

Example to plot directly into latex

19-10-2019

1 Introduction

Welcome, this document presents our market analysis for the TruCol consultancy. The objective of this document is to provide some basic insight into the order of magnitude of the potential of the TruCol consultancy to generate returns for its potential investors. Based on various pitch templates, [6], and private communications, we intend to convey this information through sharing our model and estimate of the following market parameters for the TruCol consultancy:

- **Total addressable market (TAM)**, or total available market, is the total market demand for a product or service, calculated in annual revenue or unit sales if 100% of the available market is achieved[2].
- **Serviceable available market (SAM)** is the portion of TAM targeted and served by a company's products or services[2].
- **Serviceable obtainable market (SOM)**, or share of market, is the percentage of SAM which is realistically reached[2].

Since we currently have little experience on this topic within our team we are making our data and assumptions as transparent as possible, both in this document as in our code. This way we hope to improve our model based on your feedback by enabling you to tingle with it yourself. Additionally, because the market analysis consists of a rough estimate, three different estimation methods are used for generating the TAM, SAM and SOM estimates. The redundancy is introduced to establish some reference frame within the results.

The assumptions and datapoints for the respective models are specified in section 2. Next, the models are described in section 3 (the Python models themselves are included as appendices in section D to section F respectively). The results of these models are presented in section 4. To shed some light on how sensitive the model is to for example changes in assumptions, a sensitivity analysis is presented for each model in section 5. Next the results and sensitivity of the models are discussed in section 6 and a conclusion is provided in section 7.

We invite you to tinker with the assumptions and models yourself! The data and plots in this report are automatically updated if you run `python -m code.project1.src`. If you experience any difficulties in running the code, simply reach out to us, (click on issues on the github page) and we are happy to get you running the code.

2 Assumptions

2.1 Top Down

2.2 Bottom Up

2.3 Value Theory

3 Model Description

This

3.1 Market

To compute the TAM, SAM and SOM, some form of market definition can be used. To this end it can be valuable to specify exactly what the TruCol consultancy does, where it adds value and how it does that. Furthermore, since these three afore mentioned estimates pertain to a potential future, the potential, yet deemed feasible, activities of the TruCol consultancy are included.

The TruCol consultancy provides advice and support to companies on how companies can get the most out of the TruCol protocol. To understand this the following assumptions are shared:

- **asu-0:** Task completions of tasks that are completed using the TruCol protocol are deterministically verifiable.
- **asu-1:** Solutions of tasks that are completed using the TruCol protocol are of sufficient quality.
- **asu-2:** Tasks that are completed using the TruCol protocol can be solved for the lowest costprice that is currently available in this world.
- **asu-3:** No personel needs to be attracted, screened, hired nor fired for tasks that are completed using the TruCol protocol.
- **asu-4:** Companies can benefit from public particular solutions to their task specifications.
- **asu-5:** By sampling from a bigger talent pool (this world), the average performance of the solutions will be better than what is produced by the in-house talent pool, or, for equal solution performance, a faster rate of development can be obtained on average for an equal or lower price.

Under these assumptions, one can conclude that an economically rational company would try to off-load as much of their required tasks into the TruCol protocol as it would minimise their operational costs and/or improve algorithmic efficiency of their solutions.

We help companies identify the tasks for which they can use the TruCol protocol, and we assist them in writing safe test specifications that are not easily hackable. This implies that under the given set of assumptions, the TAM for the TruCol protocol can be defined as the total costs that the companies (and consumers) in this world are willing to pay for assistance on using the TruCol protocol.

3.1.1 TruCol Total Addressable Logistics Market

This sub-sub section illustrates a rough method of estimating the logistics subsegment of the TAM for the TruCol protocol. To do this, an example of algorithmic optimisation within the logistics market as presented by McKinsey & Company is generalised conservatively to a rough estimate of the total logistics market size.

A clear example of a logistics company succesfully hiring a consultancy for algorithmic optimisation is documented by McKinsey & Company in the "how they help their clients" segment of their website[3]. The study how reports McKinsey's team, among which the McKinsey's Strategic Network Analytic Center helped an Asian logistics company. With McKinseys team, the logistics company realised an *in line haul network cost* reduction of 3.6% while reducing their *transit time* with 0.8%, yielding an overall 16% increase in profit for the logistics company, without compromising the quality. To use this report as a valuable resource to generate some rough estimates on market size, the following assumptions are made:

- **asu-6:** The logistics company made a net profit by hiring McKinsey & Company in this particular ordeal.
- **asu-7:** The example of a 16% increase in profit is generalizable to a conservative potential 0.1% of profit increases through algorithmic optimisation accross the entire logistics industry.
- **asu-8:** Companies are willing to pay at least 1 % of their potential profit increases for the assistance the TruCol consultancy company provides in identifying opportunities for optimisation and for improving test-specification security.

Based on those assumptions, one could find a potential yearly profit increase accross the entire logistics sector by summing the net profit of the logistics sector. [1] claims that this company [7] valued the logistics market at 8.1 trillion in 2016. Additionally [1] claims [7] estimates the logistics market value will grow to 15.5 trillion in 2023. However, no figures on profit are found. Hence individual companies are explored.

For DHL one can find on pdf page 37/170 in [4] that the annual profit for DHL in 2019 was 4.1 billion.

For UPS one can find on pdf page 4/257 in [8] that the annual unadjusted operating profit for UPS in 2020 was 7.7 billion. Note, [9] says UPS had a net operating profit of 1.1 billion in Q1 of 2020, implying they had to almost double their average profit in the remaining three quarters of 2020 to be consistent with an annual 7.7 billion.

For FedEx the net income as reported for 2020 has been 1.29 \$ billion in pdf page 2/17 [5].

- **Asu-9:** The net income as reported (GAAP) by FedEx can be interpreted as the profit by FedEx.

Next, the claim that fragmentation of the global market implied in 2016 that Deutsche Post DHL, Ceva Logistics, UPS, and FedEx, control less than 15% of that global market allows estimating a limit on the net global profit made in the logistics market based on the following assumptions:

- **Asu-10:** The market segment in the global logistics market maintained by the combination of DHL, UPS and FedEx is at most 15% in 2020.
- **Asu-11:** The profit in the remaining 85% of the global logistics market has the same average yearly profitability per percent market share as the combination of DHL, UPS and FedEx.

Based on assumptions 1-11 one could estimate an upperbound of

$$\begin{aligned}
net - profit_{DHL+UPS+FedEx} &= 4.1 + 7.7 + 1.29 = 13.09 \text{ billion} \\
\frac{net - profit_{global_logistics}}{net - profit_{DHL+UPS+FedEx}} &= \frac{0.85}{0.15} \\
net - profit_{global_logistics} &= net - profit_{DHL+UPS+FedEx} \cdot \frac{0.85}{0.15} \\
net - profit_{global_logistics} &= \frac{13.09 \cdot 0.85}{0.15} \\
net - profit_{global_logistics} &= 74.2 \text{ billion}
\end{aligned} \tag{1}$$

Hence, if each of those companies in the logistics sector could increase their profits on average annually by .1% using algorithmic optimisation, and if they would use the TruCol protocol to do that, and if they would be willing to invest 1% of that profit in our support and assistance in getting the most out of the TruCol protocol, we would currently estimate that this would yield roughly an income of $74.2 \cdot 0.001 \cdot 0.01 = \0.74 million

3.1.2 Additional addressable markets

Since the TruCol consultancy is market agnostic, we also seek to assist in algorithmic optimisation outside the logistics market. Several markets are worth mentioning in particular as we expect them to either heavily rely on algorithmic optimisations, or because they are particularly suited for the TruCol protocol.

- **(Automated) trading** In the highly competitive market of (automated) trading, algorithmic optimisations are key to making successful trades.
- **Space Sector** The space engineering sector already has a relatively high test driven development[?], this lowers the adoption costs of the TruCol protocol relative to most industries. Furthermore, space applications are heavily mass constrained, which generally makes them highly energy constrained as well. These energy constraints emphasise the importance of algorithmic optimisations, for example in telecommunications satellites and swarm robots.
- **Innovative Materials Research** The domain of material science has been adopting algorithmic search strategies to find new materials [?].
- **Pharmaceutical Industry** Another example of a large market that has been shifting to adopt algorithmic search strategies to find new medicines.

Each of these are multi billion dollar markets which can contribute to the TAM of the TruCol consultancy.

3.2 Emerging markets

Beyond those listed markets, the following emerging markets could be great opportunities for the TruCol consultancy to latch in and grow along in.

- **Neuromorphic Computing** This field is developing new complexity theory to adapt to the unconventional computation methods. This is an interesting opportunity to explore the versatility of the TruCol protocol.
- **Quantum Computing** This is another upcoming field with many new algorithmic implementations. The newness of the field may suggest that the amount of optimisation and exploration to be done is relatively high, possibly indicating a relatively large potential for the TruCol protocol. However, currently our team does not yet contain experience in this type of algorithmic developments.
- **Artificial Intelligence** With the introduction of GPT 3 the world has seen an example of an AI engine that is able to generate code for some basic tasks [?]. The TruCol protocol could catalyse the usage of such AI engines based on requirement specification. We expect that users of the TruCol protocol will develop a tactical advantage on requirement specifications for AI engines.

3.3 Market Size

3.4 Market Trajectory

Since the market size estimation models are somewhat of an abstract/subjective task, three different approaches are used in an attempt to establish some reference material with respect to accuracy.

Before the model is presented, it is important to realise that we propose a consultancy service that operates as an optimisation service. This means that if a certain activity, e.g. a logistics company has operational cost of 5 \$million/day, our consultancy service is only able to earn at most the margin of improvement we are able to bring

our customer. So suppose the independent usage of the TruCol provides the customer with a 2% optimisation in their operational costs, yielding them $5.000.000 \cdot 0.02 = 100.000/day\$$. Suppose our expertise is able to enable them to yield a 3% optimisation by identifying the relevant development/system processes and supporting them in improved test specification. In that assumption our consultancy would bring them an additional $3-2=1\%$ which would translate roughly to 50.000\$. That would be the value we bring to the logistics company in this hypothetical scenario.

In reality this example is oversimplified, the 2% the company could get by themselves would involve some risk pertaining to inaccurate test specification which could lead to loss of the bounty. Our company reduces this risk by providing test-specification security expertise. Furthermore, our interaction with the client may bring the client experience that can be applied in future applications of the TruCol protocol, hence the value to we bring to the client is larger than the amount they gain in terms of optimisation w.r.t. the case where they use the protocol themselves.

3.4.1 Top Down

The Top-Down approach

3.4.2 Bottom Up

3.4.3 Value Theory

4 Results

4.1 Top Down

4.2 Top Down

4.3 Top Down

5 Sensitivity Analysis

5.1 Top Down

5.2 Bottom Up

5.3 Value Theory

6 Discussion

6.1 Top Down

6.2 Bottom Up

6.3 Value Theory

7 Conclusion

References

- [1] Andrew Allen. Logistics industry to be worth 15.5tnby2023.
- [2] The businessplan shop. Tam sam som - what it means and why it matters.
- [3] McKinsey & Company. Algorithmic route optimization improves revenue for a logistics company.
- [4] DHL. 2019 financial year - focussing on our profitable core.
- [5] FedEx. Fedex corp. reports fourth quarter and full-year earnings.
- [6] Haje Jan Kamps. *Pitch Deck Design*. Apress, Berkeley, CA, 2020.
- [7] Transparency Market Research. Logistics market - snapshot.
- [8] UPS. Notice of 2021 annual meeting of shareowners and proxy statement & 2020 annual report on form 10-k.
- [9] UPS. Ups q21 earnings call.

A Appendix __main__.py

```
1 import os
2 from .Main import Main
3
4 print(f"Hi, I'll be running the main code, and I'll let you know when
   ↪ I'm done.")
5 project_nr = 1
6 main = Main()
7
8 # export the code to latex
9 main.export_code_to_latex(project_nr)
10
11 # compile the latex report
12 main.compile_latex_report(project_nr)
13
14 print(f"Done.")
```

B Appendix Main.py

```
1 # Example code that creates plots directly in report
2 # Code is an implementation of a genetic algorithm
3 import random
4 from matplotlib import pyplot as plt
5 from matplotlib import lines
6 import matplotlib.pyplot as plt
7 import numpy as np
8
9 from .Compile_latex import Compile_latex
10 from .Plot_to_tex import Plot_to_tex as plt_tex
11 from .Export_code_to_latex import export_code_to_latex
12
13 # define global variables for genetic algorithm example
14 string_length = 100
15 mutation_chance = 1.0 / string_length
16 max_iterations = 1500
17
18
19 class Main:
20     def __init__(self):
21         pass
22
23     def export_code_to_latex(self, project_nr):
24         export_code_to_latex("main.tex", project_nr)
25
26     def compile_latex_report(self, project_nr):
27         """compiles latex code to pdf"""
28         compile_latex = Compile_latex(project_nr, "main.tex")
29
30     def addTwo(self, x):
31         """adds two to the incoming integer and returns the result of
32             ↪ the computation."""
33         return x + 2
34
35 if __name__ == "__main__":
36     # initialize main class
37     main = Main()
```

C Appendix Compile_latex.py

```
1 # runs a jupyter notebook and converts it to pdf
2
3 import os
4 import shutil
5 import nbformat
6 from nbconvert.preprocessors import ExecutePreprocessor
7
8
9 class Compile_latex:
10     def __init__(self, project_nr, latex_filename):
11         self.script_dir = self.get_script_dir()
12         relative_dir = f"latex/project{project_nr}/"
13         self.compile_latex(relative_dir, latex_filename)
14         self.clean_up_after_compilation(latex_filename)
15         self.move_pdf_into_latex_dir(relative_dir, latex_filename)
16
17     # runs jupyter notebook
18     def compile_latex(self, relative_dir, latex_filename):
19         os.system(f"pdflatex {relative_dir}{latex_filename}")
20
21     def clean_up_after_compilation(self, latex_filename):
22         latex_filename_without_extention = latex_filename[:-4]
23         print(f"latex_filename_without_extention={
24             ↪ latex_filename_without_extention}")
25         self.delete_file_if_exists(f"{
26             ↪ latex_filename_without_extention}.aux")
27         self.delete_file_if_exists(f"{
28             ↪ latex_filename_without_extention}.log")
29         self.delete_file_if_exists(f"texput.log")
30
31     def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
32         pdf_filename = f"{latex_filename[:-4]}.pdf"
33         destination = f"{self.get_script_dir()}/../../../{
34             ↪ relative_dir}{pdf_filename}"
35
36         try:
37             shutil.move(pdf_filename, destination)
38         except:
39             print("Error while moving file ", pdf_filename)
40
41     def delete_file_if_exists(self, filename):
42         try:
43             os.remove(filename)
44         except:
45             print(
46                 f"Error while deleting file: {filename} but that is
47                 ↪ not too bad because the intention is for it to
48                 ↪ not be there."
49             )
50
51     def get_script_dir(self):
52         """returns the directory of this script regardless of from
53             ↪ which level the code is executed"""
54         return os.path.dirname(__file__)
55
56 if __name__ == "__main__":
57     main = Compile_latex()
```

D Appendix Export_code_to_latex.py

```
1 # runs a jupyter notebook and converts it to pdf
2 import os
3 import shutil
4 import nbformat
5 from nbconvert.preprocessors import ExecutePreprocessor
6
7
8 def export_code_to_latex(main_latex_filename, project_nr):
9     """This function exports the python files and compiled pdfs of
10         ↪ jupyter notebooks into the
11         latex of the same project number. First it scans which appendices
12         ↪ (without code, without
13         notebooks) are already manually included in the main latex code.
14         ↪ Next, all appendices
15         that contain the python code are either found or created in the
16         ↪ following order:
17         First, the __main__.py file is included, followed by the main.py
18         ↪ file, followed by all
19         python code files in alphabetic order. After this, all the pdfs
20         ↪ of the compiled notebooks
21         are added in alphabetic order of filename. This order of
22         ↪ appendices is overwritten in the
23         main tex file.
24
25     :param main_latex_filename: Name of the main latex document of
26         ↪ this project number
27     :param project_nr: The number indicating which project this code
28         ↪ pertains to.
29     """
30     script_dir = get_script_dir()
31     relative_dir = f"latex/project{project_nr}/"
32     appendix_dir = script_dir + "../..../" + relative_dir + "
33         ↪ Appendices/"
34     path_to_main_latex_file = (
35         f"{script_dir}/../..../{relative_dir}/{main_latex_filename}"
36     )
37     root_dir = script_dir[0 : script_dir.rfind(f"code/project{
38         ↪ project_nr}")]
39
40     # Get paths to files containing python code.
41     python_filepaths = get_filenames_in_dir("py", script_dir, ["
42         ↪ __init__.py"])
43     compiled_notebook_pdf_filepaths = get_compiled_notebook_paths(
44         ↪ script_dir)
45
46     # Check which files are already included in the latex appendices
47     ↪ .
48     python_files_already_included_in_appendices =
49         ↪ get_code_files_already_included_in_appendices(
50             python_filepaths, appendix_dir, ".py", project_nr, root_dir
51         )
52     notebook_pdf_files_already_included_in_appendices =
53         ↪ get_code_files_already_included_in_appendices(
54             compiled_notebook_pdf_filepaths, appendix_dir, ".ipynb",
55             ↪ project_nr, root_dir,
56     )
57
58     # Get which appendices are still missing.
59     missing_python_files_in_appendices =
60         ↪ get_code_files_not_yet_included_in_appendices(
```



```

43         python_filepaths, python_files_already_included_in_appendices
44         ↪ , ".py"
45     )
46     missing_notebook_files_in_appendices =
47     ↪ get_code_files_not_yet_included_in_appendices(
48         compiled_notebook_pdf_filepaths,
49         notebook_pdf_files_already_included_in_appendices,
50         ".pdf",
51     )
52     # Create the missing appendices.
53     created_python_appendix_filenames = create_appendices_with_code(
54         ↪ appendix_dir, missing_python_files_in_appendices, ".py",
55         ↪ project_nr, root_dir
56     )
57     created_notebook_appendix_filenames = create_appendices_with_code
58     ↪ (
59         ↪ appendix_dir,
60         ↪ missing_notebook_files_in_appendices,
61         ↪ ".ipynb",
62         ↪ project_nr,
63         ↪ root_dir,
64     )
65     appendices = get_list_of_appendix_files(
66         ↪ appendix_dir, compiled_notebook_pdf_filepaths,
67         ↪ python_filepaths
68     )
69     main_tex_code, start_index, end_index, appendix_tex_code =
70     ↪ get_appendix_tex_code(
71         ↪ path_to_main_latex_file
72     )
73     # assumes non-included non-code appendices should not be included
74     ↪ :
75     # overwrite the existing appendix lists with the current appendix
76     ↪ list.
77     (
78         ↪ non_code_appendices,
79         ↪ main_non_code_appendix_inclusion_lines,
80     ) = get_order_of_non_code_appendices_in_main(appendices,
81         ↪ appendix_tex_code)
82     python_appendix_filenames = list(
83         ↪ map(
84             ↪ lambda x: x.appendix_filename,
85             ↪ filter_appendices_by_type(appendices, "python"),
86         )
87     )
88     sorted_created_python_appendices = sort_python_appendices(
89         ↪ filter_appendices_by_type(appendices, "python")
90     )
91     sorted_python_appendix_filenames = list(
92         ↪ map(lambda x: x.appendix_filename,
93             ↪ sorted_created_python_appendices)
94     )
95     notebook_appendix_filenames = list(
96         ↪ map(
97             ↪ lambda x: x.appendix_filename,

```

```

95         filter_appendices_by_type(appendices, "notebook"),
96     )
97 )
98 sorted_created_notebook_appendices =
99     ↪ sort_notebook_appendices_alphabetically(
100         filter_appendices_by_type(appendices, "notebook")
101     )
102 sorted_notebook_appendix_filenames = list(
103     ↪ map(lambda x: x.appendix_filename,
104         sorted_created_notebook_appendices)
105 )
106
107 appendix_latex_code = create_appendices_latex_code(
108     main_non_code_appendix_inclusion_lines,
109     sorted_created_notebook_appendices,
110     project_nr,
111     sorted_created_python_appendices,
112 )
113
114 updated_main_tex_code = substitute_appendix_code(
115     end_index, main_tex_code, start_index, appendix_latex_code
116 )
117 print(f'\n\n')
118 print(f"updated_main_tex_code={updated_main_tex_code}")
119
120 overwrite_content_to_file(updated_main_tex_code,
121     ↪ path_to_main_latex_file)
122
123 def create_appendices_latex_code(
124     main_non_code_appendix_inclusion_lines,
125     notebook_appendices,
126     project_nr,
127     python_appendices,
128 ):
129     """Creates the latex code that includeds the appendices in the
130     ↪ main latex file.
131
132     :param main_non_code_appendix_inclusion_lines: latex code that
133     ↪ includes the appendices that do not contain python code nor
134     ↪ notebooks
135     :param notebook_appendices: List of Appendix objects representing
136     ↪ appendices that include the pdf files of compiled Jupiter
137     ↪ notebooks
138     :param project_nr: The number indicating which project this code
139     ↪ pertains to.
140     :param python_appendices: List of Appendix objects representing
141     ↪ appendices that include the python code files.
142     """
143     main_appendix_inclusion_lines =
144         ↪ main_non_code_appendix_inclusion_lines
145     print(f"main_appendix_inclusion_lines={
146         ↪ main_appendix_inclusion_lines}")
147
148     appendices_of_all_types = [python_appendices, notebook_appendices
149         ↪ ]
150
151     print(f"\n\n")
152     main_appendix_inclusion_lines.append(
153         ↪ f"\IfFileExists{{latex/project{project_nr}/main.tex}}{{{"
154     )
155     main_appendix_inclusion_lines = append_latex_inclusion_command(

```

```

144         appendices_of_all_types, True, main_appendix_inclusion_lines,
145         ↪ project_nr,
146     )
147     main_appendix_inclusion_lines.append(f"{}{{{{}}}}")
148     main_appendix_inclusion_lines = append_latex_inclusion_command(
149         ↪ appendices_of_all_types, False, main_appendix_inclusion_lines
150         ↪ , project_nr,
151     )
152     #main_appendix_inclusion_lines.append(f"{}{{{{}}}}")
153     print(f"main_appendix_inclusion_lines={
154         ↪ main_appendix_inclusion_lines}")
155     return main_appendix_inclusion_lines
156
157 def append_latex_inclusion_command(
158     appendices_of_all_types, is_from_root_dir,
159     ↪ main_appendix_inclusion_lines, project_nr
160 ):
161     for appendix_type in appendices_of_all_types:
162         for appendix in appendix_type:
163             line = update_appendix_tex_code(
164                 ↪ appendix.appendix_filename, is_from_root_dir,
165                 ↪ project_nr
166             )
167             print(f"appendix.appendix_filename={appendix.
168                 ↪ appendix_filename}")
169             main_appendix_inclusion_lines.append(line)
170     return main_appendix_inclusion_lines
171
172 def filter_appendices_by_type(appendices, appendix_type):
173     """Returns the list of all appendices of a certain appendix type,
174     ↪ from the incoming list of Appendix objects.
175
176     :param appendices: List of Appendix objects
177     :param appendix_type: Can consist of "no_code", "python", or "
178     ↪ notebook" and indicates different appendix types
179     """
180     return_appendices = []
181     for appendix in appendices:
182         if appendix.appendix_type == appendix_type:
183             return_appendices.append(appendix)
184     return return_appendices
185
186 def sort_python_appendices(appendices):
187     """First puts __main__.py, followed by main.py followed by a-z
188     ↪ code files.
189
190     :param appendices: List of Appendix objects
191     """
192     return_appendices = []
193     for appendix in appendices: # first get appendix containing
194         ↪ __main__.py
195         if (appendix.code_filename == "__main__.py") or (
196             ↪ appendix.code_filename == "__Main__.py"
197         ):
198             return_appendices.append(appendix)
199             appendices.remove(appendix)
200     for appendix in appendices: # second get appendix containing
201         ↪ main.py
202         if (appendix.code_filename == "main.py") or (

```

```

195         appendix.code_filename == "Main.py"
196     ):
197         return_appendices.append(appendix)
198         appendices.remove(appendix)
199 return_appendices
200
201 # Filter remaining appendices in order of a-z
202 filtered_remaining_appendices = [
203     i for i in appendices if i.code_filename is not None
204 ]
205 appendices_sorted_a_z = sort_appendices_on_code_filename(
206     filtered_remaining_appendices
207 )
208 return return_appendices + appendices_sorted_a_z
209
210
211 def sort_notebook_appendices_alphabetically(appendices):
212     """Sorts notebook appendix objects alphabetic order of their pdf
213     ↪ filenames.
214
215     :param appendices: List of Appendix objects
216     """
217     return_appendices = []
218     filtered_remaining_appendices = [
219         i for i in appendices if i.code_filename is not None
220     ]
221     appendices_sorted_a_z = sort_appendices_on_code_filename(
222         filtered_remaining_appendices
223     )
224     return return_appendices + appendices_sorted_a_z
225
226 def sort_appendices_on_code_filename(appendices):
227     """Returns a list of Appendix objects that are sorted and based
228     ↪ on the property: code_filename.
229     Assumes the incoming appendices only contain python files.
230
231     :param appendices: List of Appendix objects
232     """
233     attributes = list(map(lambda x: x.code_filename, appendices))
234     sorted_indices = sorted(range(len(attributes)), key=lambda k:
235         ↪ attributes[k])
236     sorted_list = []
237     for i in sorted_indices:
238         sorted_list.append(appendices[i])
239     return sorted_list
240
241 def get_order_of_non_code_appendices_in_main(appendices,
242     ↪ appendix_tex_code):
243     """Scans the lines of appendices in the main code, and returns
244     ↪ the lines
245     of the appendices that do not contain code, in the order in which
246     ↪ they were
247     included in the main latex file.
248
249     :param appendices: List of Appendix objects
250     :param appendix_tex_code: latex code from the main latex file
251     ↪ that includes the appendices
252     """
253     non_code_appendices = []
254     non_code_appendix_lines = []

```

```

250 appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
251 for line in appendix_tex_code:
252     appendix_filename = get_filename_from_latex_appendix_line(
253         ↪ appendices, line)
254
255     # Check if line is not commented
256     if not appendix_filename is None:
257         if not line_is_commented(line, appendix_filename):
258             appendix = get_appendix_from_filename(appendices,
259                 ↪ appendix_filename)
260             if appendix.appendix_type == "no_code":
261                 non_code_appendices.append(appendix)
262                 non_code_appendix_lines.append(line)
263
264 return non_code_appendices, non_code_appendix_lines
265
266 def get_filename_from_latex_appendix_line(appendices, appendix_line):
267     """Returns the first filename from a list of incoming filenames
268     ↪ that
269     occurs in a latex code line.
270
271     :param appendices: List of Appendix objects
272     :param appendix_line: latex code (in particular expected to be
273         ↪ the code from main that is used to include appendix latex
274         ↪ files.)
275     """
276     for filename in list(map(lambda appendix: appendix.
277         ↪ appendix_filename, appendices)):
278         if filename in appendix_line:
279             if not line_is_commented(appendix_line, filename):
280                 return filename
281
282 def get_appendix_from_filename(appendices, appendix_filename):
283     """Returns the first Appendix object with an appendix filename
284     ↪ that matches the incoming appendix_filename.
285     The Appendix objects are selected from an incoming list of
286     ↪ Appendix objects.
287
288     :param appendices: List of Appendix objects
289     :param appendix_filename: name of a latex appendix file, ends in
290         ↪ .tex,
291     """
292     for appendix in appendices:
293         if appendix_filename == appendix.appendix_filename:
294             return appendix
295
296 def get_compiled_notebook_paths(script_dir):
297     """Returns the list of jupyter notebook filepaths that were
298     ↪ compiled successfully and that are
299     included in the same dias this script (the src directory).
300
301     :param script_dir: absolute path of this file.
302     """
303     notebook_filepaths = get_filenames_in_dir(".ipynb", script_dir)
304     compiled_notebook_filepaths = []
305
306     # check if the jupyter notebooks were compiled
307     for notebook_filepath in notebook_filepaths:
308         # swap file extension

```

```

302     notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf"
303         ↪ ")
304
305     # check if file exists
306     if os.path.isfile(notebook_filepath):
307         compiled_notebook_filepaths.append(notebook_filepath)
308
309     return compiled_notebook_filepaths
310
311 def get_list_of_appendix_files(
312     appendix_dir, absolute_notebook_filepaths,
313     ↪ absolute_python_filepaths
314 ):
315     """Returns a list of Appendix objects that contain all the
316     ↪ appendix files with .tex extension.
317
318     :param appendix_dir: Absolute path that contains the appendix .
319     ↪ tex files.
320     :param absolute_notebook_filepaths: List of absolute paths to the
321     ↪ compiled notebook pdf files.
322     :param absolute_python_filepaths: List of absolute paths to the
323     ↪ python files.
324     """
325     appendices = []
326     appendices_paths = get_filenames_in_dir(".tex", appendix_dir)
327
328     for appendix_filepath in appendices_paths:
329         appendix_type = "no_code"
330         appendix_filecontent = read_file(appending_filepath)
331         line_nr_python_file_inclusion = get_line_of_latex_command(
332             appendix_filecontent, "\pythonexternal{"
333         )
334         line_nr_notebook_file_inclusion = get_line_of_latex_command(
335             appendix_filecontent, "\includepdf[pages="
336         )
337         if line_nr_python_file_inclusion > -1:
338             appendix_type = "python"
339             # get python filename
340             line = appendix_filecontent[line_nr_python_file_inclusion
341             ↪ ]
342             filename = get_filename_from_latex_inclusion_command(
343                 line, ".py", "\pythonexternal{"
344             )
345             appendices.append(
346                 Appendix(
347                     appendix_filepath,
348                     appendix_filecontent,
349                     appendix_type,
350                     filename,
351                     line,
352                 )
353             )
354         if line_nr_notebook_file_inclusion > -1:
355             appendix_type = "notebook"
356             line = appendix_filecontent[
357                 ↪ line_nr_notebook_file_inclusion]
358             filename = get_filename_from_latex_inclusion_command(
359                 line, ".pdf", "\includepdf[pages="
360             )
361             appendices.append(
362                 Appendix(
363                     appendix_filepath,

```

```

356         appendix_filecontent,
357         appendix_type,
358         filename,
359         line,
360     )
361 )
362 else:
363     appendices.append(
364         Appendix(appendix_filepath, appendix_filecontent,
365                 ↪ appendix_type)
366     )
367 return appendices
368
369 def get_filename_from_latex_inclusion_command(
370     appendix_line, extension, start_substring
371 ):
372     """returns the code/notebook filename in a latex command which
373     ↪ includes that code in an appendix.
374     The inclusion command includes a python code or jupyter notebook
375     ↪ pdf.
376
377     :param appendix_line: :Line of latex code (in particular expected
378     ↪ to be the latex code from an appendix.).
379     :param extension: The file extension of the file that is sought
380     ↪ in the appendix line. Either ".py" or ".pdf".
381     :param start_substring: The substring that characterises the
382     ↪ latex inclusion command.
383     """
384     start_index = appendix_line.index(start_substring)
385     end_index = appendix_line.index(extension)
386     return get_filename_from_dir(
387         appendix_line[start_index : end_index + len(extension)]
388     )
389
390 def get_filenames_in_dir(extension, path, excluded_files=None):
391     """Returns a list of the relative paths to all files within the
392     ↪ some path that match
393     the given file extension.
394
395     :param extension: The file extension of the file that is sought
396     ↪ in the appendix line. Either ".py" or ".pdf".
397     :param path: Absolute filepath in which files are being sought.
398     :param excluded_files: (Default value = None) Files that will not
399     ↪ be included even if they are found.
400     """
401     filepaths = []
402     for r, d, f in os.walk(path):
403         for file in f:
404             if file.endswith(extension):
405                 if (excluded_files is None) or (
406                     (not excluded_files is None) and (not file in
407                     ↪ excluded_files)
408                 ):
409                     filepaths.append(r + "/" + file)
410     return filepaths
411
412 def get_code_files_already_included_in_appendices(
413     absolute_code_filepaths, appendix_dir, extension, project_nr,
414     ↪ root_dir

```

```

407 ):
408     """Returns a list of code filepaths that are already properly
         ↳ included the latex appendix files of this project.
409
410     :param absolute_code_filepaths: List of absolute paths to the
         ↳ code files (either python files or compiled jupyter
         ↳ notebook pdfs).
411     :param appendix_dir: Absolute path that contains the appendix .
         ↳ tex files.
412     :param extension: The file extension of the file that is sought
         ↳ in the appendix line. Either ".py" or ".pdf".
413     :param project_nr: The number indicating which project this code
         ↳ pertains to.
414     :param root_dir: The root directory of this repository.
415     """
416     appendix_files = get_filenames_in_dir(".tex", appendix_dir)
417     contained_codes = []
418     for code_filepath in absolute_code_filepaths:
419         for appendix_filepath in appendix_files:
420             appendix_filecontent = read_file(appendix_filepath)
421             line_nr = check_if_appendix_contains_file(
422                 appendix_filecontent, code_filepath, extension,
423                 ↳ project_nr, root_dir
424             )
425             if line_nr > -1:
426                 # add filepath to list of files that are already in
427                 ↳ the appendices
428                 contained_codes.append(
429                     Appendix_with_code(
430                         code_filepath,
431                         appendix_filepath,
432                         appendix_filecontent,
433                         line_nr,
434                         ".py",
435                     )
436                 )
437     return contained_codes
438
439 def check_if_appendix_contains_file(
440     appendix_content, code_filepath, extension, project_nr, root_dir
441 ):
442     """Scans an appendix content to determine whether it contains a
         ↳ substring that
443     includes a code file (of either python or compiled notebook=pdf
         ↳ extension).
444
445     :param appendix_content: content in an appendix latex file.
446     :param code_filepath: Absolute path to a code file (either python
         ↳ files or compiled jupyter notebook pdfs).
447     :param extension: The file extension of the file that is sought
         ↳ in the appendix line. Either ".py" or ".pdf".
448     :param project_nr: The number indicating which project this code
         ↳ pertains to.
449     :param root_dir: The root directory of this repository.
450     """
451     # convert code_filepath to the inclusion format in latex format
452     latex_relative_filepath = (
453         f"latex/project{project_nr}/../{code_filepath[len(root_dir
         ↳ ):]}"
454     )

```



```

454 latex_command = get_latex_inclusion_command(extension,
      ↳ latex_relative_filepath)
455 return get_line_of_latex_command(appendix_content, latex_command)
456
457
458 def get_line_of_latex_command(appendix_content, latex_command):
459     """Returns the line number of a latex command if it is found.
      ↳ Returns -1 otherwise.
460
461     :param appendix_content: content in an appendix latex file.
462     :param latex_command: A line of latex code. (Expected to come
      ↳ from some appendix)
463     """
464     # check if the file is in the latex code
465     line_nr = 0
466     for line in appendix_content:
467         if latex_command in line:
468             if line_is_commented(line, latex_command):
469                 commented = True
470             else:
471                 return line_nr
472         line_nr = line_nr + 1
473     return -1
474
475
476 def line_is_commented(line, target_substring):
477     """Returns True if a latex code line is commented, returns False
      ↳ otherwise
478
479     :param line: A line of latex code that contains a relevant
      ↳ command (target substring).
480     :param target_substring: Used to determine whether the command
      ↳ that is found is commented or not.
481     """
482     left_of_command = line[: line.rfind(target_substring)]
483     if "%" in left_of_command:
484         return True
485     return False
486
487
488 def get_latex_inclusion_command(extension,
      ↳ latex_relative_filepath_to_codefile):
489     """Creates and returns a latex command that includes either a
      ↳ python file or a compiled jupyter
490     notebook pdf (whereever the command is placed). The command is
      ↳ intended to be placed in the appendix.
491
492     :param extension: The file extension of the file that is sought
      ↳ in the appendix line. Either ".py" or ".pdf".
493     :param latex_relative_filepath_to_codefile: The latex compilation
      ↳ requires a relative path towards code files
494     that are included. Therefore, a relative path towards the code is
      ↳ given.
495     """
496     if extension == ".py":
497         left = "\pythonexternal{"
498         right = "}"
499         latex_command = f"{left}{latex_relative_filepath_to_codefile
      ↳ }{right}"
500     elif extension == ".ipynb":
501
502         left = "\includepdf[pages=-]{"

```

```

503         right = "}"
504         latex_command = f"{{left}}{{latex_relative_filepath_to_codefile
505         ↪ }}{{right}}"
506     return latex_command
507
508 def read_file(filepath):
509     """Reads content of a file and returns it as a list of strings,
510     ↪ with one string per line.
511
512     :param filepath: path towards the file that is being read.
513     """
514     with open(filepath) as f:
515         content = f.readlines()
516     return content
517
518 def get_code_files_not_yet_included_in_appendices(
519     code_filepaths, contained_codes, extension
520 ):
521     """Returns a list of filepaths that are not yet properly included
522     ↪ in some appendix of this project.
523
524     :param code_filepath: Absolute path to all the code files in
525     ↪ this project (source directory).
526     (either python files or compiled jupyter notebook pdfs).
527     :param contained_codes: list of Appendix objects that include
528     ↪ either python files or compiled jupyter notebook pdfs,
529     ↪ which
530     are already included in the appendix tex files. (Does not care
531     ↪ whether those appendices are also actually
532     included in the main or not.)
533     :param extension: The file extension of the file that is sought
534     ↪ in the appendix line. Either ".py" or ".pdf".
535     """
536     contained_filepaths = list(
537         map(lambda contained_file: contained_file.code_filepath,
538             ↪ contained_codes)
539     )
540     not_contained = []
541     for filepath in code_filepaths:
542         if not filepath in contained_filepaths:
543             not_contained.append(filepath)
544     return not_contained
545
546 def create_appendices_with_code(
547     appendix_dir, code_filepaths, extension, project_nr, root_dir
548 ):
549     """Creates the latex appendix files in with relevant codes
550     ↪ included.
551
552     :param appendix_dir: Absolute path that contains the appendix .
553     ↪ tex files.
554     :param code_filepaths: Absolute path to code files that are not
555     ↪ yet included in an appendix
556     (either python files or compiled jupyter notebook pdfs).
557     :param extension: The file extension of the file that is sought
558     ↪ in the appendix line. Either ".py" or ".pdf".
559     :param project_nr: The number indicating which project this code
560     ↪ pertains to.
561     :param root_dir: The root directory of this repository.

```

```

551 """
552 appendix_filenames = []
553 appendix_reference_index = (
554     get_index_of_auto_generated_appendices(appendix_dir,
555         ↪ extension) + 1
556 )
557 for code_filepath in code_filepaths:
558     latex_relative_filepath = (
559         f"latex/project{project_nr}/../../{code_filepath[len(
560             ↪ root_dir):]}"
561     )
562     code_path_from_latex_main_path = f"../../{code_filepath[len(
563         ↪ root_dir):]}"
564     content = []
565     filename = get_filename_from_dir(code_filepath)
566
567     content = create_section(appendix_reference_index, filename,
568         ↪ content)
569     content = add_include_code_in_appendix(
570         content,
571         code_filepath,
572         code_path_from_latex_main_path,
573         extension,
574         latex_relative_filepath,
575         project_nr,
576         root_dir,
577     )
578
579     print(f"content={content}")
580
581     overwrite_content_to_file(
582         content,
583         f"{appendix_dir}Auto_generated_{extension[1:]}_App{
584             ↪ appendix_reference_index}.tex",
585         False,
586     )
587     appendix_filenames.append(
588         f"Auto_generated_{extension[1:]}_App{
589             ↪ appendix_reference_index}.tex"
590     )
591     appendix_reference_index = appendix_reference_index + 1
592 return appendix_filenames
593
594 def add_include_code_in_appendix(
595     content,
596     code_filepath,
597     code_path_from_latex_main_path,
598     extension,
599     latex_relative_filepath,
600     project_nr,
601     root_dir,
602 ):
603     """Includes the latex code that includes code in the script.
604
605     :param content: The latex content that is being written to an
606         ↪ appendix.
607     :param code_path_from_latex_main_path: the path to the code as
608         ↪ seen from the folder that contains main.tex.
609     :param extension: The file extension of the file that is sought
610         ↪ in the appendix line. Either ".py" or ".pdf".

```

```

604 :param latex_relative_filepath_to_codefile: The latex compilation
        ↳ requires a relative path towards code files
605 that are included. Therefore, a relative path towards the code is
        ↳ given.
606 """
607 print(f"before={content}")
608 # TODO: append if exists}
609 content.append(
610     f"\IfFileExists{{latex/project{project_nr}/../../{
        ↳ code_filepath[len(root_dir):]}}}{{"
611 )
612 # append current line
613 content.append(get_latex_inclusion_command(extension,
        ↳ latex_relative_filepath))
614 # TODO: append {}
615 content.append(f"}}{{{")
616 # TODO: code_path_from latex line
617 content.append(
618     get_latex_inclusion_command(extension,
        ↳ code_path_from_latex_main_path)
619 )
620 # TODO: add closing bracket }
621 content.append(f"}}")
622 print(f"after={content}")
623 return content
624
625
626 def get_index_of_auto_generated_appendices(appendix_dir, extension):
627     """Returns the maximum index of auto generated appendices of
628     a specific extension type.
629
630     :param extension: The file extension of the file that is sought
        ↳ in the appendix line. Either ".py" or ".pdf".
631     :param appendix_dir: Absolute path that contains the appendix .
        ↳ tex files.
632     """
633     max_index = -1
634     appendices =
        ↳ get_auto_generated_appendix_filenames_of_specific_extension
        ↳ (
635         appendix_dir, extension
636     )
637     for appendix in appendices:
638         substring = f"Auto_generated_{extension[1:]}_App"
639         # remove left of index
640         remainder = appendix.rfind(substring) + len(
        ↳ substring) :]
641         # remove right of index
642         index = int(remainder[:-4])
643         if index > max_index:
644             max_index = index
645     return max_index
646
647
648 def get_auto_generated_appendix_filenames_of_specific_extension(
649     appendix_dir, extension
650 ):
651     """Returns the list of auto generated appendices of
652     a specific extension type.
653
654     :param extension: The file extension of the file that is sought
        ↳ in the appendix line. Either ".py" or ".pdf".

```

```

655 :param appendix_dir: Absolute path that contains the appendix .
        ↳ tex files.
656 """
657 appendices_of_extension_type = []
658
659 # get all appendices
660 appendix_files = get_filenames_in_dir(".tex", appendix_dir)
661
662 # get appendices of particular extension type
663 for appendix_filepath in appendix_files:
664     right_of_slash = appendix_filepath[appendix_filepath.rfind("/")
        ↳ ") + 1 :]
665     if (
666         right_of_slash[: 15 + len(extension) - 1]
667         == f"Auto_generated_{extension[1:]}"
668     ):
669         appendices_of_extension_type.append(appending_filepath)
670 return appendices_of_extension_type
671
672 def create_section(appendix_reference_index, code_filename, content):
673     """Creates the header of a latex appendix file, such that it
674         ↳ contains a section that
675         indicates the section is an appendix, and indicates which python
676         ↳ or notebook file is
        being included in that appendix.
677
678     :param appendix_reference_index: A counter that is used in the
        ↳ label to ensure the appendix section labels are unique.
679     :param code_filename: file name of the code file that is included
680     :param content: A list of strings that make up the appendix, with
        ↳ one line per element.
681     """
682     # write section
683     left = "\section{Appendix "
684     middle = code_filename.replace("-", "\-")
685     right = "}\label{app:"
686     end = "}" # TODO: update appendix reference index
687     content.append(f"{left}{middle}{right}{appendix_reference_index}{"
        ↳ end}")
688     return content
689
690 def overwrite_content_to_file(content, filepath, content_has_newlines
        ↳ =True):
691     """Writes a list of lines of tex code from the content argument
692         ↳ to a .tex file
        using overwriting method. The content has one line per element.
693
694     :param content: The content that is being written to file.
695     :param filepath: Path towards the file that is being read.
696     :param content_has_newlines: (Default value = True)
697     """
698     with open(filepath, "w") as f:
699         for line in content:
700             if content_has_newlines:
701                 f.write(line)
702             else:
703                 f.write(line + "\n")
704
705 def get_appendix_tex_code(main_latex_filename):

```

```

708 """gets the latex appendix code from the main tex file.
709
710 :param main_latex_filename: Name of the main latex document of
711     ↳ this project number
712 """
713 main_tex_code = read_file(main_latex_filename)
714 start = "\\begin{appendices}"
715 end = "\\end{appendices}"
716 start_index = get_index_of_substring_in_list(main_tex_code, start
717     ↳ ) + 1
718 end_index = get_index_of_substring_in_list(main_tex_code, end)
719 return main_tex_code, start_index, end_index, main_tex_code[
720     ↳ start_index:end_index]
721
722 def get_index_of_substring_in_list(lines, target_substring):
723     """Returns the index of the line in which the first character of
724     ↳ a latex substring if it is found
725     uncommented in the incoming list.
726
727 :param lines: List of lines of latex code.
728 :param target_substring: Some latex command/code that is sought
729     ↳ in the incoming text.
730 """
731 for i in range(0, len(lines)):
732     if target_substring in lines[i]:
733         if not line_is_commented(lines[i], target_substring):
734             return i
735
736 def update_appendix_tex_code(appendix_filename, is_from_root_dir,
737     ↳ project_nr):
738     """Returns the latex command that includes an appendix .tex file
739     ↳ in an appendix environment
740     as can be used in the main tex file.
741
742 :param appendix_filename: Name of the appendix that is included
743     ↳ by the generated command.
744 :param project_nr: The number indicating which project this code
745     ↳ pertains to.
746 """
747 if is_from_root_dir:
748     left = f"\\input{{latex/project{project_nr}}/"
749 else:
750     left = "\\input{"
751     middle = "Appendices/"
752     right = "} \\newpage\\n"
753 return f"{left}{middle}{appendix_filename}{right}"
754
755 def substitute_appendix_code(
756     end_index, main_tex_code, start_index,
757     ↳ updated_appendices_tex_code
758 ):
759     """Replaces the old latex code that included the appendices in
760     ↳ the main.tex file with the new latex
761     commands that include the appendices in the latex report.
762
763 :param end_index: Index at which the appendix section ends right
764     ↳ before the latex \\end{appendix} line,
765 :param main_tex_code: The code that is saved in the main .tex
766     ↳ file.

```

```

757 :param start_index: Index at which the appendix section starts
758     ↳ right after the latex \begin{appendix} line,
759 :param updated_appendices_tex_code: The newly created code that
760     ↳ includes all the relevant appendices.
761 (relevant being (in order): manually created appendices, python
762     ↳ codes, pdfs of compiled jupyter notebooks).
763 """
764 updated_main_tex_code = (
765     main_tex_code[0:start_index]
766     + updated_appendices_tex_code
767     + main_tex_code[end_index:]
768 )
769 print(f"start_index={start_index}")
770 return updated_main_tex_code
771
772 def get_filename_from_dir(path):
773     """Returns a filename from an absolute path to a file.
774
775     :param path: path to a file of which the name is queried.
776     """
777     return path[path.rfind("/") + 1 :]
778
779 def get_script_dir():
780     """returns the directory of this script regardless of from which
781     ↳ level the code is executed"""
782     return os.path.dirname(__file__)
783
784 class Appendix_with_code:
785     """stores in which appendix file and accompanying line number in
786     ↳ the appendix in which a code file is
787     already included. Does not take into account whether this
788     ↳ appendix is in the main tex file or not
789     """
790
791     def __init__(
792         self,
793         code_filepath,
794         appendix_filepath,
795         appendix_content,
796         file_line_nr,
797         extension,
798     ):
799         self.code_filepath = code_filepath
800         self.appendix_filepath = appendix_filepath
801         self.appendix_content = appendix_content
802         self.file_line_nr = file_line_nr
803         self.extension = extension
804
805 class Appendix:
806     """stores in appendix files and type of appendix."""
807
808     def __init__(
809         self,
810         appendix_filepath,
811         appendix_content,
812         appendix_type,
813         code_filename=None,
814         appendix_inclusion_line=None,
815     ):

```

```
813 ):
814     self.appendix_filepath = appendix_filepath
815     self.appendix_filename = get_filename_from_dir(self.
        ↳ appendix_filepath)
816     self.appendix_content = appendix_content
817     self.appendix_type = appendix_type # TODO: perform
        ↳ validation of input values
818     self.code_filename = code_filename
819     self.appendix_inclusion_line = appendix_inclusion_line
```

E Appendix Model_bottom_up.py

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 from matplotlib import pyplot as plt
4 from matplotlib import lines
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 from .Plot_to_tex import Plot_to_tex as plt_tex
9
10
11 class Model_bottom_up:
12     def __init__(self):
13         pass
14
15     def addTwo(self, x):
16         """adds two to the incoming integer and returns the result of
17             ↪ the computation."""
18         return x + 2
```

F Appendix Model_top_down.py

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 from matplotlib import pyplot as plt
4 from matplotlib import lines
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 from .Plot_to_tex import Plot_to_tex as plt_tex
9
10
11 class Model_bottom_up:
12     def __init__(self):
13         pass
14
15     def addTwo(self, x):
16         """adds two to the incoming integer and returns the result of
17             ↪ the computation."""
18         return x + 2
```

G Appendix Model_value_theory.py

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 from matplotlib import pyplot as plt
4 from matplotlib import lines
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 from .Plot_to_tex import Plot_to_tex as plt_tex
9
10
11 class Model_bottom_up:
12     def __init__(self):
13         pass
14
15     def addTwo(self, x):
16         """adds two to the incoming integer and returns the result of
17             ↪ the computation."""
18         return x + 2
```

H Appendix Plot_to_tex.py

```
1  ### Call this from another file, for project 11, question 3b:
2  ### from Plot_to_tex import Plot_to_tex as plt_tex
3  ### multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints),
   ↪ dtype=int); # actually fill with data
4  ### lineLabels = [] # add a label for each dataseries
5  ### plt_tex.plotMultipleLines(plt_tex,single_x_series,
   ↪ multiple_y_series,"x-axis label [units]","y-axis label [units
   ↪ ]",lineLabels,"3b",4,11)
6  ### 4b=filename
7  ### 4 = position of legend, e.g. top right.
8  ###
9  ### For a single line, use:
10 ### plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),
   ↪ dataseries,"x-axis label [units]","y-axis label [units]",
   ↪ lineLabel,"3b",4,11)
11
12 ### You can also plot a table directly into latex, see
   ↪ example_create_a_table(..)
13 ###
14 ### Then put it in latex with for example:
15 ### \begin{table}[H]
16 ###     \centering
17 ###     \caption{Results some computation.}\label{tab:some_computation
   ↪ }
18 ###     \begin{tabular}{|c|c|} % remember to update this to show all
   ↪ columns of table
19 ###         \hline
20 ###         \input{latex/project3/tables/q2.txt}
21 ###     \end{tabular}
22 ### \end{table}
23 import random
24 from matplotlib import lines
25 import matplotlib.pyplot as plt
26 import numpy as np
27 import os
28
29
30 class Plot_to_tex:
31     def __init__(self):
32         self.script_dir = self.get_script_dir()
33         print("Created main")
34
35     # plot graph (legendPosition = integer 1 to 4)
36     def plotSingleLine(
37         self,
38         x_path,
39         y_series,
40         x_axis_label,
41         y_axis_label,
42         label,
43         filename,
44         legendPosition,
45         project_nr,
46     ):
47         fig = plt.figure()
48         ax = fig.add_subplot(111)
49         ax.plot(x_path, y_series, c="b", ls="-", label=label,
   ↪ fillstyle="none")
50         plt.legend(loc=legendPosition)
51         plt.xlabel(x_axis_label)
```

```

52     plt.ylabel(y_axis_label)
53     plt.savefig(
54         os.path.dirname(__file__)
55         + "/../../../latex/project"
56         + str(project_nr)
57         + "/Images/"
58         + filename
59         + ".png"
60     )
61
62     #         plt.show();
63
64     # plot graphs
65     def plotMultipleLines(
66         self, x, y_series, x_label, y_label, label, filename,
67         ↪ legendPosition, project_nr
68     ):
69         fig = plt.figure()
70         ax = fig.add_subplot(111)
71
72         # generate colours
73         cmap = self.get_cmap(len(y_series[:, 0]))
74
75         # generate line types
76         lineTypes = self.generateLineTypes(y_series)
77
78         for i in range(0, len(y_series)):
79             # overwrite linetypes to single type
80             lineTypes[i] = "_"
81             ax.plot(
82                 x,
83                 y_series[i, :],
84                 ls=lineTypes[i],
85                 label=label[i],
86                 fillstyle="none",
87                 c=cmap(i),
88             )
89             # color
90
91         # configure plot layout
92         plt.legend(loc=legendPosition)
93         plt.xlabel(x_label)
94         plt.ylabel(y_label)
95         plt.savefig(
96             os.path.dirname(__file__)
97             + "/../../../latex/project"
98             + str(project_nr)
99             + "/Images/"
100             + filename
101             + ".png"
102         )
103
104         print(f"plotted lines")
105
106     # Generate random line colours
107     # Source: https://stackoverflow.com/questions/14720331/how-to-
108     ↪ generate-random-colors-in-matplotlib
109     def get_cmap(n, name="hsv"):
110         """Returns a function that maps each index in 0, 1, ..., n-1
111             ↪ to a distinct
112             RGB color; the keyword argument name must be a standard mpl
113             ↪ colormap name."""

```

```

110         return plt.cm.get_cmap(name, n)
111
112     def generateLineTypes(y_series):
113         # generate varying linetypes
114         typeOfLines = list(lines.lineStyles.keys())
115
116         while len(y_series) > len(typeOfLines):
117             typeOfLines.append("-.")
118
119         # remove void lines
120         for i in range(0, len(y_series)):
121             if typeOfLines[i] == "None":
122                 typeOfLines[i] = "-"
123             if typeOfLines[i] == ":":
124                 typeOfLines[i] = ":"
125             if typeOfLines[i] == " ":
126                 typeOfLines[i] = "--"
127         return typeOfLines
128
129     # Create a table with: table_matrix = np.zeros((4,4),dtype=object
130     ↪ ) and pass it to this object
131     def put_table_in_tex(self, table_matrix, filename, project_nr):
132         cols = np.shape(table_matrix)[1]
133         format = "%s"
134         for col in range(1, cols):
135             format = format + " & %s"
136         format = format + ""
137         plt.savetxt(
138             os.path.dirname(__file__)
139             + "/../../../latex/project"
140             + str(project_nr)
141             + "/tables/"
142             + filename
143             + ".txt",
144             table_matrix,
145             delimiter=" & ",
146             fmt=format,
147             newline=" \\\\ \hline \n",
148         )
149
150     # replace this with your own table creation and then pass it to
151     ↪ put_table_in_tex(..)
152     def example_create_a_table(self):
153         project_nr = "1"
154         table_name = "example_table_name"
155         rows = 2
156         columns = 4
157         table_matrix = np.zeros((rows, columns), dtype=object)
158         table_matrix[:, :] = "" # replace the standard zeros with
159         ↪ empty cell
160         print(table_matrix)
161         for column in range(0, columns):
162             for row in range(0, rows):
163                 table_matrix[row, column] = row + column
164         table_matrix[1, 0] = "example"
165         table_matrix[0, 1] = "grid sizes"
166
167         self.put_table_in_tex(table_matrix, table_name, project_nr)
168
169     def get_script_dir(self):
170         """returns the directory of this script regardless of from
171         ↪ which level the code is executed"""

```

```
168         return os.path.dirname(__file__)
169
170
171 if __name__ == "__main__":
172     main = Plot_to_tex()
173     main.example_create_a_table()
```
