ANUARI DE FILOLOGIA. ANTIQVA ET MEDIAEVALIA (Anu.Filol.Antiq.Mediaeualia)
8/2018, pp. 352—360. ISSN: 2014-1386

THE COMPUTER WILL DO IT: TESTING PYTHON AND ARCMAP ON THE *ITINERARIUM MARITIMUM*

NÚRIA GARCIA I CASACUBERTA
Archaeology, University of Southampton
ngc1g14@soton.ac.uk

ABSTRACT
In this paper I would like to present some code to have the computer generate maps from texts. My test-case is a passage in the *Itinerarium Maritimum*. First of all, place names in the text will be identified. From there, some Python code will locate the coordinates by double-checking the place names with a reference document downloaded from the Pleiades Gazeteer. The data extracted will then be saved in a .txt file and inserted into mapping software, in this case ArcMap, in order to generate the cartographical image.
KEY WORDS: Python, ArcMap, *Itinerarium Maritimum*, maps, Latin texts

QUE HO FACI L'ORDINADOR: APLICACANT PYTHON I ARCMAP A L'*ITINERARIUM MARITIMUM*

RESUM
En aquest article voldria presentar un model de codificació per tal que l'ordinador generi mapes a partir de topònims en textos. El meu cas d'estudi serà un passatge de l'*Itinerarium Maritimum*. Primer de tot, caldrà identificar els topònims en el text. A partir d'aquí, un codi en llenguatge Python els atribuirà coordenades a base de contrastar aquests topònims amb un document de referència descarregat del Pleiades Gazeteer. Les dades es guardaran en un fitxer .txt i s'insertaran en el software topogràfic, en aquest cas ArcMap, per tal de generar la imatge cartogràfica.
PARAULES CLAU*: PYTHON*, *ARCMAP*, ITINERARIUM MARITIMUM, *MAPES*, *TEXTOS LLATINS*

1. INTRODUCTION

Among the myriads of concepts and skills that Professor Mayer taught me, the most persistent theme was an avidness for completeness and interdisciplinarity. These teachings were most present in my recent work with ancient geographical texts, as I became more and more familiar with geomorphological notions and climatic events. Professor Mayer also demonstrated in every lecture his masterful abilities in order to render the texts (and quite often also the intertexts) intelligible to the wider public, something that was truly inspiring and that I endeavour to put into practice every day. In the case of my geographical research, a good starting point to present the content to unfamiliar audiences are, of course, maps. In this paper, I would like to present some basic Python code created by myself in order to have the computer generate them in a few clicks. While I am aware of online repositories of maps or tools to

Data de recepció: 30/12/2017
Data d'acceptació: 15/02/2018

create them[1], my aim in writing my own code was to be able to display sites customised to my needs in an effective, quick, and easy-to-use way. As a beginner in the Digital Humanities, I would like to stress particularly this last point: the simpler the code, the more capable general public will be of using it.
First of all, though, a word of warning: the computer is no substitute at all for the human mind, it is merely a tool to make some things, particularly repatitive tasks, significantly faster. Scholars first have to collect the appropriate data, order it in the correct manner, and present the results adequately for the audience to understand. Otherwise, using a computer, or any other tool for that matter, risks becoming superfluous and insignificant, if not straightforwardly erroneous.

2. Some careful planning

A computer will not be able to do anything that a human has not previously programmed it to do. Moreover, if one gives to a computer a command that is in the slightest sense imprecise or ambiguous, it will either return a bizarre error message or an absolutely unexpected, unwanted, and useless bunch of results. Owing to this, one needs to do some careful planning before one can actually run any software whatsoever.
Obviously, the first question in the process should be: what is it that I need to do? In this case, we need the computer to generate a map showing the locations mentioned in our text. Next follows some thought of how we will communicate with the computer so that it does generate the map. There exists mapping software, both free and open-source, or professional. My software of preference is ArcMap, but this is a personal choice. In order to generate a cartographic image, ArcMap will need a background layer with coordinates embedded in it, and a list of the coordinates that we want it to display. ArcMap does have templates with coordinates, so we do not need to worry about those. Therefore, all that is left for us to do is to feed the coordinates that we need into the mapping software. If the user only has one or two sites, it is probably easier to introduce the coordinates manually. However, when the user is working with a relatively large corpus, copy-pasting coordinates can take large amounts of precious time, as I discovered by experience. Because of this reason, I decided to create some code that would generate the list of coordinates for me.

3. Creating the code and the map

3.1 Identifying and listing the place names in the text

I will concentrate on a specific text, namely a part of the *Itinerarium Maritimum*, the *Itinerarium portuum uel positionum* from Rome to Arelate (pages 79-81 in the edition by Otto Cuntz). The sole reason for that is that this text is interesting for my research, but any other text containing geographical names

[1] More significantly, there was some discussion about this issue recently in the Digital Classicist Mailing List (messages on 30th August 2017). The best institution to suit this need is the Ancient World Mapping Centre, as well as their Github page.

could serve the same purpose. The text of the *Itinerarium* presents two crucial challenges in order to process it in a computer:

1. The orthography does not follow rules, particularly not the standarisation of names established nowadays.
2. Place names appear declined generally in ablative and accusative, depending on the context. Instead, toponyms in the modern gazeteers are found in the nominative.

A good example of that is *Pirgos* in the first line of the text: *Ad portu Augusti Pirgos, positio, m. p. XXVIII.* This is an accusative indicating locus quo, but in the next line, the text reads *Pirgis* in ablative, expressing locus unde (*A Pirgis Panapione, positio, m. p. III*). However, that toponym comes from Greek πύϱγοι ('towers'), and the standard form accepted nowadays, and therefore the form that we can find in the gazeteers, is Pyrgi.

To overcome these challenges, two methods are valid: either making the toponym list manually or taking a few more steps with the computer. If the work involves only a short passage or only a small number of items, my advice is to make the list manually, as it is faster and easier. The computer method involves marking up the text, which is enormously time-consuming. I would only recommend the marking up if we are likely to need to retrieve the toponyms at a later stage, or if we wished to mark up the rest of the text regardless of the map making.

Due to the restricted space of this paper, I will proceed to write my toponym list manually, so that I can focus on the code for mapping. If the text was written in a modern language, and particularly English, there are natural language processing tools already available in Python to detect such terms as place names. However, I am not aware of any such tools focusing on Greek and Latin toponymy. In addition, due to the issues pointed out above, even if those tools existed, their application to the *Itinerarium* would be extremely complex, if not impossible at the present stage of our technology.

Notwithstanding, the computer method can be summarised as follows: first of all, an electronic version of the text is needed. While sites like the Packard Humanities Institute and the Latin Library provide plenty of typed versions of Latin texts, I have not been able to find any electronic version of the *Itinerarium* online. Given that I am only interested in 3 pages, the easiest thing to do was to create the electronic text myself: I scanned the book as a pdf, I enabled the text recognition option, and I pasted the result into Notepad in order to obatin a .txt file that is later machine-readable.

After that, we need XML processing software so that the text can be marked up using the TEI guidelines. This would allow the computer to identify which words are the toponyms and to associate them with their regularised nominative form. The resulting file should be filtered by Python code based on Natural Language Processing. That code should, firstly, pick up the parts of the text marked up as toponyms, secondly select only those that are not repeated (i.e., if a toponym

appears twice or more, it is only considered once), and thirdly print the labels. If we were only dealing with the two lines of text above, the final result should be a list with the words *Portus Augusti*, *Pyrgi*, and *Panapio*.
However, as stated above, for the purposes of this paper it was faster to write the toponym list manually. The full list of toponyms in the passage selected, modified to their nominative forms in the orthography accepted nowadays, is the following: *Portus, Pyrgi, Panapio, Castrum Novum, Centum Cellae, Algae, Rapinium, Graviscae, Maltanum, Quintiana, Regae, Amine, Portus Herculis, Incitaria, Domitiana, Alminia, Portus Talamonis, Umbro, lacus Aprile, Alma, Scabris, Falesia, Populonium, Volaterrae, Portus Pisanus, Pisae, Luna, Macra, Segesta, Portus Veneris, Portus Delphini, Genua, Vada Sabatia, Albigaunum, Portus Maurici, Tavia, Vintimilia, Hercules Monoecus, Avisio, Anao, Olivula, Nicaea, Antipolis, Lero, Forum Iulium, Sambracitanus Sinus, Heraclia Caccabaria, Alconis, Pomponiana, Telo Martius, Tauroention, Cariscae, Citarista portus, Aemines, Imandrae, Massilia, Incarus, Dilis, Fossae Marianae, Gradus Massilitanorum, Rhodanus, Arelate*.

3.2 Generating a list of coordinates

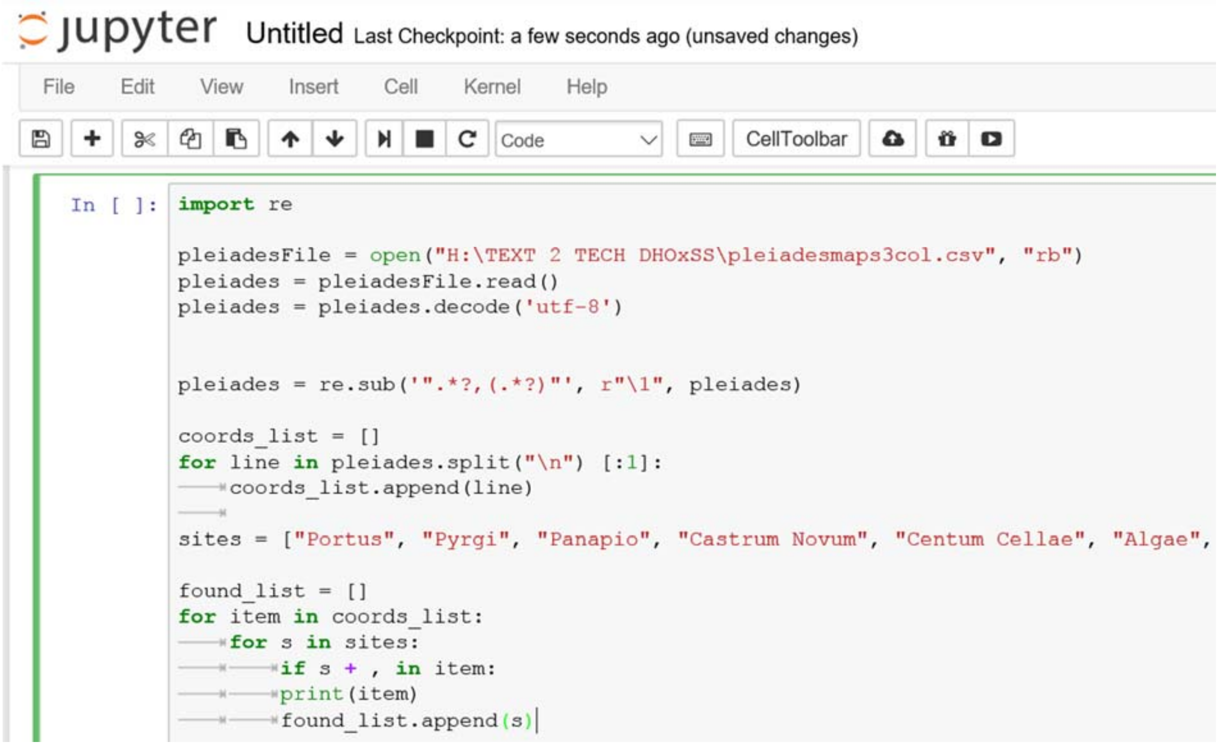
We have now identified the sites that we wish the map to show. We now need to find coordinates for them. The first thing we need is a database of coordinates from which we can select the elements that we wish. In the case of the Classical world, the Pleiades Gazeteer very conveniently offers an option to download their data in CSV UTF-8 format. Their file, which we can open in Excel or similar software, is extremely rich, but for the purposes of our code we only need the columns listing site names, latitudes and longitudes. To simplify things and avoid confusing error messages in the computer, I suggest deleting all the columns except those three and saving the resulting file, always in CSV UTF-8 format. I will now proceed to the actual writing of the code.
The Python software that I am using is the Jupyter Notebook (version 5.0.0) in the Anaconda Navigator. When we open a new file on the Notebook, first of all, we will need to import the '*re*' package, which is used for matching text patterns. After that, we need to write some code to allow the Jupyter Notebook to import our CSV file, which in my case I named *pleiadesmaps3col.csv*. We then need to use Python to instruct the computer to read the file and to decode the UTF-8 format. Finally, we need to use Python to have the computer separate the elements in our CSV file and group them by lines in a new variable, which I have called *coords_list*.
Once we have done that, we can either input in Jupyter the list of toponyms that we need or read it in if we have it elsewhere. For the reasons expressed above, I decided to type my list directly on the Jupyter Notebook. We have to make sure though that the elements in our new variable are entered with the correct syntax (e.g. sites=["Roma*", "Pyrgi", "Panapio"]*).

The next step is to create another variable in the form of an empty list, which I have named *found_list*. We can now introduce a couple of *for*-loops with an *if*-statement at the end, and instruct Python that:

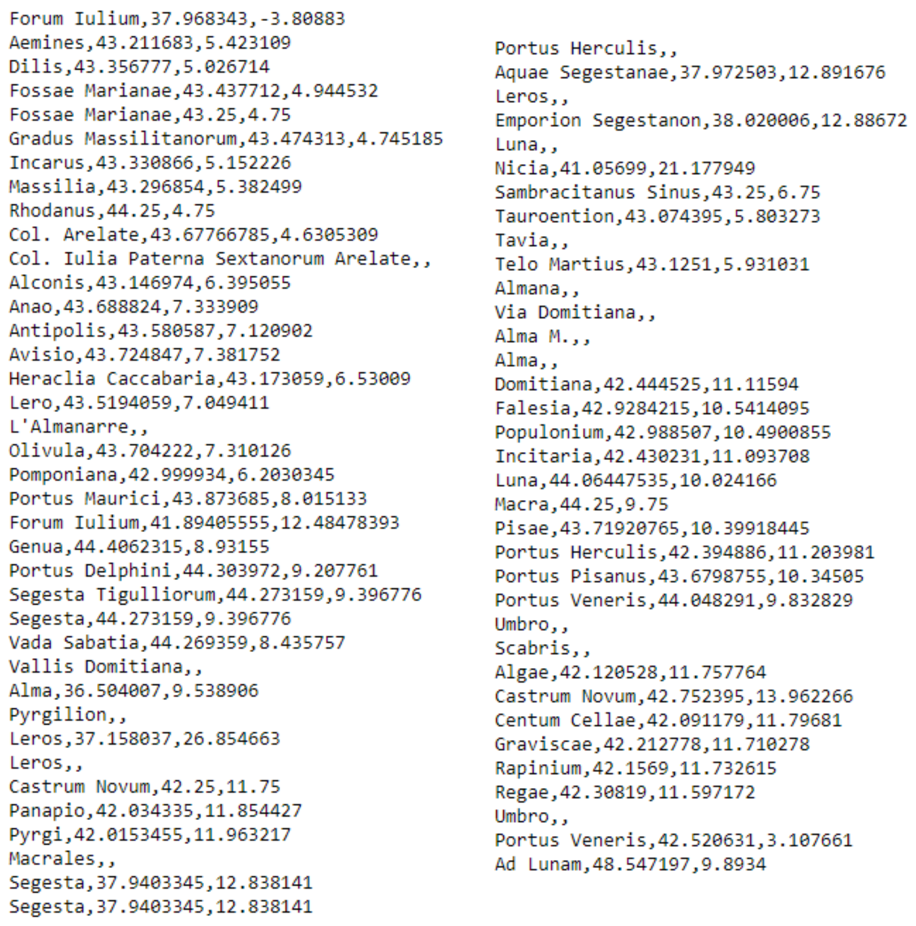
1. for every item in our *coords_list*;
2. it should check every item in our *sites* list;
3. And if the *sites* are contained within the *coords_list*, it should add them to our empty *found_list*, and print the results.

It is useful to add a comma between speechmarks after the site name in this *if* statement to avoid random results where different locations contain part of the spelling of one of the sites that we are actually looking for. In other words, if we do not add the comma to the *if*-statement, when the computer searches for *Macra*, it will return both *Macra* and *Macrales* in the results.

At the end of the process, the code will produce a list of sites names with their coordinates: we should save that list as a .txt file, which is the format recognised by our mapping software. The following figures show the successive commands and the result that they produce when we run the code:


jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Code
CellToolbar

```
In [ ]: import re

pleiadesFile = open("H:\TEXT 2 TECH DHOxSS\pleiadesmaps3col.csv", "rb")
pleiades = pleiadesFile.read()
pleiades = pleiades.decode('utf-8')


pleiades = re.sub('".*?,(.*?)"', r"\1", pleiades)

coords_list = []
for line in pleiades.split("\n") [:1]:
    coords_list.append(line)

sites = ["Portus", "Pyrgi", "Panapio", "Castrum Novum", "Centum Cellae", "Algae",

found_list = []
for item in coords_list:
    for s in sites:
        if s + , in item:
        print(item)
        found_list.append(s)
```



```
Forum Iulium,37.968343,-3.80883
Aemines,43.211683,5.423109
Dilis,43.356777,5.026714
Fossae Marianae,43.437712,4.944532
Fossae Marianae,43.25,4.75
Gradus Massilitanorum,43.474313,4.745185
Incarus,43.330866,5.152226
Massilia,43.296854,5.382499
Rhodanus,44.25,4.75
Col. Arelate,43.67766785,4.6305309
Col. Iulia Paterna Sextanorum Arelate,,
Alconis,43.146974,6.395055
Anao,43.688824,7.333909
Antipolis,43.580587,7.120902
Avisio,43.724847,7.381752
Heraclia Caccabaria,43.173059,6.53009
Lero,43.5194059,7.049411
L'Almanarre,,
Olivula,43.704222,7.310126
Pomponiana,42.999934,6.2030345
Portus Maurici,43.873685,8.015133
Forum Iulium,41.89405555,12.48478393
Genua,44.4062315,8.93155
Portus Delphini,44.303972,9.207761
Segesta Tigulliorum,44.273159,9.396776
Segesta,44.273159,9.396776
Vada Sabatia,44.269359,8.435757
Vallis Domitiana,,
Alma,36.504007,9.538906
Pyrgilion,,
Leros,37.158037,26.854663
Leros,,
Castrum Novum,42.25,11.75
Panapio,42.034335,11.854427
Pyrgi,42.0153455,11.963217
Macrales,,
Segesta,37.9403345,12.838141
Segesta,37.9403345,12.838141
```

```
Portus Herculis,,
Aquae Segestanae,37.972503,12.891676
Leros,,
Emporion Segestanon,38.020006,12.88672
Luna,,
Nicia,41.05699,21.177949
Sambracitanus Sinus,43.25,6.75
Tauroention,43.074395,5.803273
Tavia,,
Telo Martius,43.1251,5.931031
Almana,,
Via Domitiana,,
Alma M.,,
Alma,,
Domitiana,42.444525,11.11594
Falesia,42.9284215,10.5414095
Populonium,42.988507,10.4900855
Incitaria,42.430231,11.093708
Luna,44.06447535,10.024166
Macra,44.25,9.75
Pisae,43.71920765,10.39918445
Portus Herculis,42.394886,11.203981
Portus Pisanus,43.6798755,10.34505
Portus Veneris,44.048291,9.832829
Umbro,,
Scabris,,
Algae,42.120528,11.757764
Castrum Novum,42.752395,13.962266
Centum Cellae,42.091179,11.79681
Graviscae,42.212778,11.710278
Rapinium,42.1569,11.732615
Regae,42.30819,11.597172
Umbro,,
Portus Veneris,42.520631,3.107661
Ad Lunam,48.547197,9.8934
```

Fig. 1 a/b. Outline of the commands in the Python cell and their results

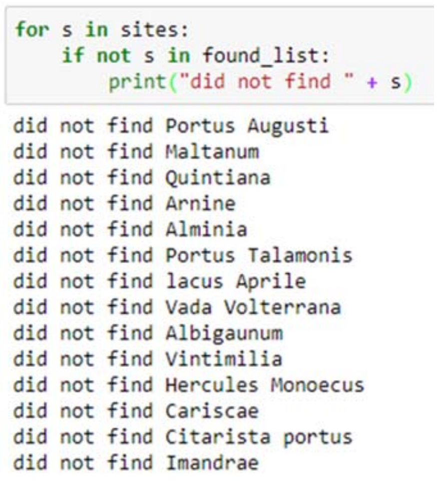
In spite of this success, it is easy to see that the results of the code above need some more polishing before we can introduce them to ArcMap. For example, there are sites like Umbro, which are not associated to any coordinates because those are not known, and contrarily, there are sites like Segesta which appear several times (twice with the exact same coordinates for no obvious reason). Similarly, it is not the case in this text but sometimes there are sites like Portus Veneris, that are repeated because they refer to two different locations with the same name. At this stage the human skill is irreplaceable to sort all these things out.

Repeated sites, or different sites with the same name, are not easy to spot when they do not appear together, so a good first strategy here would be to put the resulting list in alphabetical order. An easy way to exclude the "wrong" site in the case of multiple locations with the same name would be to restrict the scope of the acceptable coordinates with further *if* conditions before the *print* command, in the lines of: "if position 1 (the latitude) is smaller than 40, do not print", "if position 2 (the longitude) is bigger than 15, do not print" etc. Alternatively, if we are only dealing with a restricted number of coordinates, we can add our *if* conditions to print only use the results containing coordinates starting with a certain number.

4. UNFOUND SITES

In fact, one has to be not only familiar with one's data, but also with the data from the external sources consulted, in this case Pleiades. For example, the code is not going to detect *Hercle Manico* or *Nicia* (the spellings in the *Itinerarium*), but *Herakles Monoikos* and *Nicaea*, which are the versions on Pleiades. Using the wrong Latinised or Hellenised version (in this case respectively *Hercules Monoecus* and *Nikaia*) will result in the same lack of results because the code will not be able to find a match. Therefore, it is useful to write a second bit of code in order to learn which locations have not been found, and thus be able to solve potential problems of this kind[2] by finding the spelling version (either Latinised or Hellenised) that the computer is able to match.

A similar issue is posed by the sites that have not been identified on land, and therefore do not feature in the original database that we were searching from (the Pleiades CSV and, consequently, *coords_list*). As one should expect, there are gaps in our knowledge still nowadays and not all of the sites will have been identified successfully in the physical territory or been entered already in a computer system. We have instructed the code above to print only those sites that it has found in the database, but if one is doing serious work, one would also wish to know which sites are missing in order to point those out for the reader or to undertake further research. Again, if we are working with a very small number of sites, we should be able to see from ourselves which are the missing ones. However, the list above is relatively long, and it would take time and a rather sharp eye to identify if there are elements missing in our results. An easy solution here is to add an extra *for*-loop iterating through *sites* and printing those that were not included in our *found_list*:



```
for s in sites:
    if not s in found_list:
        print("did not find " + s)
```

```
did not find Portus Augusti
did not find Maltanum
did not find Quintiana
did not find Arnine
did not find Alminia
did not find Portus Talamonis
did not find lacus Aprile
did not find Vada Volterrana
did not find Albigaunum
did not find Vintimilia
did not find Hercules Monoecus
did not find Cariscae
did not find Citarista portus
did not find Imandrae
```

Figure 2. Sites not found

[2] A similar problem is posed by the mention of a place called *Vada* in the *Itinerarium*. Due to the route followed, that site probably refers to the port known as Vada Volterrana, but this toponym is not found in Pleiades, which is the reason why I have chosen to identify the site as Volaterrae for the purposes of this paper. More accurately, I should have taken the *Vada* from the *"did not find"* results and manually added the coordinates to the .txt file from another source.

On the other hand, toponyms found with missing coordinates, like Umbro, are easy to spot. The researcher can quickly locate them manually and delete them from the list, transfer them to a footnote, or whichever operation they think is suitable. However, a code similar to that above could be run on the *found_list* asking to print those elements that contain two consecutive commas (and therefore have the coordinates missing, like: *Umbro,,*). Again, running the code is more useful when dealing with a large number of results.

5. INTRODUCING THE COORDINATES LIST TO THE MAPPING SOFTWARE

Once our .txt file is saved, we are ready to import it to our mapping software of choice. For my convenience I usually employ ArcMap, but other options are available, like QGIS or even tools based on Google Maps. No matter what software we choose, the essential requirement is that the background map needs to have coordinates embedded in it. Some of the templates in ArcMap, for example, do not support coordinates, and consequently will produce no results. Once this has been established, it is just a matter of importing the .txt file into the software using the appropriate icons. In the case of ArcMap, and perhaps other software as well, we will need to define which is the X and the Y axis for the correct display of the points, but this is easily done in the pop-up menu. As a final result, we obtain the following image:

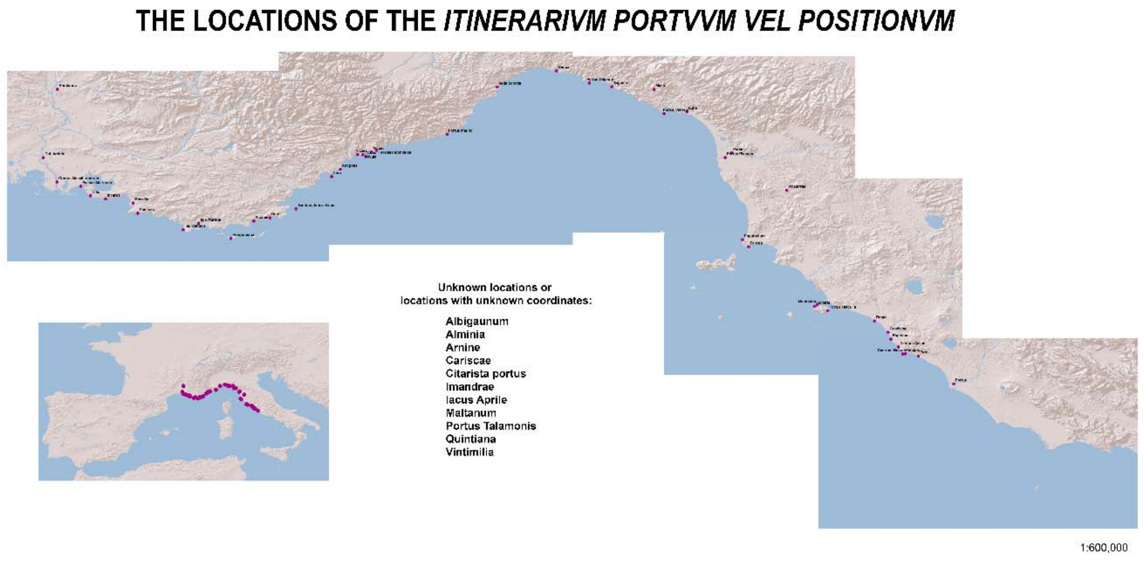


Fig. 3. The resulting map

6. CONCLUSION: SERVICE AND PURPOSE OF THIS SOFTWARE

The use of this software is obvious: to make maps. Maps are an essential complement to help scholars read through each other's research, particularly at the more local sites. While every Classicist knows fully well the locations of Rome, Athens or Tarraco, most of them will not be as familiar with Tortosa or Vic, for example. In such cases, it is mandatory to include a small map so that quality research can reach the wider public effectively.

Personally, I created this software in order to generate maps customised to my needs. Generally speaking, I found that the maps available online in most cases did not mark all of the places that I needed to discuss throughout my thesis, or else they offered images that were not good quality – and this when I was indeed able to find maps that came from reliable websites and were not excluded for use due to copyright issues. In consequence, the necessity of generating my own customised maps became evident.

While the formatting of the text in order to be able to run the code on it may look laborious, I can confirm by experience that it does speed up the work significantly, particularly when dealing with large corpuses or lists. Once the main software has been planned, it can be used over and over very comfortably with different sets of data, and the results are highly satisfactory. I, for my part, would like to encourage vehemently interdisciplinarity between philology and other sciences, as well as widespread scientific diffusion to both specialised and amateur audiences, like Professor Mayer constantly advocates for. Finally, if I may borrow Tacitus's famous quote (*Dialogus de oratoribus*, 32), *ipsa multarum artium scientia etiam aliud agentis nos ornat.*

BIBLIOGRAPHY

CUNTZ, O. (1929), *Itineraria Romana,* Leipzig, Teubner.
MCGILLIVRAY, B . (2014), *Methods in Latin computational linguistics*, Leiden, Brill.

References
Anaconda Navigator: <https://anaconda.org/anaconda/anaconda/navigator>
Ancient World Mapping Centre: <www.awmc.unc.edu>
Ancient World Mapping Centre (Github): <https://github.com/AWMC/geodata>
ArcMap: <desktop.arcgis.com/en/arcmap/>
Digital Classicist Mailing List: <www.digitalclassicist.org/list/>
Pleiades: <pleiades.stoa.org>
Python: <https://python.org>
QGIS: <https://qgis.org>
The Text Encoding Initiative (TEI): <www.tei-c.org>

Note: all websites accessed regularly until December, 2017