Example to plot directly into latex

19-10-2019

1 Introduction

Welcome, this document presents the roadmap for the TruCol consultancy as well as the estimated costs of the accompanying human labour. The objective of this document is to provide some basic insight into the order of magnitude of the costs of developing a sustainable variant of the TruCol consultancy such that it is able to generate returns for its potential investors.

2 Assumptions

3 Model Description

4 Results

5 Sensitivity Analysis

6 Discussion

7 Conclusion

A Appendix __main__.py

```python
import os
from .Main import Main
from .Compile_gantt_locally import compile_gantt_locally

print(f"Hi, I'll be running the main code, and I'll let you know when
    ↪  I'm done.")
project_nr = 1
main = Main()

# compile the gantt chart locally
compile_gantt_locally(main.relative_src_filepath, main.
    ↪ plant_uml_java_filename)

# export the code to latex
main.export_code_to_latex(project_nr)

# compile the latex report
main.compile_latex_report(project_nr)

print(f"Done.")
```

B   Appendix Main.py

```python
# Example code that creates plots directly in report
# Code is an implementation of a genetic algorithm
import random
from matplotlib import pyplot as plt
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np

from .Compile_latex import Compile_latex
from .Plot_to_tex import Plot_to_tex as plt_tex
from .Export_code_to_latex import export_code_to_latex

# define global variables for genetic algorithm example
string_length = 100
mutation_chance = 1.0 / string_length
max_iterations = 1500


class Main:
    def __init__(self):
        self.project_nr=1
        self.plant_uml_java_filename="plantuml.jar"
        self.relative_src_filepath=f"code/project{self.project_nr}/
            src/"
        self.relative_plant_uml_java_filepath=f"code/project{self.
            project_nr}/src/{self.plant_uml_java_filename}"

    def export_code_to_latex(self, project_nr):
        export_code_to_latex("main.tex", project_nr)

    def compile_latex_report(self, project_nr):
        """compiles latex code to pdf"""
        compile_latex = Compile_latex(project_nr, "main.tex")

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
            the computation."""
        return x + 2


if __name__ == "__main__":
    # initialize main class
    main = Main()
```

C Appendix Activity.py

```python
# The bottom up model that computes the TAM and TSM
import random
import numpy as np

from .Plot_to_tex import Plot_to_tex as plt_tex


class Activity:
    def __init__(self, description, duration,  children=None,color=
        None, parent=None,starts_at_child_nr_start=None,
        starts_at_child_nr_end=None):
        self.parent=parent
        self.children=children
        self.description=description
        self.tag=tag
        self.duration=duration
        if (is None self.parent):
            if (is None color):
                raise Exception("Parent activity needs a colour")
            self.colour=colour
        else:
            self.colour=parent.colour

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
            the computation."""
        return x + 2
```

D Appendix Compile_gantt_locally.py

```python
# pip install plantuml

# to compile locally:
# export  PLANTUML_LIMIT_SIZE=8192
# java -jar plantuml.jar -verbose sequenceDiagram.txt

# To compile with server:
from plantuml import PlantUML
import os
import subprocess
from os.path import abspath
from shutil import copyfile

def compile_gantt_locally(relative_src_filepath,
    plant_uml_java_filename):
    relative_plant_uml_java_filepath= f"{relative_src_filepath}{
        plant_uml_java_filename}"
    print(f'relative_plant_uml_java_filepath={
        relative_plant_uml_java_filepath}')
    if check_if_java_file_exists(relative_plant_uml_java_filepath):
        print('FOUND')
        os.environ["PLANTUML_LIMIT_SIZE"] = "8192"
        run_bash_command(f"java -jar {
            relative_plant_uml_java_filepath} -verbose {
            relative_src_filepath}Diagrams/example_gantt.uml")
        #export_gantt_to_latex(filename)
    pass


def check_if_java_file_exists(relative_filepath):
    if os.path.isfile(relative_filepath):
        return True
    else:
        raise Exception(f"File:{relative_filepath} is not accessible"
            )

def run_bash_command(bashCommand):
    subprocess.Popen(bashCommand, shell=True)

def export_gantt_to_latex(filename):
    if filename.endswith(".png"):
        copyfile(abspath(f'./Diagrams/{filename}'),abspath(f'./../../
            latex/researchplan/Images/{filename}'))
```

E   Appendix Compile_latex.py

```python
# runs a jupyter notebook and converts it to pdf

import os
import shutil
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor


class Compile_latex:
    def __init__(self, project_nr, latex_filename):
        self.script_dir = self.get_script_dir()
        relative_dir = f"latex/project{project_nr}/"
        self.compile_latex(relative_dir, latex_filename)
        self.clean_up_after_compilation(latex_filename)
        self.move_pdf_into_latex_dir(relative_dir, latex_filename)

    # runs jupyter notebook
    def compile_latex(self, relative_dir, latex_filename):
        os.system(f"pdflatex {relative_dir}{latex_filename}")

    def clean_up_after_compilation(self, latex_filename):
        latex_filename_without_extention = latex_filename[:-4]
        print(f"latex_filename_without_extention={
            latex_filename_without_extention}")
        self.delete_file_if_exists(f"{
            latex_filename_without_extention}.aux")
        self.delete_file_if_exists(f"{
            latex_filename_without_extention}.log")
        self.delete_file_if_exists(f"texput.log")

    def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
        pdf_filename = f"{latex_filename[:-4]}.pdf"
        destination = f"{self.get_script_dir()}/../../../{
            relative_dir}{pdf_filename}"

        try:
            shutil.move(pdf_filename, destination)
        except:
            print("Error while moving file ", pdf_filename)

    def delete_file_if_exists(self, filename):
        try:
            os.remove(filename)
        except:
            print(
                f"Error while deleting file: {filename} but that is
                    not too bad because the intention is for it to
                    not be there."
            )

    def get_script_dir(self):
        """returns the directory of this script regardles of from
            which level the code is executed"""
        return os.path.dirname(__file__)


if __name__ == "__main__":
    main = Compile_latex()
```

F Appendix Create_python_gantt.py

```python
# The bottom up model that computes the TAM and TSM
import random
import numpy as np

from .Activity import Activity


def create_python_gantt():
    activity=


def addTwo(self, x):
    """adds two to the incoming integer and returns the result of the
        ↪ computation."""
    return x + 2
```

G Appendix Export_code_to_latex.py

```python
# runs a jupyter notebook and converts it to pdf
import os
import shutil
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor


def export_code_to_latex(main_latex_filename, project_nr):
    """This function exports the python files and compiled pdfs of
        ↪ jupiter notebooks into the
    latex of the same project number. First it scans which appendices
        ↪  (without code, without
    notebooks) are already manually included in the main latex code.
        ↪ Next, all appendices
    that contain the python code are eiter found or created in the
        ↪ following order:
    First, the __main__.py file is included, followed by the main.py
        ↪ file, followed by all
    python code files in alphabetic order. After this, all the pdfs
        ↪ of the compiled notebooks
    are added in alphabetic order of filename. This order of
        ↪ appendices is overwritten in the
    main tex file.

    :param main_latex_filename: Name of the main latex document of
        ↪ this project number
    :param project_nr: The number  indicating which project this code
        ↪  pertains to.
    """
    script_dir = get_script_dir()
    relative_dir = f"latex/project{project_nr}/"
    appendix_dir = script_dir + "/../../../" + relative_dir + "
        ↪ Appendices/"
    path_to_main_latex_file = (
        f"{script_dir}/../../../{relative_dir}/{main_latex_filename}"
    )
    root_dir = script_dir[0 : script_dir.rfind(f"code/project{
        ↪ project_nr}")]

    # Get paths to files containing python code.
    python_filepaths = get_filenames_in_dir("py", script_dir, ["
        ↪ __init__.py"])
    compiled_notebook_pdf_filepaths = get_compiled_notebook_paths(
        ↪ script_dir)

    # Check which files are already included in the latex appendicess
        ↪ .
    python_files_already_included_in_appendices =
        ↪ get_code_files_already_included_in_appendices(
        python_filepaths, appendix_dir, ".py", project_nr, root_dir
    )
    notebook_pdf_files_already_included_in_appendices =
        ↪ get_code_files_already_included_in_appendices(
        compiled_notebook_pdf_filepaths, appendix_dir, ".ipynb",
            ↪ project_nr, root_dir,
    )

    # Get which appendices are still missing.
    missing_python_files_in_appendices =
        ↪ get_code_files_not_yet_included_in_appendices(
```

```python
        python_filepaths, python_files_already_included_in_appendices
            ↪ , ".py"
    )
    missing_notebook_files_in_appendices =
        ↪ get_code_files_not_yet_included_in_appendices(
        compiled_notebook_pdf_filepaths,
        notebook_pdf_files_already_included_in_appendices,
        ".pdf",
    )

    # Create the missing appendices.
    created_python_appendix_filenames = create_appendices_with_code(
        appendix_dir, missing_python_files_in_appendices, ".py",
            ↪ project_nr, root_dir
    )

    created_notebook_appendix_filenames = create_appendices_with_code
        ↪ (
        appendix_dir,
        missing_notebook_files_in_appendices,
        ".ipynb",
        project_nr,
        root_dir,
    )

    appendices = get_list_of_appendix_files(
        appendix_dir, compiled_notebook_pdf_filepaths,
            ↪ python_filepaths
    )

    main_tex_code, start_index, end_index, appendix_tex_code =
        ↪ get_appendix_tex_code(
        path_to_main_latex_file
    )

    # assumes non-included non-code appendices should not be included
        ↪ :
    # overwrite the existing appendix lists with the current appendix
        ↪  list.
    (
        non_code_appendices,
        main_non_code_appendix_inclusion_lines,
    ) = get_order_of_non_code_appendices_in_main(appendices,
        ↪ appendix_tex_code)

    python_appendix_filenames = list(
        map(
            lambda x: x.appendix_filename,
            filter_appendices_by_type(appendices, "python"),
        )
    )
    sorted_created_python_appendices = sort_python_appendices(
        filter_appendices_by_type(appendices, "python")
    )
    sorted_python_appendix_filenames = list(
        map(lambda x: x.appendix_filename,
            ↪ sorted_created_python_appendices)
    )

    notebook_appendix_filenames = list(
        map(
            lambda x: x.appendix_filename,
```

```python
                filter_appendices_by_type(appendices, "notebook"),
        )
    )
    sorted_created_notebook_appendices =
        ↪ sort_notebook_appendices_alphabetically(
        filter_appendices_by_type(appendices, "notebook")
    )
    sorted_notebook_appendix_filenames = list(
        map(lambda x: x.appendix_filename,
            ↪ sorted_created_notebook_appendices)
    )

    appendix_latex_code = create_appendices_latex_code(
        main_non_code_appendix_inclusion_lines,
        sorted_created_notebook_appendices,
        project_nr,
        sorted_created_python_appendices,
    )

    updated_main_tex_code = substitute_appendix_code(
        end_index, main_tex_code, start_index, appendix_latex_code
    )
    print(f'\n\n')
    print(f"updated_main_tex_code={updated_main_tex_code}")

    overwrite_content_to_file(updated_main_tex_code,
        ↪ path_to_main_latex_file)


def create_appendices_latex_code(
    main_non_code_appendix_inclusion_lines,
    notebook_appendices,
    project_nr,
    python_appendices,
):
    """Creates the latex code that includeds the appendices in the
        ↪ main latex file.

    :param main_non_code_appendix_inclusion_lines: latex code that
        ↪ includes the appendices that do not contain python code nor
        ↪  notebooks
    :param notebook_appendices: List of Appendix objects representing
        ↪  appendices that include the pdf files of compiled Jupiter
        ↪ notebooks
    :param project_nr: The number indicating which project this code
        ↪ pertains to.
    :param python_appendices: List of Appendix objects representing
        ↪ appendices that include the python code files.
    """
    main_appendix_inclusion_lines =
        ↪ main_non_code_appendix_inclusion_lines
    print(f"main_appendix_inclusion_lines={
        ↪ main_appendix_inclusion_lines}")

    appendices_of_all_types = [python_appendices, notebook_appendices
        ↪ ]

    print(f"\n\n")
    main_appendix_inclusion_lines.append(
        f"\IfFileExists{{latex/project{project_nr}/main.tex}}{{"
    )
    main_appendix_inclusion_lines = append_latex_inclusion_command(
```

```python
144            appendices_of_all_types, True, main_appendix_inclusion_lines,
             ↪    project_nr,
145        )
146        main_appendix_inclusion_lines.append(f"}}{{")
147        main_appendix_inclusion_lines = append_latex_inclusion_command(
148            appendices_of_all_types, False, main_appendix_inclusion_lines
             ↪    , project_nr,
149        )
150        #main_appendix_inclusion_lines.append(f"}}")
151        print(f"main_appendix_inclusion_lines={
             ↪    main_appendix_inclusion_lines}")
152        return main_appendix_inclusion_lines


155    def append_latex_inclusion_command(
156        appendices_of_all_types, is_from_root_dir,
             ↪    main_appendix_inclusion_lines, project_nr
157    ):
158        for appendix_type in appendices_of_all_types:
159            for appendix in appendix_type:
160                line = update_appendix_tex_code(
161                    appendix.appendix_filename, is_from_root_dir,
                     ↪    project_nr
162                )
163                print(f"appendix.appendix_filename={appendix.
                     ↪    appendix_filename}")
164                main_appendix_inclusion_lines.append(line)
165        return main_appendix_inclusion_lines


168    def filter_appendices_by_type(appendices, appendix_type):
169        """Returns the list of all appendices of a certain appendix type,
             ↪     from the incoming list of Appendix objects.
170
171        :param appendices: List of Appendix objects
172        :param appendix_type: Can consist of "no_code", "python", or "
             ↪    notebook" and indicates different appendix types
173        """
174        return_appendices = []
175        for appendix in appendices:
176            if appendix.appendix_type == appendix_type:
177                return_appendices.append(appendix)
178        return return_appendices


181    def sort_python_appendices(appendices):
182        """First puts __main__.py, followed by main.py followed by a-z
             ↪     code files.
183
184        :param appendices: List of Appendix objects
185        """
186        return_appendices = []
187        for appendix in appendices:  # first get appendix containing
             ↪    __main__.py
188            if (appendix.code_filename == "__main__.py") or (
189                appendix.code_filename == "__Main__.py"
190            ):
191                return_appendices.append(appendix)
192                appendices.remove(appendix)
193        for appendix in appendices:  # second get appendix containing
             ↪    main.py
194            if (appendix.code_filename == "main.py") or (
```

```python
                 appendix.code_filename == "Main.py"
        ):
            return_appendices.append(appendix)
            appendices.remove(appendix)
    return_appendices

    # Filter remaining appendices in order of a-z
    filtered_remaining_appendices = [
        i for i in appendices if i.code_filename is not None
    ]
    appendices_sorted_a_z = sort_appendices_on_code_filename(
        filtered_remaining_appendices
    )
    return return_appendices + appendices_sorted_a_z


def sort_notebook_appendices_alphabetically(appendices):
    """Sorts notebook appendix objects alphabetic order of their pdf
        ↪ filenames.

    :param appendices: List of Appendix objects
    ,"""
    return_appendices = []
    filtered_remaining_appendices = [
        i for i in appendices if i.code_filename is not None
    ]
    appendices_sorted_a_z = sort_appendices_on_code_filename(
        filtered_remaining_appendices
    )
    return return_appendices + appendices_sorted_a_z


def sort_appendices_on_code_filename(appendices):
    """Returns a list of Appendix objects that are sorted and  based
        ↪ on the property: code_filename.
    Assumes the incoming appendices only contain python files.

    :param appendices: List of Appendix objects
    """
    attributes = list(map(lambda x: x.code_filename, appendices))
    sorted_indices = sorted(range(len(attributes)), key=lambda k:
        ↪ attributes[k])
    sorted_list = []
    for i in sorted_indices:
        sorted_list.append(appendices[i])
    return sorted_list


def get_order_of_non_code_appendices_in_main(appendices,
    ↪ appendix_tex_code):
    """Scans the lines of appendices in the main code, and returns
        ↪ the lines
    of the appendices that do not contain code, in the order in which
        ↪  they were
    included in the main latex file.

    :param appendices: List of Appendix objects
    :param appendix_tex_code: latex code from the main latex file
        ↪ that includes the appendices
    """
    non_code_appendices = []
    non_code_appendix_lines = []
```

```python
        appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
        for line in appendix_tex_code:
            appendix_filename = get_filename_from_latex_appendix_line(
                ↪ appendices, line)

            # Check if line is not commented
            if not appendix_filename is None:
                if not line_is_commented(line, appendix_filename):
                    appendix = get_appendix_from_filename(appendices,
                        ↪ appendix_filename)
                    if appendix.appendix_type == "no_code":
                        non_code_appendices.append(appendix)
                        non_code_appendix_lines.append(line)
    return non_code_appendices, non_code_appendix_lines


def get_filename_from_latex_appendix_line(appendices, appendix_line):
    """Returns the first filename from a list of incoming filenames
        ↪ that
    occurs in a latex code line.

    :param appendices: List of Appendix objects
    :param appendix_line: latex code (in particular expected to be
        ↪ the code from main that is used to include appendix latex
        ↪ files.)
    """
    for filename in list(map(lambda appendix: appendix.
        ↪ appendix_filename, appendices)):
        if filename in appendix_line:
            if not line_is_commented(appendix_line, filename):
                return filename


def get_appendix_from_filename(appendices, appendix_filename):
    """Returns the first Appendix object with an appendix filename
        ↪ that matches the incoming appendix_filename.
    The Appendix objects are selected from an incoming list of
        ↪ Appendix objects.

    :param appendices: List of Appendix objects
    :param appendix_filename: name of a latex appendix file, ends in
        ↪ .tex,
    """
    for appendix in appendices:
        if appendix_filename == appendix.appendix_filename:
            return appendix


def get_compiled_notebook_paths(script_dir):
    """Returns the list of jupiter notebook filepaths that were
        ↪ compiled successfully and that are
    included in the same dias this script (the src directory).

    :param script_dir: absolute path of this file.
    """
    notebook_filepaths = get_filenames_in_dir(".ipynb", script_dir)
    compiled_notebook_filepaths = []

    # check if the jupyter notebooks were compiled
    for notebook_filepath in notebook_filepaths:

        # swap file extension
```

```python
302             notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf
                    ↪ ")

304             # check if file exists
305             if os.path.isfile(notebook_filepath):
306                 compiled_notebook_filepaths.append(notebook_filepath)
307         return compiled_notebook_filepaths


310 def get_list_of_appendix_files(
311     appendix_dir, absolute_notebook_filepaths,
            ↪ absolute_python_filepaths
312 ):
313     """Returns a list of Appendix objects that contain all the
            ↪ appendix files with .tex extension.

315     :param appendix_dir: Absolute path that contains the appendix .
            ↪ tex files.
316     :param absolute_notebook_filepaths: List of absolute paths to the
            ↪  compiled notebook pdf files.
317     :param absolute_python_filepaths: List of absolute paths to the
            ↪ python files.
318     """
319     appendices = []
320     appendices_paths = get_filenames_in_dir(".tex", appendix_dir)

322     for appendix_filepath in appendices_paths:
323         appendix_type = "no_code"
324         appendix_filecontent = read_file(appendix_filepath)
325         line_nr_python_file_inclusion = get_line_of_latex_command(
326             appendix_filecontent, "\pythonexternal{"
327         )
328         line_nr_notebook_file_inclusion = get_line_of_latex_command(
329             appendix_filecontent, "\includepdf[pages="
330         )
331         if line_nr_python_file_inclusion > -1:
332             appendix_type = "python"
333             # get python filename
334             line = appendix_filecontent[line_nr_python_file_inclusion
                    ↪ ]
335             filename = get_filename_from_latex_inclusion_command(
336                 line, ".py", "\pythonexternal{"
337             )
338             appendices.append(
339                 Appendix(
340                     appendix_filepath,
341                     appendix_filecontent,
342                     appendix_type,
343                     filename,
344                     line,
345                 )
346             )
347         if line_nr_notebook_file_inclusion > -1:
348             appendix_type = "notebook"
349             line = appendix_filecontent[
                    ↪ line_nr_notebook_file_inclusion]
350             filename = get_filename_from_latex_inclusion_command(
351                 line, ".pdf", "\includepdf[pages="
352             )
353             appendices.append(
354                 Appendix(
355                     appendix_filepath,
```

```python
                            appendix_filecontent,
                            appendix_type,
                            filename,
                            line,
                        )
                    )
                else:
                    appendices.append(
                        Appendix(appendix_filepath, appendix_filecontent,
                            ↪ appendix_type)
                    )
    return appendices


def get_filename_from_latex_inclusion_command(
    appendix_line, extension, start_substring
):
    """"returns the code/notebook filename in a latex command which
        ↪ includes that code in an appendix.
    The inclusion command includes a python code or jupiter notebook
        ↪ pdf.

    :param appendix_line: :Line of latex code (in particular expected
        ↪  to be the latex code from an appendix.).
    :param extension: The file extension of the file that is sought
        ↪ in the appendix line. Either ".py" or ".pdf".
    :param start_substring: The substring that characterises the
        ↪ latex inclusion command.
    """
    start_index = appendix_line.index(start_substring)
    end_index = appendix_line.index(extension)
    return get_filename_from_dir(
        appendix_line[start_index : end_index + len(extension)]
    )


def get_filenames_in_dir(extension, path, excluded_files=None):
    """"Returns a list of the relative paths to all files within the
        ↪ some path that match
    the given file extension.

    :param extension: The file extension of the file that is sought
        ↪ in the appendix line. Either ".py" or ".pdf".
    :param path: Absolute filepath in which files are being sought.
    :param excluded_files: (Default value = None) Files that will not
        ↪  be included even if they are found.
    """
    filepaths = []
    for r, d, f in os.walk(path):
        for file in f:
            if file.endswith(extension):
                if (excluded_files is None) or (
                    (not excluded_files is None) and (not file in
                        ↪ excluded_files)
                ):
                    filepaths.append(r + "/" + file)
    return filepaths


def get_code_files_already_included_in_appendices(
    absolute_code_filepaths, appendix_dir, extension, project_nr,
        ↪ root_dir
```

```python
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,
                   ↪ project_nr, root_dir
423             )
424             if line_nr > -1:
425                 # add filepath to list of files that are already in
                   ↪ the appendices
426                 contained_codes.append(
427                     Appendix_with_code(
428                         code_filepath,
429                         appendix_filepath,
430                         appendix_filecontent,
431                         line_nr,
432                         ".py",
433                     )
434                 )
435     return contained_codes
436
437
438 def check_if_appendix_contains_file(
439     appendix_content, code_filepath, extension, project_nr, root_dir
440 ):
441     """Scans an appendix content to determine whether it contains a
           ↪ substring that
442     includes a code file (of either python or compiled notebook=pdf
           ↪ extension).
443
444     :param appendix_content: content in an appendix latex file.
445     :param code_filepath: Absolute path to a code file (either python
           ↪  files or compiled jupyter notebook pdfs).
446     :param extension: The file extension of the file that is sought
           ↪ in the appendix line. Either ".py" or ".pdf".
447     :param project_nr: The number  indicating which project this code
           ↪  pertains to.
448     :param root_dir: The root directory of this repository.
449     """
450     # convert code_filepath to the inclusion format in latex format
451     latex_relative_filepath = (
452         f"latex/project{project_nr}/../../{code_filepath[len(root_dir
               ↪ ):]}"
453     )
```

```python
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 jupiter
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=-]{"
```

```python
            right = "}"
            latex_command = f"{left}{latex_relative_filepath_to_codefile
                ↪ }{right}"
        return latex_command


    def read_file(filepath):
        """Reads content of a file and returns it as a list of strings,
            ↪ with one string per line.

        :param filepath: path towards the file that is being read.
        """
        with open(filepath) as f:
            content = f.readlines()
        return content


    def get_code_files_not_yet_included_in_appendices(
        code_filepaths, contained_codes, extension
    ):
        """Returns a list of filepaths that are not yet properly included
            ↪  in some appendix of this project.

        :param code_filepath: Absolute path to all the code files in
            ↪ this project (source directory).
        (either python files or compiled jupyter notebook pdfs).
        :param contained_codes: list of  Appendix objects that include
            ↪ either python files or compiled jupyter notebook pdfs,
            ↪ which
        are already included in the appendix tex files. (Does not care
            ↪ whether those appendices are also actually
        included in the main or not.)
        :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
        """
        contained_filepaths = list(
            map(lambda contained_file: contained_file.code_filepath,
                ↪ contained_codes)
        )
        not_contained = []
        for filepath in code_filepaths:
            if not filepath in contained_filepaths:
                not_contained.append(filepath)
        return not_contained


    def create_appendices_with_code(
        appendix_dir, code_filepaths, extension, project_nr, root_dir
    ):
        """Creates the latex appendix files in with relevant codes
            ↪ included.

        :param appendix_dir: Absolute path that contains the appendix .
            ↪ tex files.
        :param code_filepaths: Absolute path to code files that are not
            ↪ yet included in an appendix
        (either python files or compiled jupyter notebook pdfs).
        :param extension: The file extension of the file that is sought
            ↪ in the appendix line. Either ".py" or ".pdf".
        :param project_nr: The number  indicating which project this code
            ↪  pertains to.
        :param root_dir: The root directory of this repository.
```

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

```python
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


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[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


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".
```

```python
        :param appendix_dir: Absolute path that contains the appendix .
            ↪ tex files.
        """
        appendices_of_extension_type = []

        # get all appendices
        appendix_files = get_filenames_in_dir(".tex", appendix_dir)

        # get appendices of particular extention type
        for appendix_filepath in appendix_files:
            right_of_slash = appendix_filepath[appendix_filepath.rfind("/
                ↪ ") + 1 :]
            if (
                right_of_slash[: 15 + len(extension) - 1]
                == f"Auto_generated_{extension[1:]}"
            ):
                appendices_of_extension_type.append(appendix_filepath)
        return appendices_of_extension_type


def create_section(appendix_reference_index, code_filename, content):
    """Creates the header of a latex appendix file, such that it
        ↪ contains a section that
    indicates the section is an appendix, and indicates which pyhon
        ↪ or notebook file is
    being included in that appendix.

    :param appendix_reference_index: A counter that is used in the
        ↪ label to ensure the appendix section labels are unique.
    :param code_filename: file name of the code file that is included
    :param content: A list of strings that make up the appendix, with
        ↪  one line per element.
    """
    # write section
    left = "\section{Appendix "
    middle = code_filename.replace("_", "\_")
    right = "}\label{app:"
    end = "}"  # TODO: update appendix reference index
    content.append(f"{left}{middle}{right}{appendix_reference_index}{
        ↪ end}")
    return content


def overwrite_content_to_file(content, filepath, content_has_newlines
    ↪ =True):
    """Writes a list of lines of tex code from the content argument
        ↪ to a .tex file
    using overwriting method. The content has one line per element.

    :param content: The content that is being written to file.
    :param filepath: Path towards the file that is being read.
    :param content_has_newlines:  (Default value = True)
    """
    with open(filepath, "w") as f:
        for line in content:
            if content_has_newlines:
                f.write(line)
            else:
                f.write(line + "\n")


def get_appendix_tex_code(main_latex_filename):
```

```python
708     """gets the latex appendix code from the main tex file.

709
710     :param main_latex_filename: Name of the main latex document of
        ↪ this project number
711     """
712     main_tex_code = read_file(main_latex_filename)
713     start = "\\begin{appendices}"
714     end = "\end{appendices}"
715     start_index = get_index_of_substring_in_list(main_tex_code, start
        ↪ ) + 1
716     end_index = get_index_of_substring_in_list(main_tex_code, end)
717     return main_tex_code, start_index, end_index, main_tex_code[
        ↪ start_index:end_index]

718

719
720 def get_index_of_substring_in_list(lines, target_substring):
721     """Returns the index of the line in which the first character of
        ↪ a latex substring if it is found
722     uncommented in the incoming list.

723
724     :param lines: List of lines of latex code.
725     :param target_substring: Some latex command/code that is sought
        ↪ in the incoming text.
726     """
727     for i in range(0, len(lines)):
728         if target_substring in lines[i]:
729             if not line_is_commented(lines[i], target_substring):
730                 return i

731

732
733 def update_appendix_tex_code(appendix_filename, is_from_root_dir,
    ↪ project_nr):
734     """Returns the latex command that includes an appendix .tex file
        ↪ in an appendix environment
735     as can be used in the main tex file.

736
737     :param appendix_filename: Name of the appendix that is included
        ↪ by the generated command.
738     :param project_nr: The number indicating which project this code
        ↪ pertains to.
739     """
740     if is_from_root_dir:
741         left = f"\input{{latex/project{project_nr}/"
742     else:
743         left = "\input{"
744     middle = "Appendices/"
745     right = "} \\newpage\n"
746     return f"{left}{middle}{appendix_filename}{right}"

747

748
749 def substitute_appendix_code(
750     end_index, main_tex_code, start_index,
        ↪ updated_appendices_tex_code
751 ):
752     """Replaces the old latex code that included the appendices in
        ↪ the main.tex file with the new latex
753     commands that include the appendices in the latex report.

754
755     :param end_index: Index at which the appendix section ends right
        ↪ before the latex \end{appendix} line,
756     :param main_tex_code: The code that is saved in the main .tex
        ↪ file.
```

```python
        :param start_index: Index at which the appendix section starts
            ↪ right after the latex \begin{appendix} line,
        :param updated_appendices_tex_code: The newly created code that
            ↪ includes all the relevant appendices.
        (relevant being (in order): manually created appendices, python
            ↪ codes, pdfs of compiled jupiter notebooks).
        """
        updated_main_tex_code = (
            main_tex_code[0:start_index]
            + updated_appendices_tex_code
            + main_tex_code[end_index:]
        )
        print(f"start_index={start_index}")
        return updated_main_tex_code


def get_filename_from_dir(path):
    """Returns a filename from an absolute path to a file.

    :param path: path to a file of which the name is queried.
    """
    return path[path.rfind("/") + 1 :]


def get_script_dir():
    """returns the directory of this script regardles of from which
        ↪ level the code is executed"""
    return os.path.dirname(__file__)


class Appendix_with_code:
    """stores in which appendix file and accompanying line number in
        ↪ the appendix in which a code file is
    already included. Does not take into account whether this
        ↪ appendix is in the main tex file or not
    """

    def __init__(
        self,
        code_filepath,
        appendix_filepath,
        appendix_content,
        file_line_nr,
        extension,
    ):
        self.code_filepath = code_filepath
        self.appendix_filepath = appendix_filepath
        self.appendix_content = appendix_content
        self.file_line_nr = file_line_nr
        self.extension = extension


class Appendix:
    """stores in appendix files and type of appendix."""

    def __init__(
        self,
        appendix_filepath,
        appendix_content,
        appendix_type,
        code_filename=None,
        appendix_inclusion_line=None,
```

```python
    ):
        self.appendix_filepath = appendix_filepath
        self.appendix_filename = get_filename_from_dir(self.
         ↪ appendix_filepath)
        self.appendix_content = appendix_content
        self.appendix_type = appendix_type  # TODO: perform
         ↪ validation of input values
        self.code_filename = code_filename
        self.appendix_inclusion_line = appendix_inclusion_line
```

H   Appendix Gantt.py

```python
# The bottom up model that computes the TAM and TSM
import random
import numpy as np

from .Plot_to_tex import Plot_to_tex as plt_tex
from .Create_python_gantt import create_python_gantt


class Gantt:
    def __init__(self):
        self.start_line="@startgantt"
        self.project_start_date="2021/07-22"
        self.closed_days=["saturday","sunday"]
        self.activities=create_python_gantt()
        self.end_line="@endgantt"



    def for

    def addTwo(self, x):
        """adds two to the incoming integer and returns the result of
            the computation."""
        return x + 2
```

I   Appendix Plot_to_tex.py

```python
### Call this from another file, for project 11, question 3b:
### from Plot_to_tex import Plot_to_tex as plt_tex
### multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints),
    ↪ dtype=int); # actually fill with data
### lineLabels = [] # add a label for each dataseries
### plt_tex.plotMultipleLines(plt_tex,single_x_series,
    ↪ multiple_y_series,"x-axis label [units]","y-axis label [units
    ↪ ]",lineLabels,"3b",4,11)
### 4b=filename
### 4 = position of legend, e.g. top right.
###
### For a single line, use:
### plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),
    ↪ dataseries,"x-axis label [units]","y-axis label [units]",
    ↪ lineLabel,"3b",4,11)

### You can also plot a table directly into latex, see
    ↪ example_create_a_table(..)
###
### Then put it in latex with for example:
###\begin{table}[H]
###     \centering
###     \caption{Results some computation.}\label{tab:some_computation
    ↪ }
###     \begin{tabular}{|c|c|} % remember to update this to show all
    ↪ columns of table
###         \hline
###         \input{latex/project3/tables/q2.txt}
###     \end{tabular}
###\end{table}
import random
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np
import os


class Plot_to_tex:
    def __init__(self):
        self.script_dir = self.get_script_dir()
        print("Created main")

    # plot graph (legendPosition = integer 1 to 4)
    def plotSingleLine(
        self,
        x_path,
        y_series,
        x_axis_label,
        y_axis_label,
        label,
        filename,
        legendPosition,
        project_nr,
    ):
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(x_path, y_series, c="b", ls="-", label=label,
            ↪ fillstyle="none")
        plt.legend(loc=legendPosition)
        plt.xlabel(x_axis_label)
```

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

```python
        return plt.cm.get_cmap(name, n)

    def generateLineTypes(y_series):
        # generate varying linetypes
        typeOfLines = list(lines.lineStyles.keys())

        while len(y_series) > len(typeOfLines):
            typeOfLines.append("-.")

        # remove void lines
        for i in range(0, len(y_series)):
            if typeOfLines[i] == "None":
                typeOfLines[i] = "-"
            if typeOfLines[i] == "":
                typeOfLines[i] = ":"
            if typeOfLines[i] == " ":
                typeOfLines[i] = "--"
        return typeOfLines

    # Create a table with: table_matrix = np.zeros((4,4),dtype=object
    #   ↪ ) and pass it to this object
    def put_table_in_tex(self, table_matrix, filename, project_nr):
        cols = np.shape(table_matrix)[1]
        format = "%s"
        for col in range(1, cols):
            format = format + " & %s"
        format = format + ""
        plt.savetxt(
            os.path.dirname(__file__)
            + "/../../../latex/project"
            + str(project_nr)
            + "/tables/"
            + filename
            + ".txt",
            table_matrix,
            delimiter=" & ",
            fmt=format,
            newline="  \\\\ \hline \n",
        )

    # replace this with your own table creation and then pass it to
    #   ↪ put_table_in_tex(..)
    def example_create_a_table(self):
        project_nr = "1"
        table_name = "example_table_name"
        rows = 2
        columns = 4
        table_matrix = np.zeros((rows, columns), dtype=object)
        table_matrix[:, :] = ""  # replace the standard zeros with
        #   ↪ emtpy cell
        print(table_matrix)
        for column in range(0, columns):
            for row in range(0, rows):
                table_matrix[row, column] = row + column
        table_matrix[1, 0] = "example"
        table_matrix[0, 1] = "grid sizes"

        self.put_table_in_tex(table_matrix, table_name, project_nr)

    def get_script_dir(self):
        """returns the directory of this script regardles of from
        ↪ which level the code is executed"""
```

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