9 | 1697 | [I, ΣΤΟ, PH, ΣΘΩ] | 4 |
ΣΥΝΩΚΙΣΘΗ | 19 | 0.0 | 9 | 1697 | [ΣΥ, NΩ, KI, ΣΘH] | 4 |
ΠΛΗΡΟΥΣΘΩ | 7 | 0.0 | 9 | 1697 | [ΠΛΗ, ΡΟΥ, ΣΘΩ] | 3 |
ΕΚΑΑΤΟΣΤΩ | 3 | 0.0 | 9 | 1697 | [Ε, ΚΑ, Α, ΤΟ, ΣΤΩ] | 5 |
ΛΗΛΥΘΟΤΩΝ | 3 | 0.0 | 9 | 1697 | [ΛΗ, ΛΥ, ΘΟ, ΤΩΝ] | 4 |
ΑΝΑΝΨΩΜΕΝ | 4 | 0.0 | 9 | 1697 | [Α, ΝΑΝ, ΨΩ, ΜΕΝ] | 4 |
ΑΛΕΦΩΝΤΑΙ | 2 | 0.0 | 9 | 1697 | [Α, ΛΕ, ΦΩΝ, ΤΑΙ] | 4 |
ΣΤΕΝΩΜΑΤΑ | 8 | 0.0 | 9 | 1697 | [ΣΤΕ, NΩ, ΜΑ, ΤΑ] | 4 |

It turns out that there are just very few words meeting the criteria of the riddle. I could already make the analysis of the words manually. But to make everything reusable for later usage, I will set up filter procedure for other criteria too.

Before that, I will however add one extension to the original search and allow the count of the letters to be between 8 and 10. That is due to double consonant and long / short vowelspecialty of the Greek language. Let's see the result of this filter variation:

‘‘‘python b = df.copy() b = b[b[4] == 1697] b = b[b[6] == 4] b = b[b[3] > 7] b = b[b[3] < 11]

print("Total records: %s" % len(b)) HTML(b.sort_values(3, ascending=False).to_html(index=False)) ‘‘‘

Total records: 31

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΚΑΤΑΣΧΕΤΟΣ | 17 | 0.0 | 10 | 1697 | [ΚΑ, ΤΑ, ΣΧΕ, ΤΟΣ] | 4 |
ΝΕΒΡΟΦΟΝΩΝ | 1 | 0.0 | 10 | 1697 | [ΝΕ, ΒΡΟ, ΦΟ, ΝΩΝ] | 4 |
ΑΝΕΣΤΛΩΤΑΙ | 3 | 0.0 | 10 | 1697 | [Α, ΝΕΣΤ, ΛΩ, ΤΑΙ] | 4 |
ΕΜΦΥΣΣΑΝΤΑ | 9 | 0.0 | 10 | 1697 | [ΕΜ, ΦΥΣ, ΣΑΝ, ΤΑ] | 4 |
ΑΚΑΤΣΧΕΤΟΣ | 27 | 0.0 | 10 | 1697 | [Α, ΚΑΤ, ΣΧΕ, ΤΟΣ] | 4 |
ΕΠΤΑΠΛΑΣΩΣ | 18 | 0.0 | 10 | 1697 | [Ε, ΠΤΑ, ΠΛΑ, ΣΩΣ] | 4 |
ΠΑΡΑΙΤΣΕΩΣ | 25 | 0.0 | 10 | 1697 | [ΠΑ, ΡΑΙΤ, ΣΕ, ΩΣ] | 4 |
ΕΥΗΘΕΣΤΤΟΥ | 4 | 0.0 | 10 | 1697 | [ΕΥ, Η, ΘΕΣΤ, ΤΟΥ] | 4 |
ΥΦΙΖΝΟΥΣΙΝ | 3 | 0.0 | 10 | 1697 | [Υ, ΦΙΖ, ΝΟΥ, ΣΙΝ] | 4 |
ΚΑΤΑΨΥΞΕΙΣ | 9 | 0.0 | 10 | 1697 | [ΚΑ, ΤΑ, ΨΥ, ΞΕΙΣ] | 4 |
ΠΑΡΕΣΤΩΣΑΙ | 2 | 0.0 | 10 | 1697 | [ΠΑ, ΡΕ, ΣΤΩ, ΣΑΙ] | 4 |
ΔΙΕΣΤΗΚΤΩΝ | 92 | 0.0 | 10 | 1697 | [ΔΙ, Ε, ΣΤΗ, ΚΤΩΝ] | 4 |
ΠΑΧΥΤΕΡΑΙΣ | 8 | 0.0 | 10 | 1697 | [ΠΑ, ΧΥ, ΤΕ, ΡΑΙΣ] | 4 |
ΡΑΒΔΟΦΟΡΩΝ | 3 | 0.0 | 10 | 1697 | [ΡΑ, ΒΔΟ, ΦΟ, ΡΩΝ] | 4 |
ΛΙΒΥΣΣΕΩΝ | 2 | 0.0 | 9 | 1697 | [ΛΙ, ΒΥΣ, ΣΕ, ΩΝ] | 4 |
ΣΤΕΝΩΜΑΤΑ | 8 | 0.0 | 9 | 1697 | [ΣΤΕ, NΩ, ΜΑ, ΤΑ] | 4 |
ΑΛΕΦΩΝΤΑΙ | 2 | 0.0 | 9 | 1697 | [Α, ΛΕ, ΦΩΝ, ΤΑΙ] | 4 |
ΑΝΑΝΨΩΜΕΝ | 4 | 0.0 | 9 | 1697 | [Α, ΝΑΝ, ΨΩ, ΜΕΝ] | 4 |
ΛΗΛΥΘΟΤΩΝ | 3 | 0.0 | 9 | 1697 | [ΛΗ, ΛΥ, ΘΟ, ΤΩΝ] | 4 |
ΑΛΕΙΦΑΤΩΝ | 2 | 0.0 | 9 | 1697 | [Α, ΛΕΙ, ΦΑ, ΤΩΝ] | 4 |
ΑΦΕΛΩΝΤΑΙ | 20 | 0.0 | 9 | 1697 | [Α, ΦΕ, ΛΩΝ, ΤΑΙ] | 4 |
ΙΣΤΟΡΗΣΘΩ | 9 | 0.0 | 9 | 1697 | [Ι, ΣΤΟ, PH, ΣΘΩ] | 4 |
ΑΜΦΕΚΑΛΥΨ | 1 | 0.0 | 9 | 1697 | [ΑΜ, ΦΕ, ΚΑ, ΛΥΨ] | 4 |
ΣΥΝΩΚΙΣΘΗ | 19 | 0.0 | 9 | 1697 | [ΣΥ, NΩ, KI, ΣΘH] | 4 |

ΠΑΡΑΧΩΡΕΙ | 130 | 0.0 | 9 | 1697 | [ΠΑ, ΡΑ, ΧΩ, ΡΕΙ] | 4 |
ΧΩΣΑΜΕΝΑ | 1 | 0.0 | 8 | 1697 | [ΧΩ, ΣΑ, ΜΕ, ΝΑ] | 4 |
ΑΝΕΜΦΑΤΩ | 1 | 0.0 | 8 | 1697 | [Α, ΝΕΜ, ΦΑ, ΤΩ] | 4 |
ΦΟΒΗΘΗΤΩ | 5 | 0.0 | 8 | 1697 | [ΦΟ, ΒΗ, ΘΗ, ΤΩ] | 4 |
ΗΝΩΧΛΗΣΑ | 1 | 0.0 | 8 | 1697 | [Η, ΝΩ, ΧΛΗ, ΣΑ] | 4 |
ΠΥΡΕΤΩΔΗ | 6 | 0.0 | 8 | 1697 | [ΠΥ, ΡΕ, ΤΩ, ΔΗ] | 4 |
ΦΩΤΟΕΙΔΗ | 28 | 0.0 | 8 | 1697 | [ΦΩ, ΤΟ, ΕΙ, ΔΗ] | 4 |

But this was just for the experiment. I will stick on the more strict parameters in the following riddle solver.

Custom syllable filter

There is still two other criterias for the word filter. One of them is a bit more complicated. Poem says that there are two letters in the first three syllables. And the rest of the letters, that is three, are in the last syllable. I don't need to specify the last syllable letter count because I already limit total letter count to nine. If the first three syllables contain two letters, that is six in total, then the last must have the rest of the three letters.

The fifth column has appropriate syllable information that I can use for this kind of filter. Let's see the result with this and all the previous filters:

‘‘‘python c = df.copy() c = c[c[4] == 1697] c = c[c[6] == 4] c = c[c[3] == 9] # the first three of the syllable contain two letters, the last one the rest i.e. three. c = c[c.apply(lambda x: len(x[5][0]) == 2 and len(x[5][1]) == 2 and len(x[5][2]) == 2, axis=1)]

print("Total records: %s" % len(c)) HTML(c.sort_values(0).to_html(index=False)) ‘‘‘

Total records: 4

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΜΦΕΚΑΛΥΨ | 1 | 0.0 | 9 | 1697 | [ΑΜ, ΦΕ, ΚΑ, ΛΥΨ] | 4 |
ΛΗΛΥΘΟΤΩΝ | 3 | 0.0 | 9 | 1697 | [ΛΗ, ΛΥ, ΘΟ, ΤΩΝ] | 4 |
ΠΑΡΑΧΩΡΕΙ | 130 | 0.0 | 9 | 1697 | [ΠΑ, ΡΑ, ΧΩ, ΡΕΙ] | 4 |
ΣΥΝΩΚΙΣΘΗ | 19 | 0.0 | 9 | 1697 | [ΣΥ, ΝΩ, ΚΙ, ΣΘΗ] | 4 |

That is a pretty narrow result already, just handful items to analyse.

Consonant filter

Finally there is the rule of five consonants (mutes/males) in the word in the original riddle. That requires defining the consonants list and checking that the total count of the consonants is exactly five, no more, no less. I will do an exercise to filter all words having 4 syllables and 5 consonants.

‘‘‘python d = df.copy() d = d[d[4] == 1697] d = d[d[6] == 4] d = d[d[3] == 9] d = d[d.apply(lambda x: sum(list(x[0].count(c) for c in "ΨΖΞΒΦΧΘΓΔΜΛΚΠΡΣΤ")) == 5, axis=1)]

print("Total records: %s" % len(d)) HTML(d.sort_values(0).to_html(index=False)) ‘‘‘

Total records: 2

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΜΦΕΚΑΛΥΨ | 1 | 0.0 | 9 | 1697 | [ΑΜ, ΦΕ, ΚΑ, ΛΥΨ] | 4 |

ΙΣΤΟΡΗΣΘΩ | 9 | 0.0 | 9 | 1697 | [I, ΣΤΟ, ΡΗ, ΣΘΩ] | 4 |

Let's refactor all of this and the previous ones to the single callable function with reusable sub functions and apply it to dataframe

“‘python # the word should have n mutes ie consonants consonants = “ΨΖΞΒΦΧΘΓΔΜΛΚΠΡΣΤΝϹϡ” def nmutes(x, n):

word, tot = x[0], 0 for c in consonants:

tot += word.count(c) if tot > n:

return False

return tot == n

the word should have n vowels vowels = “ΩΗΥΕΙΟΑ” def nvowels(x, n):

word, tot = x[0], 0 for c in vowels:

tot += word.count(c) if tot > n:

return False

return tot == n

the word should have n syllables def nsyllables(x, n):

return x[6] == n

the word should have two letters in the first three syllables, and the rest (3) letters in the last def has_two_letters_in_first_three_syllables(x):

return len(x[5][0]) == 2 and len(x[5][1]) == 2 and len(x[5][2]) == 2

this makes n letters in total def nletters(x, n):

return x[3] == n

isopsephical value def nisopsephy(x, n):

return x[4] > n[0] and x[4] < n[1] if type(n) is list else x[4] == n

riddle 1 wrapper function def riddle1(x, isopsephy, letters = 9, mutes = 5, syllables = 4):

return nisopsephy(x, isopsephy) and nsyllables(x, syllables) and nletters(x, letters) and nmutes(x, mutes) and has_two_letters_in_first_three_syllables(x)

“‘

```
`python # solve the riddle 1a e = df.copy() e = e[e.apply(lambda x:
riddle1(x, 1697), axis=1)] HTML(e.to_html(index=False)) `
```

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΜΦΕΚΑΛΥΨ | 1 | 0.0 | 9 | 1697 | [ΑΜ, ΦΕ, ΚΑ, ΛΥΨ] | 4 |
ΣΥΝΩΚΙΣΘΗ | 19 | 0.0 | 9 | 1697 | [ΣΥ, ΝΩ, ΚΙ, ΣΘΗ] | 4 |
ΛΗΛΥΘΟΤΩΝ | 3 | 0.0 | 9 | 1697 | [ΛΗ, ΛΥ, ΘΟ, ΤΩΝ] | 4 |

Thus we have found three good matches for the riddle: ΑΜΦΕΚΑΛΥΨ, ΣΥΝΩΚΙΣΘΗ, and ΛΗΛΥΘΟΤΩΝ. From these, the word especially interesting is:

ΑΜΦΕΚΑΛΥΨ

(amphekalyps / amfecalyps) meaning “covering from both sides” or “all around covering”.

Next we should make some text and linquistic examination, how well these proposed words fits to the immediate context of the sibylline verses. Where are the exact occurrences of the word in the Greek corpora, in which context? Is it a word suitable for an epithet, does it have any religious spiritual significance, and so forth. These questions I will leave for the [further study](Study of the results of the isopsephical riddle solver.ipynb) of the words.

From the references of the riddle in the ancient greek and latin texts, there are some other interpretations of the isopsephic value given in the text. Instead of reading it 1697, it could be 1696, 1692, 1937, 1496, or even 506. Having factored procedure in a function, that takes isopsephic value as a parameter, I can make searches altering the value and see the results for comparison and inspection.

Variation 1696

‘‘‘python # solve the riddle 1b f = df.copy() f = f[f.apply(lambda x: riddle1(x, 1696), axis=1)]

print(“Total records: %s” % len(f)) HTML(f.sort_values(0).to_html(index=False)) ‘‘‘

Total records: 5

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΜΠΕΧΟΝΩΝ | 3 | 0.0 | 9 | 1696 | [ΑΜ, ΠΕ, ΧΟ, ΝΩΝ] | 4 |
ΔΥΝΑΜΩΣΑΣ | 4 | 0.0 | 9 | 1696 | [ΔΥ, ΝΑ, ΜΩ, ΣΑΣ] | 4 |
ΚΑΤΕΡΥΚΩΝ | 1 | 0.0 | 9 | 1696 | [ΚΑ, ΤΕ, ΡΥ, ΚΩΝ] | 4 |
ΚΕΝΟΦΩΝΑΣ | 47 | 0.0 | 9 | 1696 | [ΚΕ, ΝΟ, ΦΩ, ΝΑΣ] | 4 |
ΠΕΡΙΧΑΝΩΝ | 7 | 0.0 | 9 | 1696 | [ΠΕ, ΡΙ, ΧΑ, ΝΩΝ] | 4 |

Variation 1692

‘‘‘python # solve the riddle 1b f = df.copy() f = f[f.apply(lambda x: riddle1(x, 1692), axis=1)]

print(“Total records: %s” % len(f)) HTML(f.sort_values(0).to_html(index=False)) ‘‘‘

Total records: 4

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΝΤΩΝΥΜΑΝ | 22 | 0.0 | 9 | 1692 | [ΑΝ, ΤΩ, ΝΥ, ΜΑΝ] | 4 |
ΚΑΤΑΨΥΞΙΣ | 22 | 0.0 | 9 | 1692 | [ΚΑ, ΤΑ, ΨΥ, ΞΙΣ] | 4 |
ΠΑΡΩΞΥΝΑΣ | 9 | 0.0 | 9 | 1692 | [ΠΑ, ΡΩ, ΞΥ, ΝΑΣ] | 4 |
ΠΟΛΥΔΩΡΗΣ | 4 | 0.0 | 9 | 1692 | [ΠΟ, ΛΥ, ΔΩ, ΡΗΣ] | 4 |

Variation 1937

‘‘‘python # solve the riddle 1b f = df.copy() f = f[f.apply(lambda x: riddle1(x, 1937), axis=1)]

print(“Total records: %s” % len(f)) HTML(f.sort_values(0).to_html(index=False)) ‘‘‘

Total records: 1

0 | 1 | 2 | 3 | 4 | 5 | 6 |

— | — | — | — | — | — | — | — |
ΔΗΛΩΣΩΜΕΝ | 1 | 0.0 | 9 | 1937 | [ΔΗ, ΛΩ, ΣΩ, ΜΕΝ] | 4 |

Variation 1496

```python # solve the riddle 1c f = df.copy() f = f[f.apply(lambda x: riddle1(x, 1496), axis=1)]
print("Total records: %s" % len(f)) HTML(f.sort_values(0).to_html(index=False)) ```

Total records: 10

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΜΤΕΛΩΝΟΣ | 4 | 0.0 | 9 | 1496 | [ΑΜ, ΤΕ, ΛΩ, ΝΟΣ] | 4 |
ΚΕΚΩΛΥΚΑΣ | 9 | 0.0 | 9 | 1496 | [ΚΕ, ΚΩ, ΛΥ, ΚΑΣ] | 4 |
ΠΑΡΩΤΙΔΑΣ | 17 | 0.0 | 9 | 1496 | [ΠΑ, ΡΩ, ΤΙ, ΔΑΣ] | 4 |
ΠΕΝΙΩΝΤΑΣ | 3 | 0.0 | 9 | 1496 | [ΠΕ, ΝΙ, ΩΝ, ΤΑΣ] | 4 |
ΠΠΝΕΤΩΣΑΝ | 17 | 0.0 | 9 | 1496 | [ΠΠ, ΝΕ, ΤΩ, ΣΑΝ] | 4 |
ΣΥΝΕΚΟΨΑΝ | 4 | 0.0 | 9 | 1496 | [ΣΥ, ΝΕ, ΚΟ, ΨΑΝ] | 4 |
ΣΩΡΗΤΙΚΗΝ | 2 | 0.0 | 9 | 1496 | [ΣΩ, ΡΗ, ΤΙ, ΚΗΝ] | 4 |
ΤΑΠΕΝΩΣΙΝ | 108 | 0.0 | 9 | 1496 | [ΤΑ, ΠΕ, ΝΩ, ΣΙΝ] | 4 |
ΤΕΛΑΜΩΝΟΣ | 175 | 0.0 | 9 | 1496 | [ΤΕ, ΛΑ, ΜΩ, ΝΟΣ] | 4 |
ΦΑΛΑΡΙΔΩΝ | 1 | 0.0 | 9 | 1496 | [ΦΑ, ΛΑ, ΡΙ, ΔΩΝ] | 4 |

#### Variation 506

```python # solve the riddle 1b f = df.copy() f = f[f.apply(lambda x: riddle1(x, 506), axis=1)]
print("Total records: %s" % len(f)) HTML(f.sort_values(0).to_html(index=False)) ```

Total records: 9

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
ΑΝΤΕΛΕΞΕΝ | 2 | 0.00 | 9 | 506 | [ΑΝ, ΤΕ, ΛΕ, ΞΕΝ] | 4 |
ΑΡΣΕΝΙΚΟΝ | 528 | 0.01 | 9 | 506 | [ΑΡ, ΣΕ, ΝΙ, ΚΟΝ] | 4 |
ΑΡΤΙΓΑΛΑΞ | 3 | 0.00 | 9 | 506 | [ΑΡ, ΤΙ, ΓΑ, ΛΑΞ] | 4 |
ΕΓΓΙΖΟΣΗΣ | 9 | 0.00 | 9 | 506 | [ΕΓ, ΓΙ, ΖΟ, ΣΗΣ] | 4 |
ΕΓΓΥΘΗΚΗΝ | 1 | 0.00 | 9 | 506 | [ΕΓ, ΓΥ, ΘΗ, ΚΗΝ] | 4 |
ΘΗΡΑΤΙΚΗΝ | 2 | 0.00 | 9 | 506 | [ΘΗ, ΡΑ, ΤΙ, ΚΗΝ] | 4 |
ΜΕΤΑΒΟΛΗΝ | 2557 | 0.03 | 9 | 506 | [ΜΕ, ΤΑ, ΒΟ, ΛΗΝ] | 4 |
ΠΕΠΗΓΟΣΙΝ | 1 | 0.00 | 9 | 506 | [ΠΕ, ΠΗ, ΓΟ, ΣΙΝ] | 4 |
ΠΠΝΟΜΕΝΑΣ | 4 | 0.00 | 9 | 506 | [ΠΠ, ΝΟ, ΜΕ, ΝΑΣ] | 4 |

### Riddle 2

> And he doth hear four vowels, and two consonants in him are twice announced; the whole sum I will name: for eight ones, and as many tens on these, and yet eight hundred will reveal the name to men insatiate;

The solution of this riddle has been known for isopsephists from the early centuries of Christian fathers. Iraneus and Hippothylus (see "The Greek Qabalah: Alphabetical Mysticism and Numerology in the Ancient World" by Kieren Barry, page 138) commented about the Marcusian Ogdoad (888), former Father with a very colourful language in a [heated word war](https://chs.harvard.edu/CHS/article/display/6308) between gnostic and orthodox theology. It was known that the number 888 referred to ΙΗΣΟΥΣ (Jesus).

So what I'm doing here is to check how well does the programmical approach to solving this type of riddle really work:

```python # riddle 2 wrapper function def riddle2a(x, isopsephy, mutes = 2, vowels = 4):

    return nisopsephy(x, isopsephy) and nletters(x, mutes+vowels) and nvowels(x, vowels) and nmutes(x, mutes)

# solve the riddle 2a h = df.copy() h = h[h.apply(lambda x: riddle2a(x, 888), axis=1)]

print("Total records: %s" % len(h)) HTML(h.sort_values(0).to_html(index=False)) ```

Total records: 6

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| ΑΞΙΩΘΗ | 18 | 0.00 | 6 | 888 | [Α, ΞΙ, Ω, ΘΗ] | 4 |
| ΙΗΣΟΥΣ | 6128 | 0.08 | 6 | 888 | [Ι, Η, ΣΟΥΣ] | 3 |
| ΙΗΤΡΟΥ | 80 | 0.00 | 6 | 888 | [Ι, Η, ΤΡΟΥ] | 3 |
| ΙΟΥΣΗΣ | 57 | 0.00 | 6 | 888 | [Ι, ΟΥ, ΣΗΣ] | 3 |
| ΟΥΣΙΗΣ | 3 | 0.00 | 6 | 888 | [ΟΥ, ΣΙ, ΗΣ] | 3 |
| ΤΗΡΙΟΥ | 11 | 0.00 | 6 | 888 | [ΤΗ, ΡΙ, ΟΥ] | 3 |

__The Number of the **Beast**__

I can't resist of using this solver for the most well known riddle, the infamous Number of the Beast, 666. It is just a matter of a parameter that we can put on the riddle solver and see, how many Greek word candidates there are. Worth of noting is, that in the original puzzle, where the wisdom and ability to calculate is asked, only the isopsephic value is given and that it should be the name of a human (or man in general). Revelation 13:18 (Textus Receptus):

<blockquote> ωδε η σοφια εστιν ο εχων τον νουν ψηφισατω τον αριθμον του θηριου αριθμος γαρ ανθρωπου εστιν και ο αριθμος αυτου χξς </blockquote>

Translation (King James, 1611):

<blockquote> Here is wisdom. Let him that hath understanding count the number of the beast: for it is the number of a man; and his number is Six hundred threescore and six. </blockquote>

```python # riddle 2b wrapper function def riddle2b(x, isopsephy, letters = 0):

    return nisopsephy(x, isopsephy) and (nletters(x, letters) if letters else True)

# solve the riddle 2b i = df.copy() i = i[i.apply(lambda x: riddle2b(x, 666), axis=1)]

print("Total records: %s" % len(i)) HTML(i.sort_values(0).to_html(index=False)) ```

Total records: 592

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| ΑΒΔΗΡΙΤΙΚΑΙΣ | 2 | 0.00 | 12 | 666 | [Α, ΒΔΗ, ΡΙ, ΤΙ, ΚΑΙΣ] | 5 |

ΑΓΝΤΑΤΑΙ | 3 | 0.00 | 8 | 666 | [ΑΓΝ, ΤΑ, ΤΑΙ] | 3 |
ΑΔΜΑΝΤΟΣ | 78 | 0.00 | 8 | 666 | [ΑΔ, ΜΑΝ, ΤΟΣ] | 3 |

Apparent problem is there are too many results for easy examination. On the other hand, one can pick up interesting words from the list nevertheless like: *ΑΡΣΕΝΙΚΟΙΣ*, *ΙΑΠΕΤΟΣ*, *ΛΑΤΕΙΝΟΣ*, and *ΤΕΙΤΑΝ* of cource.

In some old papyrus and texts of early Christian fathers we find that $\chi\xi\varsigma$ is written as 616 rather than 666. So we could see, what are the words meeting this value:

‘‘‘python # solve the riddle 2c i = df.copy() i = i[i.apply(lambda x: riddle2b(x, 616), axis=1)]

print(“Total records: %s” % len(i)) ‘‘‘

    Total records: 611

611 words to analyze!

### Riddle 3

    And three times three hundred and eight and forty shalt thou make complete.

This is also an example what happens if there are not enough parameters for the filter algorithm. In the original riddle it is interesting that the name *Rome* is already mentioned before and after the isopsephical hint. This might be a useful notice however, if the riddle maker didn't want to make puzzle too hard to solve, but gave enough clues for the problem in a very near by context.

‘‘‘python # riddle 3 wrapper function def riddle3(x, isopsephy):

    return nisopsephy(x, isopsephy)

# solve the riddle 3a j = df.copy() j = j[j.apply(lambda x: riddle3(x, 948), axis=1)]

print(“Total records: %s” % len(j)) ‘‘‘

    Total records: 495

`python HTML(j.sort_values(3).head(6).to_html(index=False)) `

0 | 1 | 2 | 3 | 4 | 5 | 6 |
— | — | — | — | — | — | — |
PHMΩ | 2 | 0.00 | 4 | 948 | [PH, MΩ] | 2 |
MHPΩ | 296 | 0.00 | 4 | 948 | [MH, PΩ] | 2 |
ΩPMH | 13 | 0.00 | 4 | 948 | [ΩP, MH] | 2 |
HMPΩ | 31 | 0.00 | 4 | 948 | [HM, PΩ] | 2 |
PΩMH | 1891 | 0.03 | 4 | 948 | [PΩ, MH] | 2 |
MΩPH | 9 | 0.00 | 4 | 948 | [MΩ, PH] | 2 |

Thus, the word for Rome *ΡΩΜΗ* meets the criteria of being having isopsephic value 948. It is repeated 1891 times in the Greek corpora but would have been very difficult to spot from the 495 different words if we didn't know what to search for.

# CHAPTER 3

## Study of the results of the riddle solver


Fig. 3.1: Michelangelo's Erithrean Sibyl, Sistine Chapel

# CHAPTER 4

## Appendix 1 - Store database

Minimum code to create a unique word database for the riddle solver. Download, preprocess, and store Greek corpora, then save and retrieve word database as a *DataFrame* object.

```
pip install grcriddles
```

```
# import download and preprocess function
from grcriddles import download_and_preprocess_corpora, save_database
# call function to create Greek file directories and retrieve corpora data
greek_corpora = download_and_preprocess_corpora()
# save and retrieve word database
df = save_database(greek_corpora)
# how many records there are in the database?
print("Total records: %s" % len(df))
```

Output:

```
Total records: 1708
```

CHAPTER 5

Appendix 2 - Solve riddles

Minimum code to solve isopsephical riddles in the Pseudo-Sibylline oracles. Get word database and filter by different columns.

```
pip install grcriddles
```

```
from grcriddles import get_database
# get words with length 9, isopsephy 1697, consonants 5,
# and the first three syllables having 2 letters each
# syllable count is going to be 4 with above parameters
words = get_database({0: 'Word', 1: 'Count', 3: 'Chars', 4: 'Isopsephy', 5: 'Syllables
↪', 7: 'Vowels', 8: 'Mutes'})
a = words[words['Isopsephy'] == 1697]
a = a[a['Chars'] == 9]
a = a[a['Mutes'] == 5]
a[a.apply(lambda x: len(x['Syllables'][0]) == 2 and \
                    len(x['Syllables'][1]) == 2 and \
                    len(x['Syllables'][2]) == 2, axis=1)]
```

Output:

```
           Count    Chars  Isopsephy          Syllables  Vowels       Mutes
Word
ΑΜΦΕΚΑΛΥΨ      1        9       1697  [ΑΜ, ΦΕ, ΚΑ, ΛΥΨ]       4           5
ΛΗΛΥΘΟΤΩΝ      1        9       1697  [ΛΗ, ΛΥ, ΘΟ, ΤΩΝ]       4           5
ΜΕΤΑΝΑΣΤΩ      1        9       1697  [ΜΕ, ΤΑ, ΝΑ, ΣΤΩ]       4           5
ΣΥΝΩΚΙΣΘΗ     13        9       1697  [ΣΥ, ΝΩ, ΚΙ, ΣΘΗ]       4           5
```

```
# get words containing ΑΜΦΕΚΑΛΥ stem word
words.filter(like="ΑΜΦΕΚΑΛΥ", axis=0)
```

Output:

```
             Count  Chars  Isopsephy              Syllables  Vowels  Mutes
Word
```

ΑΜΦΕΚΑΛΥΠΤΕ	3	11	1382	[ΑΜ, ΦΕ, ΚΑ, ΛΥ, ΠΤΕ]	5	6
ΑΜΦΕΚΑΛΥΠΤΟΝ	2	12	1497	[ΑΜ, ΦΕ, ΚΑ, ΛΥ, ΠΤΟΝ]	5	7
ΑΜΦΕΚΑΛΥΦΘΗ	2	11	1514	[ΑΜ, ΦΕ, ΚΑ, ΛΥ, ΦΘΗ]	5	6
ΑΜΦΕΚΑΛΥΨ	1	9	1697	[ΑΜ, ΦΕ, ΚΑ, ΛΥΨ]	4	5
ΑΜΦΕΚΑΛΥΨΑΝ	2	11	1748	[ΑΜ, ΦΕ, ΚΑ, ΛΥ, ΨΑΝ]	5	6
ΑΜΦΕΚΑΛΥΨΕ	18	10	1702	[ΑΜ, ΦΕ, ΚΑ, ΛΥ, ΨΕ]	5	5
ΑΜΦΕΚΑΛΥΨΕΝ	20	11	1752	[ΑΜ, ΦΕ, ΚΑ, ΛΥ, ΨΕΝ]	5	6