**Algorithmics 2021 fall - Exam home assignment**

**Period:** From Dec 20th to Jan 7th.

**Rules:** Exam is personal, no collaborations or even discussions are allowed (with any other people - students, parents, colleagues, etc). Provide concise self-implementations instead of relying so much on copy-pasted code. There are theoretical subtasks and practical subtasks.

Write on the report your full name and study book code. Begin the report by a statement that you completed the exam on your own and cited/reported all materials that you consulted and used in your solutions. Keep track of time spent and indicate the used hours for each task.

Hint: develop and test first on small(er) data sizes before final experimentation on full scale. Necessary hint: make sure to assess your time limitations. There is no need to complete all programming tasks to perfection, think through and choose wisely. Note that tasks may not have any single correct answer. Goal is to demonstrate independent thinking.

**Submission:** via **courses.cs.ut.ee** - the **PDF** and code (zip, tar, tgz, … )

**Task 1: (15p) Regular expressions, automata and matching:**

Use **regexp:** `'(([aeiouy]{3,4}|[ect]{4,6})([lmnor]{3,4}))+'`

**Which words are matched in this file?**
https://raw.githubusercontent.com/dwyl/english-words/master/words.txt

**Are there any words matching from line beginning to very end? Which ones?**

For easy matching use egrep or perl, for example. Also, in the respective manuals or tutorials you can read all about regular expressions.

**Draw an automaton that would match this regular expression**. Can be done manually, no need to build a parser for regexp; no need to do specific Thompson or Glushkov constructions. Just draw the simplest automaton that you can make of it, most likely it shall be non-deterministic.

Create a tabulated representation of that automaton (non-deterministic). How many states does it have? **Read that automaton into your own program.**

**Make a deterministic automaton** (How many states? Print it in tabulated form; If possible, also draw)

**Minimize this automaton** (How many states? Print it in tabulated version and draw)

**With your minimized deterministic automaton match against the same file** and demonstrate that **your code will find exactly the same matches** as by standard matching of regular expressions (grep. perl, python, etc). How many times is your code slower than ordinary egrep?

**Task 2: (10p) What words make books unique?**

Download top-100 books into your computer as text files from here:
https://www.gutenberg.org/browse/scores/top

Build a single list of top 100 **most frequent words** across all these 100 books. Report words and their usage frequency as per 100,000 words, by order of frequency. Which 2-3 words are "most surprising" to you?

Build a list of top 50 **most characteristic words relevant to each book** by comparing the frequency of word occurrences in the book over frequency of same words across all other books. Report 3 characteristic word lists in your report. Explain which measure you used. And how these words are characteristic to the book.

Build a Word Map producing algorithm and tool on your own. **Use this same score that you used as a basis of calculating an importance of the word for creating a "Wordmap" (image) of those 50 words.** Come up with some (approximate) hierarchical recursive division of the image to accommodate these "rectangles", as "Tree map" or "Wordmap" and print the word within that area. Report 3 images in report and others in the zip file. Insert in your report **your algorithm's pseudocode** for **Wordmap** generation

In the compressed file include the lists for most frequent words across all books and most specific ones for each book, together with respective word maps generated by your code.

**Task 3 Optimization for fitting the best function to match the data points (15 p):**

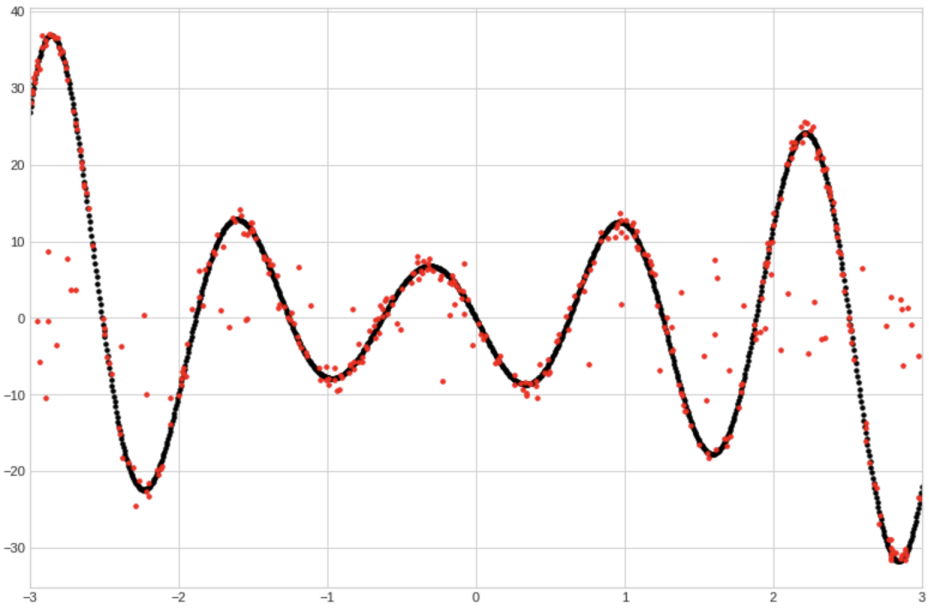
Use your skills and create a method for "guessing" the function that generated the example data (with added noise) from that data alone. Use the data generator provided and come up with the best fitting function against these data points. The below plot shows function **"0.567 * np.sin(x*5)*( + 4.808*1 + 1.977*x + 1.075*x*x + 1.105*x*x*x + 0.638*np.sin(x) + 0.64*np.cos(x) )"** and the data generated based on that function (in red).

The code used to generate the below plot and respective data is in the Colab file.
https://colab.research.google.com/drive/1gXwtL9nFOpjYO-dQjy499_MqUcmqY8qx#scrollTo=dkP_p6oX4v0c

The second cell in Python notebook is your data generator. Functions get more complex, starting from lower polynomial functions, then added some sin(x) and cos(x) and finally multiplied by sin(5*x). Your task is to "guess" the weights used in the function (a numeric vector). In this case: [ 4.808, 1.977, 1.075, 1.105, 0.638, 0.64, 0.567 ].

**The first cell produces this:**

```
Actual:   0.567 * np.sin(x*5)*(  + 4.808*1 + 1.977*x + 1.075*x*x + 1.105*x*x*x + 0.638*np.sin(x) + 0.64*np.cos(x) )
 [1.6731091072586137, 0.8917209390157854, 2.2502803943954657, 1.6079972676716805, 1.8750846331468507, -0.09236587582721
 [-15.730779087670262, 10.391774642756017, 24.517569487912738, -2.04460554950856, -0.8992518618631733, 2.54149813892553
```




**Make your own optimizer code!** In principle it could be attempted with some simulated annealing or genetic algorithms techniques (genome is then (up to) 7 floating point weights). Most likely the best choice would be to use **Differential Evolution.**

Start with simple cases (fewer parameters), figure out how to fit the simpler functions first. **Use the mean and median square errors** (distance from your own function to the reported points in the data). **Experiment with the population size, and measure the convergence rate in order to decide when to stop.**

Report for each function class the best and worst predictions (10 repetitions, 3 lengths of data). Generate respective plots and add these directly into your report. Plot can show the data points (e.g. gray), actual function (e.g. black) and your predictions based on various length data and/or chosen method (use colors and their intensities, e.g. green, blue, red).

Report also the mean and median errors (it is recommended to optimize against median square error). If you want to play with the level of error in the underlying data, please feel free to modify the generator.

**Does more data help match the function better?**

**Do median error and mean error provide very different results?**

**Bonus points (5) if you in addition demonstrate how some other optimizer could be used directly for this example case.**

EOF, submission deadline January 7th