Example to plot directly into latex

19-10-2019

1 Introduction

2 Genetic Algorithm Performance

To illustrate how the python code exports the figures directly into the report, this second "hw2" is included. Below are the pictures that are created by the code listed in **??** and **??**.

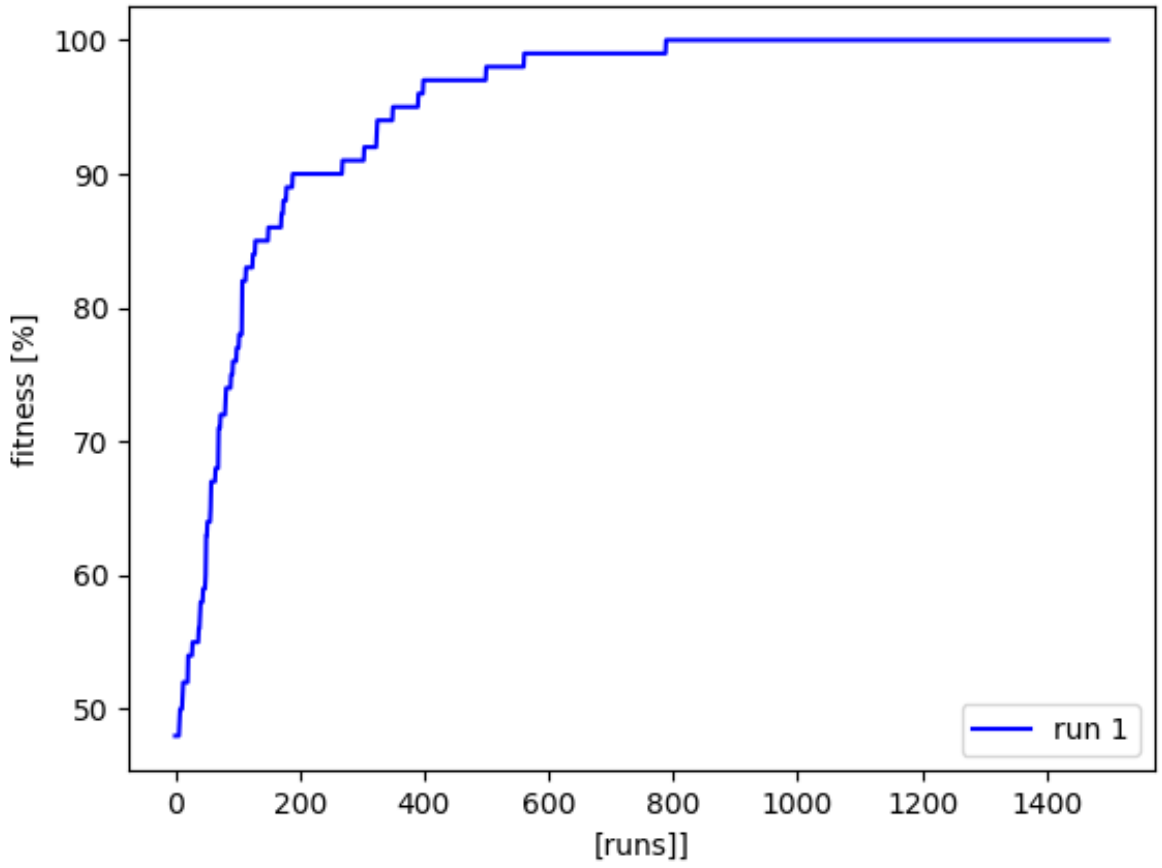


Figure 1: Performance of some genetic algorithm

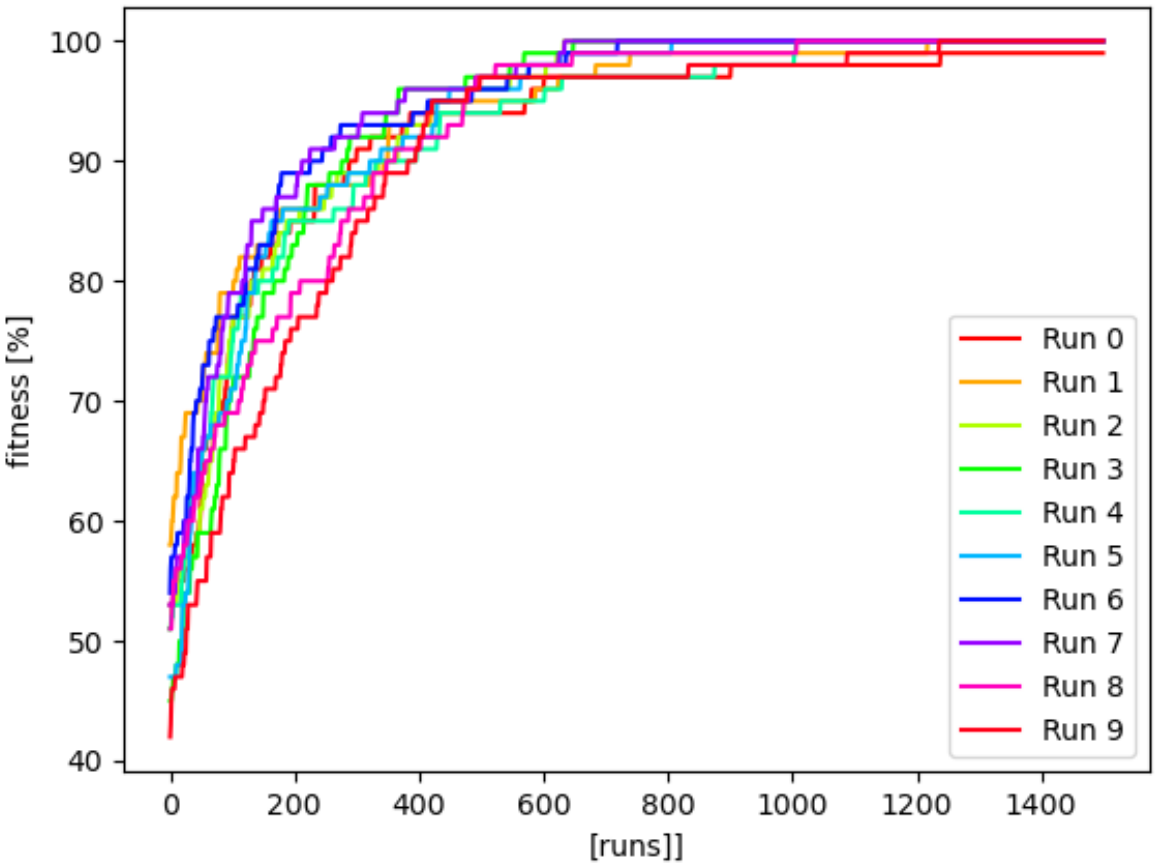


Figure 2: Performance of some genetic algorithm

A Appendix __main__.py

```
'''
Runs the main code.

First it runs the notebooks in this directory
Then it converts those notebooks to pdf
This is followed by compiling the latex  report of this project to pdf.

For illustration purposes, a genetic algorithm is also executed that
plots some images into the latex report. Since the report is compiled
before the genetic algorithm is ran, the new results are only included
after the second of this main

'''
from .Main import Main
import os

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()

notebook_names = ['AE4868_example_notebook_update20201025.ipynb']

# run the jupyter notebooks for assignment 1
main.run_jupyter_notebooks(project_nr,notebook_names)

# convert jupyter notebook for assignment 1 to pdf
main.convert_notebooks_to_pdf(project_nr,notebook_names)

# compile the latex report
main.compile_latex_report(project_nr)


##################################################################
############example code to illustrate python-latex  image sync #########
###############runs arbitrary genetic algorithm, can be deleted ###########
##################################################################
# run a genetic algorithm to create some data for a plot.
print("Running method a of Main.py to execute some genetic algorithm" )
res = main.do_run_a()

# plot some graph with a single line, general form is:
# plt_tex.plotSingleLines(plt_tex,x,y,"x-axis label","y-axis label", lineLabels,"filename",legend_position,project_nr)
# main.plt_tex.plotSingleLine(plt_tex,range(0, len(res)),res,"[runs ]]","fitness [%]","run 1","4a",4,project_nr)

# run a genetic algorithm to create some data for another plot.
print("Running method 4b of Main.py to execute some genetic algorithm ")
main.do4b(project_nr)

# run a genetic algorithm to create some data for another plot.
print("Running method 4c of Main.py to execute some genetic algorithm ")
```

```
51 main.do4c(project_nr)
52
53 print(f'Done with runing code.')
```

B Appendix Main.py

```python
from .Compile_latex import Compile_latex
from .Plot_to_tex import Plot_to_tex as plt_tex
from .Run_jupyter_notebooks import Run_jupyter_notebook

from matplotlib import pyplot as plt
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np
import random

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


class Main:
    """Runs jupiter notebooks, then compiles them to pdf
        Exports those notebook pdfs to the latex of this project
        nr, then compiles the latex report to pdf.

        Als runs a genetic algorithm in conventional .py files
        and exports them to the latex report, to illustrate the
        functionality of the python and latex integration.

        Note that the latex is already compiled before the
        genetic algorith (GA) is ran, so these results of the GA
        are one version behind the latex pdf report.
    """

    def __init__(self):
        self.run_jupyter_notebook = Run_jupyter_notebook()
        pass


    def run_jupyter_notebooks(self,project_nr,notebook_names):
        """calls a method that runs each jupyter notebook in the list
           ↪  of incoming notebook names

        :param project_nr: the numberr identifying which project is
           ↪ being  ran and compiled
        :param notebook_names: list of strings with the names of the
           ↪ notebooks that need to be ran

        """
        notebook_path = f'code/project{project_nr}/src/'

        for notebook_name in notebook_names:
            self.run_jupyter_notebook.run_notebook(f'{notebook_path}{
               ↪ notebook_name}')


    def convert_notebooks_to_pdf(self,project_nr,notebook_names):
        """calls a method that converts each jupyter notebook in the
           ↪ list of incoming notebook names

        :param project_nr: the numberr identifying which project is
           ↪ being  ran and compiled
        :param notebook_names: list of strings with the names of the
           ↪ notebooks that need to be ran
```

```
54
55          """
56          notebook_path = f'code/project{project_nr}/src/'
57
58          for notebook_name in notebook_names:
59              self.run_jupyter_notebook.convert_notebook_to_pdf(f'{
                    ↪ notebook_path}{notebook_name}')
60
61
62      def compile_latex_report(self,project_nr):
63          """compiles latex code to pdf
64
65          :param project_nr: the numberr identifying which project is
                ↪ being  ran and compiled
66
67          """
68          compile_latex =Compile_latex(project_nr ,'main.tex')
69
70
71      ################################################################
72      ############example code to illustrate python-latex  image sync
            ↪ #########
73      ###############runs arbitrary genetic algorithm, can be deleted
            ↪ ############
74      ################################################################
75      def count(self,bits):
76          """counts how many bits there are in a chromosome
77
78          :param bits: representing values of dna in chromosome(s)
79
80          """
81          count = 0
82          for bit in bits:
83              if bit:
84                  count = count + 1
85          return count
86
87
88      def gen_bit_sequence(self):
89          """generates a random bit sequence that represents a
                ↪ chromosome of DNA"""
90          bits = []
91          for _ in range(string_length):
92              bits.append(True if random.randint(0, 1) == 1 else False)
93          return bits
94
95
96      def mutate_bit_sequence(self,sequence):
97          """Randomly changes a bit sequence that changes the
                ↪ chromosome(s) of DNA
98          This is simulating for example radiation effects that
                ↪ generate arbitrary new offspring
99
100         :param sequence: sequence of binary bits that represent a
                ↪ chromosome of DNA
101
102         """
103         retval = []
104         for bit in sequence :
105             do_mutation = random.random() <= mutation_chance
106             if(do_mutation):
107                 retval.append(not bit)
```

```
            else:
                retval.append(bit)
        return retval


    #execute a run a
    def do_run_a(self):
        """Performs a run of the genetic algorithm, like simulating
           ↪ evolution
         and returns the fitness of the population.
        """

        seq = self.gen_bit_sequence()
        fitness = self.count(seq)
        results = [fitness]
        for run in range(max_iterations-1):
            new_seq = self.mutate_bit_sequence(seq)
            new_fitness = self.count(new_seq)
            if new_fitness > fitness:
                seq = new_seq
                fitness = new_fitness
            results.append(max(results[-1],fitness))
        return results


    #execute a run c
    def do_run_c(self):
        """Performs a run of the genetic algorithm, like simulating
           ↪ evolution
         and returns the fitness of the population.
        """
        seq = self.gen_bit_sequence()
        fitness = self.count(seq)
        results = [fitness]
        for run in range(max_iterations):
            new_seq = self.mutate_bit_sequence(seq)
            new_fitness = self.count(new_seq)
            seq = new_seq
            fitness = new_fitness
            results.append(max(results[-1], fitness))
        return results


    def do4b(self,project_nr):
        """Performs a run of the genetic algorithm, like simulating
           ↪ evolution
         and exports the optimum fitness of the population per
            ↪ generation
         as an image to the latex report of the incoming project nr.

        :param project_nr: the numberr identifying which project is
           ↪ being  ran and compiled

        """
        optimum_found = 0

        # generate plot data
        plotResult = np.zeros((10,max_iterations), dtype=int);
        lineLabels = []

        # perform computation
        for run in range(10):
```

```
165             res = self.do_run_a()
166             if res[-1] == string_length:
167                 optimum_found +=1
168 
169             # store computation data for plotting
170             lineLabels.append(f'Run {run}')
171             plotResult[run,:]=res;
172 
173         # plot multiple lines into report (res is an array of
                ↪ dataseries (representing the lines))
174         # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
                ↪ axis label",lineLabels,"filename",legend_position,
                ↪ project_nr)
175         plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
                ↪ plotResult,"[runs]]","fitness [%]",lineLabels,"4b",4,
                ↪ project_nr)
176         print("total optimum found: {} out of {} runs".format(
                ↪ optimum_found,10))
177 
178     def do4c(self,project_nr):
179         """Performs a run of the genetic algorithm, like simulating
                ↪ evolution
180          and exports the optimum fitness of the population per
                ↪ generation
181          as an image to the latex report of the incoming project nr.
182 
183         :param project_nr: the numberr identifying which project is
                ↪ being  ran and compiled
184 
185         """
186         optimum_found = 0
187 
188         # generate plot data
189         plotResult = np.zeros((10,max_iterations+1), dtype=int);
190         lineLabels = []
191 
192         # perform computation
193         for run in range(10):
194             res = self.do_run_c()
195             if res[-1] == string_length:
196                 optimum_found +=1
197 
198             # Store computation results for plot
199             lineLabels.append(f'Run {run}')
200             plotResult[run,:]=res;
201 
202         # plot multiple lines into report (res is an array of
                ↪ dataseries (representing the lines))
203         # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
                ↪ axis label",lineLabels,"filename",legend_position,
                ↪ project_nr)
204         plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
                ↪ plotResult,"[runs]]","fitness [%]",lineLabels,"4c",4,
                ↪ project_nr)
205 
206         print("total optimum found: {} out of {} runs".format(
                ↪ optimum_found, 10))
207 
208 
209     def addTwo(self,x):
210         """adds two to the incoming integer and returns the result of
                ↪  the computation.
```

```
211
212             :param x: incoming integer
213
214             """
215             return x+2
216
217 if __name__ == '__main__':
218     # initialize main class
219     main = Main()
```

Appendix python code that exports figures to latex

```
from matplotlib import lines
import matplotlib.pyplot as plt
import numpy as np
import os
import random


class Plot_to_tex:
    """Plots incoming images and/or tables to a latex report with a certain layout."""
    """
    Example of how to include an exported table into your latex report.

    \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}
    """
    def __init__(self):
        self.script_dir = self.get_script_dir()


    def plotSingleLine(self,x_path,y_series,x_axis_label,y_axis_label,label,filename,legendPosition,project_nr):
        """Outputs a plot with a single line to a latex report

        :param x_path: x coordinates of a line
        :param y_series: y coordinates of a line
        :param x_axis_label: label of x axis
        :param y_axis_label: label of y axis
        :param label: string describing the line (label)
        :param filename: filename of the image that is exported to latex
        :param legendPosition: integer in range 1 to 4 representing the legend position (or string 'best')
        :param project_nr: the number identifying to which latex project this image is exported

        """
        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);
        plt.ylabel(y_axis_label);
        plt.savefig(os.path.dirname(__file__)+'/../../../latex/project'+str(project_nr)+'/Images/'+filename+'.png');
#         plt.show();


    def plotMultipleLines(self,x,y_series,x_label,y_label,label,filename,legendPosition,project_nr):
```

```
50          """Outputs a plot with mulltiple lines to a latex report
51
52          :param x: list of x coordinates of the lines of the plot
53          :param y_series: y coordinates of the lines of the plot
54          :param x_label: label of x axis
55          :param y_label: label of y axis
56          :param label: list of strings describing the lines (labels)
57          :param filename: filename of the image that is exported to
              ↪ latex
58          :param legendPosition: integer in range 1 to 4 representing
              ↪ the legend position (or string 'best')
59          :param project_nr: the number identifying to which latex
              ↪ project this image is exported
60
61          """
62          fig=plt.figure();
63          ax=fig.add_subplot(111);
64
65          # generate colours
66          cmap = self.get_cmap(len(y_series[:,0]))
67
68          # generate line types
69          lineTypes = self.generateLineTypes(y_series)
70
71          for i in range(0,len(y_series)):
72              # overwrite linetypes to single type
73              lineTypes[i] = "-"
74              ax.plot(x,y_series[i,:],ls=lineTypes[i],label=label[i],
                  ↪ fillstyle='none',c=cmap(i)); # color
75
76          # configure plot layout
77          plt.legend(loc=legendPosition);
78          plt.xlabel(x_label);
79          plt.ylabel(y_label);
80          plt.savefig(os.path.dirname(__file__)+'/../../../latex/
              ↪ project'+str(project_nr)+'/Images/'+filename+'.png');
81
82          print(f'plotted lines')
83
84
85      def get_cmap(n, name='hsv'):
86          """Returns a function that maps each index in 0, 1, ..., n-1
              ↪ to a distinct
87          RGB color; the keyword argument name must be a standard mpl
              ↪ colormap name.
88          Source: https://stackoverflow.com/questions/14720331/how-to-
              ↪ generate-random-colors-in-matplotlib
89
90          :param n: number of lines that need a distinct colour
91          :param name:  (Default value = 'hsv') the type of linecolour
              ↪ palet, e.g. rainbow, grayscale etc
92
93          """
94          return plt.cm.get_cmap(name, n)
95
96
97      def generateLineTypes(y_series):
98          """Generates returns a list of a vissible line type for each
              ↪ incoming line/y_series
99
100         :param y_series: list with list of y-coordinates representing
              ↪  the lines
```

```
        """
        # 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


    def put_table_in_tex(self, table_matrix,filename,project_nr):
        """Outputs a table into a latex report

        :param table_matrix: numpy array with the table data
        :param filename: filename of the table that is exported to
           ↪ latex
        :param project_nr: the number identifying to which latex
           ↪ project this table is exported

        """
        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')


    def example_create_a_table(self):
        """Example code that generates the numpy array with
        table data that can be exported to a latex table. Can
        be modified to generate your own latex table"""
        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):
```

```
157         """returns the path of the directory of this script"""
158         return os.path.dirname(__file__)
159 
160 
161 if __name__ == '__main__':
162     main = Plot_to_tex()
163     main.example_create_a_table()
```

Appendix python code that compiles the latex report to pdf

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


class Compile_latex:
    """Runs jupyter notebooks, converts them to pdf,
    exports the notebook pdfs to latex and compiles the
    latex report of the incoming project nr"""


    def __init__(self,project_nr,latex_filename):
        """Constructs attributes used throughout latex compilation

        :param project_nr: the numberr identifying which project is being  ran and compiled
        :param latex_filename: name of the main latex .tex file that manages the latex document
        """

        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)


    def compile_latex(self,relative_dir,latex_filename):
        """Executes a commandline line to compile the latex report

        :param relative_dir: the relative dir towards the latex main .tex file
        :param latex_filename: name of the main latex .tex file that manages the latex document

        """
        os.system(f'pdflatex {relative_dir}{latex_filename}')


    def clean_up_after_compilation(self,latex_filename):
        """Removes the unneeded files that were generated during latex to pdf compilation.

        :param latex_filename: name of the main latex .tex file that manages the latex document

        """
        latex_filename_without_extention = latex_filename[:-4]
        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):
        """Moves the compiled/generated pdf file from the root of this repository to the
        relative latex directory of this project.
```

```
52
53          :param relative_dir: param latex_filename:
54          :param latex_filename: name of the main latex .tex file that
                ↪ manages the latex document
55
56          """
57          pdf_filename = f'{latex_filename[:-4]}.pdf'
58          destination= f'{self.get_script_dir()}/../../../{relative_dir
                ↪ }{pdf_filename}'
59
60          try:
61              shutil.move(pdf_filename, destination)
62          except:
63              print("Error while moving file ", pdf_filename)
64
65
66      def delete_file_if_exists(self,filename):
67          """Deletes files if they exist
68
69          :param filename: name of file that will be deleted if it
                ↪ exists in the root of this repository
70
71          """
72          try:
73              os.remove(filename)
74          except:
75              print(f'Error while deleting file: {filename} but that is
                    ↪  not too bad because the intention is for it to not
                    ↪  be there.')
76
77
78      def get_script_dir(self):
79          """returns the directory of this script regardles of from
                ↪ which level the code is executed"""
80          return os.path.dirname(__file__)
81
82
83  if __name__ == '__main__':
84      main = Compile_latex()
```

Appendix python code that runs the jupyter notebook(s)

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

class Run_jupyter_notebook:
    """runs a list of  jupyter notebooks and converts it to pdf"""


    def __init__(self):
        self.script_dir = self.get_script_dir()


    def run_jupyter_notebooks(self,project_nr,notebook_names):
        """runs a jupyter notebook in this directory

        :param project_nr: the numberr identifying which project is being  ran and compiled
        :param notebook_names: list of strings with the names of the notebooks that need to be ran

        """
        notebook_path = f'code/project{project_nr}/src/'

        for notebook_name in notebook_names:
            self.run_jupyter_notebook.run_notebook(f'{notebook_path}{notebook_name}')


    def convert_notebooks_to_pdf(self,project_nr,notebook_names):
        """converts a jupyter notebook to pdf

        :param project_nr: the numberr identifying which project is being  ran and compiled
        :param notebook_names: list of strings with the names of the notebooks that need to be ran

        """
        notebook_path = f'code/project{project_nr}/src/'

        for notebook_name in notebook_names:
            self.run_jupyter_notebook.convert_notebook_to_pdf(f'{notebook_path}{notebook_name}')


    def compile_latex_report(self,project_nr):
        """compiles latex code to pdf

        :param project_nr: the numberr identifying which project is being  ran and compiled

        """
        compile_latex =Compile_latex(project_nr ,'main.tex')


    def run_notebook(self,notebook_filename):
        """runs a  jupyter notebook that is located in this folder

        :param notebook_filename: the name of the notebook that needs to be ran

```

```
53          """
54          # Load your notebook
55          with open(notebook_filename) as f:
56              nb = nbformat.read(f, as_version=4)
57
58          # Configure
59          ep = ExecutePreprocessor(timeout=600, kernel_name='python3')
60
61          # Execute
62          #ep.preprocess(nb, {'metadata': {'path': 'notebooks/'}})
63          ep.preprocess(nb, {'metadata': {'path': f'{self.
              ↪ get_script_dir()}'}})
64
65          # Save output notebook
66          with open(notebook_filename, 'w', encoding='utf-8') as f:
67              nbformat.write(nb, f)
68
69
70      def convert_notebook_to_pdf(self,notebook_filename):
71          """Compiles a jupyter notebook that is located in this folder
              ↪  to pdf
72
73          :param notebook_filename: the name of the notebook that needs
              ↪  to be compiled to pdf
74
75          """
76          os.system(f'jupyter nbconvert --to pdf {notebook_filename}')
77
78
79      def get_script_dir(self):
80          """returns the directory of this script regardles of from
              ↪ which level the code is executed"""
81          return os.path.dirname(__file__)
82
83
84  if __name__ == '__main__':
85      main = Run_jupyter_notebook()
```

Appendix Example Jupyter Notebook

AE4868_example_notebook_update20201025

November 18, 2020

[1]:
```
def addThree(input_nr):
    '''returns the input integer plus 3, used to verify unit test'''
    return input_nr + 3
```

[2]:
```
##############################################################################
# IMPORT STATEMENTS ##########################################################
##############################################################################
import os
import numpy as np
from tudatpy.kernel import constants
from tudatpy.kernel.interface import spice_interface
from tudatpy.kernel.simulation import environment_setup
from tudatpy.kernel.simulation import propagation_setup
from tudatpy.kernel.astro import conversion

# Set path to latex image folders for project 1

if (os.path.abspath('')[-12:]=="project1/src"):
    latex_image_path = '../../../latex/project1/Images/'
else:
    latex_image_path = 'latex/project1/Images/' # when ran as test


# Load spice kernels.
spice_interface.load_standard_kernels()

# Set simulation start and end epochs.
simulation_start_epoch = 0.0
simulation_end_epoch = constants.JULIAN_DAY

##############################################################################
# CREATE ENVIRONMENT #########################################################
##############################################################################

# Create default body settings for selected celestial bodies
bodies_to_create = ["Sun", "Earth", "Moon", "Mars", "Venus"]
```

```
# Create default body settings for bodies_to_create, with "Earth"/"J2000" as
# global frame origin and orientation. This environment will only be valid
# in the indicated time range
# [simulation_start_epoch --- simulation_end_epoch]
body_settings = environment_setup.get_default_body_settings(
    bodies_to_create,
    simulation_start_epoch,
    simulation_end_epoch,
    "Earth","J2000")

# Create system of selected celestial bodies
bodies = environment_setup.create_system_of_bodies(body_settings)

################################################################################
# CREATE VEHICLE ###############################################################
################################################################################

# Create vehicle objects.
bodies.create_empty_body( "Delfi-C3" )
bodies.get_body( "Delfi-C3").set_constant_mass(400.0)

# Create aerodynamic coefficient interface settings, and add to vehicle
reference_area = 4.0
drag_coefficient = 1.2
aero_coefficient_settings = environment_setup.aerodynamic_coefficients.constant(
    reference_area,[drag_coefficient,0,0]
)
environment_setup.add_aerodynamic_coefficient_interface(
            bodies, "Delfi-C3", aero_coefficient_settings )

# Create radiation pressure settings, and add to vehicle
reference_area_radiation = 4.0
radiation_pressure_coefficient = 1.2
occulting_bodies = ["Earth"]
radiation_pressure_settings = environment_setup.radiation_pressure.cannonball(
    "Sun", reference_area_radiation, radiation_pressure_coefficient,␣
 ↪occulting_bodies
)
environment_setup.add_radiation_pressure_interface(
            bodies, "Delfi-C3", radiation_pressure_settings )

################################################################################
# CREATE ACCELERATIONS #########################################################
################################################################################

# Define bodies that are propagated.
bodies_to_propagate = ["Delfi-C3"]
```

```
# Define central bodies.
central_bodies = ["Earth"]

# Define accelerations acting on Delfi-C3 by Sun and Earth.
accelerations_settings_delfi_c3 = dict(
    Sun=
    [
        propagation_setup.acceleration.cannonball_radiation_pressure(),
        propagation_setup.acceleration.point_mass_gravity()
    ],
    Earth=
    [
        propagation_setup.acceleration.spherical_harmonic_gravity(5, 5),
        propagation_setup.acceleration.aerodynamic()
    ])

# Define point mass accelerations acting on Delfi-C3 by all other bodies.
for other in set(bodies_to_create).difference({"Sun", "Earth"}):
    accelerations_settings_delfi_c3[other] = [
        propagation_setup.acceleration.point_mass_gravity()]

# Create global accelerations settings dictionary.
acceleration_settings = {"Delfi-C3": accelerations_settings_delfi_c3}

# Create acceleration models.
acceleration_models = propagation_setup.create_acceleration_models(
    bodies,
    acceleration_settings,
    bodies_to_propagate,
    central_bodies)

#########################################################################
# CREATE PROPAGATION SETTINGS ###########################################
#########################################################################

# Set initial conditions for the Asterix satellite that will be
# propagated in this simulation. The initial conditions are given in
# Keplerian elements and later on converted to Cartesian elements.
earth_gravitational_parameter = bodies.get_body( "Earth" ).
 ↪gravitational_parameter
initial_state = conversion.keplerian_to_cartesian(
    gravitational_parameter=earth_gravitational_parameter,
    semi_major_axis=7500.0E3,
    eccentricity=0.1,
    inclination=np.deg2rad(85.3),
    argument_of_periapsis=np.deg2rad(235.7),
```

```
    longitude_of_ascending_node=np.deg2rad(23.4),
    true_anomaly=np.deg2rad(139.87)
)

# Define list of dependent variables to save.
dependent_variables_to_save = [
    propagation_setup.dependent_variable.total_acceleration( "Delfi-C3" ),
    propagation_setup.dependent_variable.keplerian_state( "Delfi-C3", "Earth" ),
    propagation_setup.dependent_variable.latitude( "Delfi-C3", "Earth" ),
    propagation_setup.dependent_variable.longitude( "Delfi-C3", "Earth"),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",␣
 ↪"Sun"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",␣
 ↪"Moon"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",␣
 ↪"Mars"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",␣
 ↪"Venus"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.spherical_harmonic_gravity_type,␣
 ↪"Delfi-C3", "Earth"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.aerodynamic_type, "Delfi-C3", "Earth"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.cannonball_radiation_pressure_type,␣
 ↪"Delfi-C3", "Sun"
    )
    ]


# Create propagation settings.
propagator_settings = propagation_setup.propagator.translational(
    central_bodies,
    acceleration_models,
    bodies_to_propagate,
    initial_state,
```

```
    simulation_end_epoch,
    output_variables = dependent_variables_to_save
)
# Create numerical integrator settings.
fixed_step_size = 10.0
integrator_settings = propagation_setup.integrator.runge_kutta_4(
    simulation_start_epoch,
    fixed_step_size
)

###########################################################################
# PROPAGATE ORBIT #########################################################
###########################################################################

# Create simulation object and propagate dynamics.
dynamics_simulator = propagation_setup.SingleArcDynamicsSimulator(
    bodies, integrator_settings, propagator_settings)
states = dynamics_simulator.state_history
dependent_variables = dynamics_simulator.dependent_variable_history

###########################################################################
# PRINT INITIAL AND FINAL STATES ##########################################
###########################################################################

print(
    f"""
Single Earth-Orbiting Satellite Example.
The initial position vector of Delfi-C3 is [km]: \n{
    states[simulation_start_epoch][:3] / 1E3}
The initial velocity vector of Delfi-C3 is [km/s]: \n{
    states[simulation_start_epoch][3:] / 1E3}
After {simulation_end_epoch} seconds the position vector of Delfi-C3 is [km]:␣
 ↪\n{
    states[simulation_end_epoch][:3] / 1E3}
And the velocity vector of Delfi-C3 is [km/s]: \n{
    states[simulation_end_epoch][3:] / 1E3}
    """
)
```

```
Single Earth-Orbiting Satellite Example.
The initial position vector of Delfi-C3 is [km]:
[7037.48400133 3238.05901792 2150.7241875 ]
The initial velocity vector of Delfi-C3 is [km/s]:
[-1.46565763 -0.04095839  6.62279761]
After 86400.0 seconds the position vector of Delfi-C3 is [km]:
[-4602.79426676 -1421.16740978  5883.69740624]
```

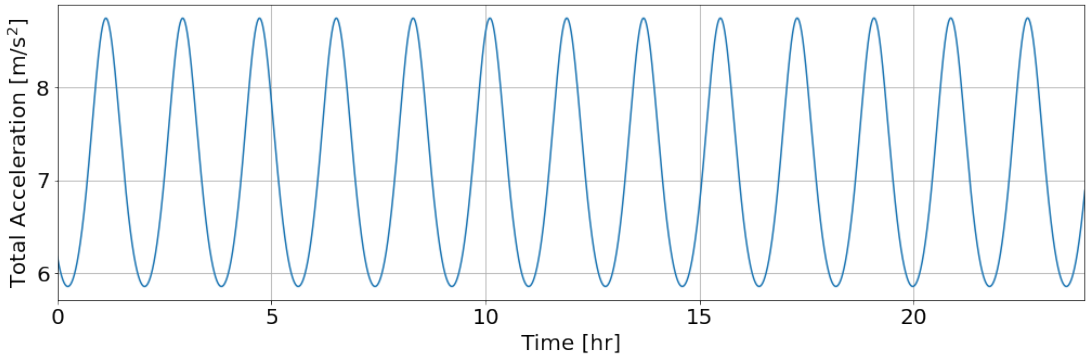
```
And the velocity vector of Delfi-C3 is [km/s]:
[-4.53846052 -2.36988263 -5.04163195]
```

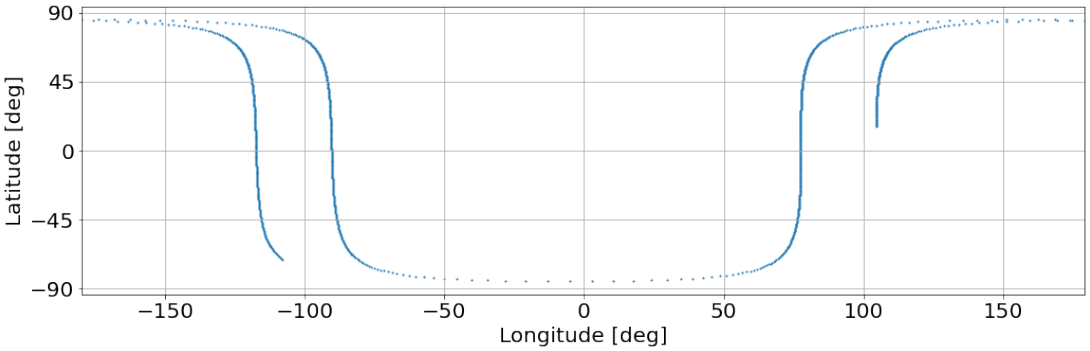
[3]:
```
import os
from matplotlib import pyplot as plt

time = dependent_variables.keys()
dependent_variable_list = np.vstack(list(dependent_variables.values()))
font_size = 20

plt.rcParams.update({'font.size': font_size})

# dependent variables
# 0-2: total acceleration
# 3-8: Keplerian state
# 9: latitude
# 10: longitude
# 11: Acceleration Norm PM Sun
# 12: Acceleration Norm PM Moon
# 13: Acceleration Norm PM Mars
# 14: Acceleration Norm PM Venus
# 15: Acceleration Norm SH Earth

total_acceleration = np.sqrt( dependent_variable_list[:,0] ** 2 +␣
 ↪dependent_variable_list[:,1] ** 2 + dependent_variable_list[:,2] ** 2 )

time_hours = [ t / 3600 for t in time]
# Total Acceleration
plt.figure( figsize=(17,5))
plt.grid()
plt.plot( time_hours , total_acceleration )
plt.xlabel('Time [hr]')
plt.ylabel( 'Total Acceleration [m/s$^2$]')
plt.xlim( [min(time_hours), max(time_hours)] )
plt.savefig( fname = f'{latex_image_path}total_acceleration.png',␣
 ↪bbox_inches='tight')



# Ground Track
latitude = dependent_variable_list[:,9]
longitude = dependent_variable_list[:,10]

part = int(len(time)/24*3)
latitude = np.rad2deg( latitude[0:part] )
longitude = np.rad2deg( longitude[0:part] )
```

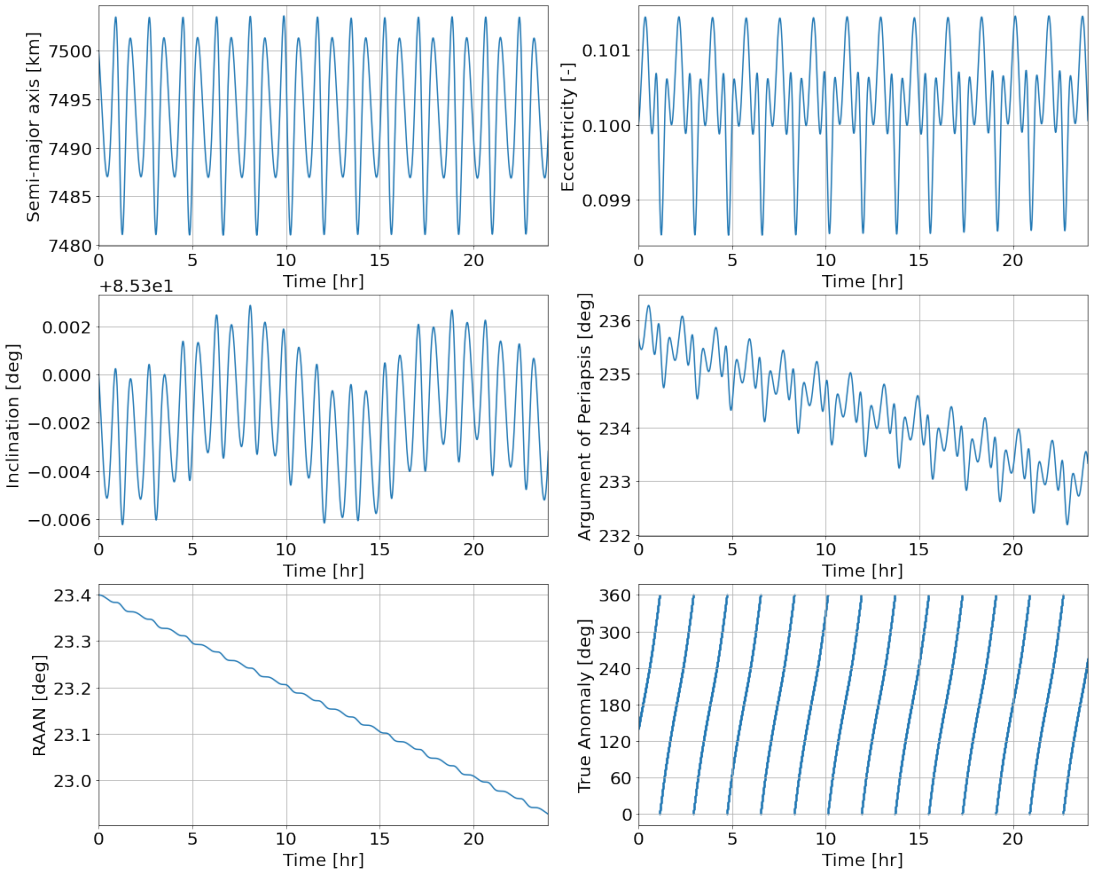
```
plt.figure( figsize=(17,5))
plt.grid()
plt.yticks(np.arange(-90, 91, step=45))
plt.scatter( longitude, latitude, s=1 )
plt.xlabel('Longitude [deg]')
plt.ylabel( 'Latitude [deg]')
plt.xlim( [min(longitude), max(longitude)] )
plt.savefig( fname = f'{latex_image_path}ground_track.png', bbox_inches='tight')

# Kepler Elements
kepler_elements = dependent_variable_list[:,3:9]

fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots( 3, 2, figsize =␣
↪(20,17) )

# Semi-major Axis
semi_major_axis = [ element/1000 for element in kepler_elements[:,0] ]
ax1.plot( time_hours, semi_major_axis )
ax1.set_ylabel( 'Semi-major axis [km]' )

# Eccentricity
eccentricity = kepler_elements[:,1]
ax2.plot( time_hours, eccentricity )
ax2.set_ylabel( 'Eccentricity [-]' )

# Inclination
inclination = [ np.rad2deg( element ) for element in kepler_elements[:,2] ]
ax3.plot( time_hours, inclination )
ax3.set_ylabel( 'Inclination [deg]')

# Argument of Periapsis
argument_of_periapsis = [ np.rad2deg( element ) for element in kepler_elements[:
↪,3] ]
ax4.plot( time_hours, argument_of_periapsis )
ax4.set_ylabel( 'Argument of Periapsis [deg]' )

# Right Ascension of the Ascending Node
raan = [ np.rad2deg( element ) for element in kepler_elements[:,4] ]
ax5.plot( time_hours, raan )
ax5.set_ylabel( 'RAAN [deg]' )

# True Anomaly
true_anomaly = [ np.rad2deg( element ) for element in kepler_elements[:,5] ]
ax6.scatter( time_hours, true_anomaly, s=1 )
ax6.set_ylabel( 'True Anomaly [deg]' )
ax6.set_yticks(np.arange(0, 361, step=60))
```

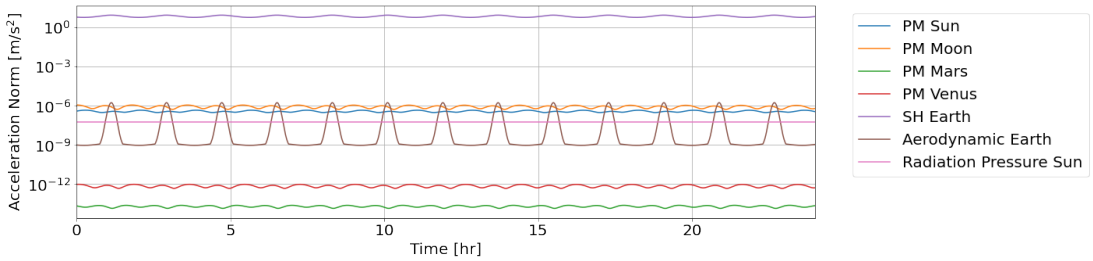
```
for ax in fig.get_axes():
    ax.set_xlabel('Time [hr]')
    ax.set_xlim( [min(time_hours), max(time_hours)] )
    ax.grid()

plt.savefig( fname = f'{latex_image_path}kepler_elements.png',␣
 ↪bbox_inches='tight')

plt.figure( figsize=(17,5))

# Point Mass Gravity Acceleration Sun
acceleration_norm_pm_sun = dependent_variable_list[:, 11]
plt.plot( time_hours, acceleration_norm_pm_sun, label='PM Sun')

# Point Mass Gravity Acceleration Moon
acceleration_norm_pm_moon = dependent_variable_list[:, 12]
plt.plot( time_hours, acceleration_norm_pm_moon, label='PM Moon')

# Point Mass Gravity Acceleration Mars
acceleration_norm_pm_mars = dependent_variable_list[:, 13]
plt.plot( time_hours, acceleration_norm_pm_mars, label='PM Mars')

# Point Mass Gravity Acceleration Venus
acceleration_norm_pm_venus = dependent_variable_list[:, 14]
plt.plot( time_hours, acceleration_norm_pm_venus, label='PM Venus')

# Spherical Harmonic Gravity Acceleration Earth
acceleration_norm_sh_earth = dependent_variable_list[:, 15]
plt.plot( time_hours, acceleration_norm_sh_earth, label='SH Earth')

# Aerodynamic Acceleration Earth
acceleration_norm_aero_earth = dependent_variable_list[:, 16]
plt.plot( time_hours, acceleration_norm_aero_earth, label='Aerodynamic Earth')

# Cannonball Radiation Pressure Acceleration Sun
acceleration_norm_rp_sun = dependent_variable_list[:, 17]
plt.plot( time_hours, acceleration_norm_rp_sun, label='Radiation Pressure Sun')

plt.grid()
plt.legend( bbox_to_anchor=(1.04,1) )
plt.xlim( [min(time_hours), max(time_hours)])
plt.yscale('log')
plt.xlabel( 'Time [hr]' )
plt.ylabel( 'Acceleration Norm [m/s$^2$]' )

plt.savefig( fname = f'{latex_image_path}acceleration_norms.png',␣
 ↪bbox_inches='tight')
```

```
#plt.savefig('acceleration_norms.png', bbox_inches='tight')
```






Semi-major axis [km]
7500
7495
7490
7485
7480
0
5
10
15
20
Time [hr]
Eccentricity [-]
0.101
0.100
0.099
Time [hr]
+8.53e1
Inclination [deg]
0.002
0.000
−0.002
−0.004
−0.006
Time [hr]
Argument of Periapsis [deg]
236
235
234
233
232
Time [hr]
RAAN [deg]
23.4
23.3
23.2
23.1
23.0
Time [hr]
True Anomaly [deg]
360
300
240
180
120
60
0
Time [hr]


Acceleration Norm [m/s²]
10⁰
10⁻³
10⁻⁶
10⁻⁹
10⁻¹²
0
5
10
15
20
Time [hr]
PM Sun
PM Moon
PM Mars
PM Venus
SH Earth
Aerodynamic Earth
Radiation Pressure Sun

[ ]: