

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

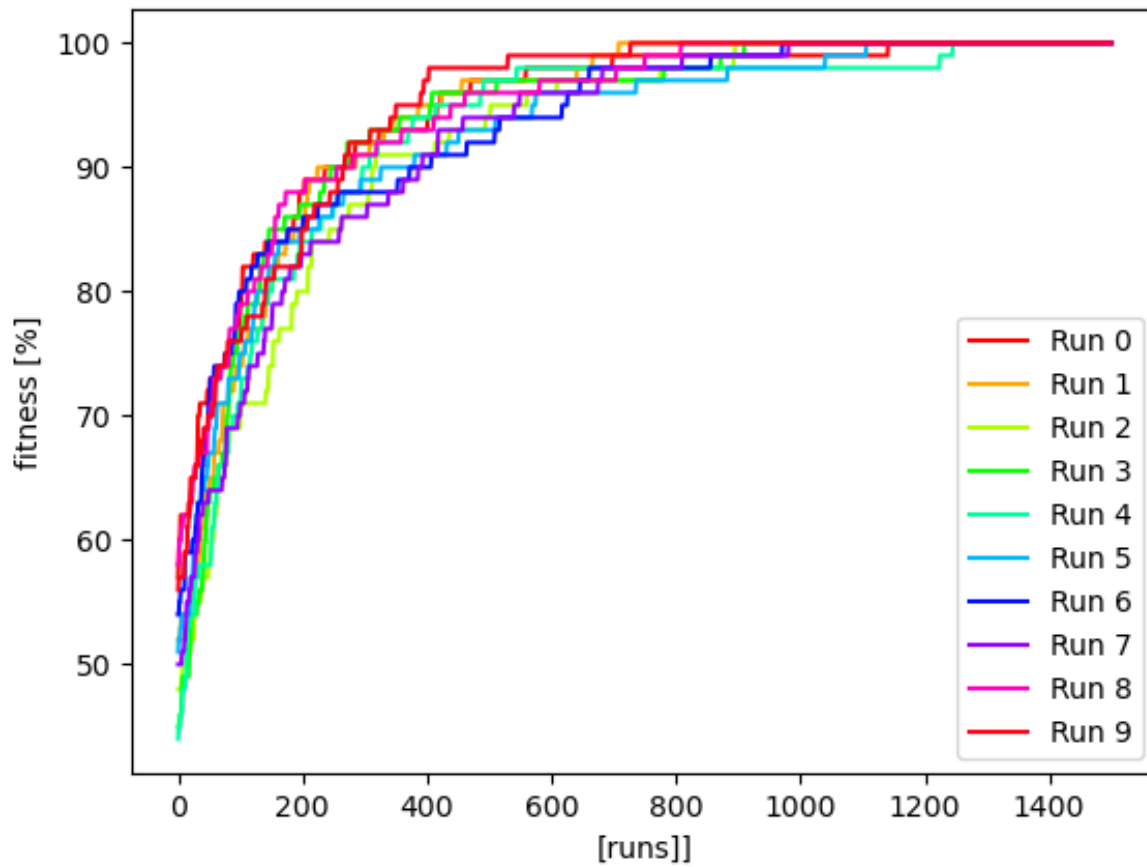


Figure 2: Performance of some genetic algorithm

A Appendix __main__.py

```
1  '''
2  Runs the main code.
3
4  First it runs the notebooks in this directory
5  Then it converts those notebooks to pdf
6  This is followed by compiling the latex report of this project to
   ↪ pdf.
7
8  For illustration purposes, a genetic algorithm is also executed that
9  plots some images into the latex report. Since the report is compiled
10 before the genetic algorithm is ran, the new results are only
   ↪ included
11 after the second of this main
12
13 '''
14 from .Main import Main
15 import os
16
17 print(f'Hi, I\'ll be running the main code, and I\'ll let you know
   ↪ when I\'m done.')
18 project_nr = 1
19 main = Main()
20
21 notebook_names = ['AE4868_example_notebook_update20201025.ipynb']
22
23 # run the jupyter notebooks for assignment 1
24 main.run_jupyter_notebooks(project_nr,notebook_names)
25
26 # convert jupyter notebook for assignment 1 to pdf
27 main.convert_notebooks_to_pdf(project_nr,notebook_names)
28
29 # compile the latex report
30 main.compile_latex_report(project_nr)
31
32
33 #####
34 #####example code to illustrate python-latex image sync
   ↪ #####
35 #####runs arbitrary genetic algorithm, can be deleted
   ↪ #####
36 #####
37 # run a genetic algorithm to create some data for a plot.
38 print("Running method a of Main.py to execute some genetic algorithm"
   ↪ )
39 res = main.do_run_a()
40
41 # plot some graph with a single line, general form is:
42 # plt_tex.plotSingleLines(plt_tex,x,y,"x-axis label","y-axis label",
   ↪ lineLabels,"filename",legend_position,project_nr)
43 # main.plt_tex.plotSingleLine(plt_tex,range(0, len(res)),res,"[runs
   ↪ ]]", "fitness [%]", "run 1", "4a", 4, project_nr)
44
45 # run a genetic algorithm to create some data for another plot.
46 print("Running method 4b of Main.py to execute some genetic algorithm
   ↪ ")
47 main.do4b(project_nr)
48
49 # run a genetic algorithm to create some data for another plot.
50 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

```
1 from .Compile_latex import Compile_latex
2 from .Plot_to_tex import Plot_to_tex as plt_tex
3 from .Run_jupyter_notebooks import Run_jupyter_notebook
4
5 from matplotlib import pyplot as plt
6 from matplotlib import lines
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import random
10
11 # define global variables for genetic algorithm example
12 string_length = 100
13 mutation_chance= 1.0/string_length
14 max_iterations = 1500
15
16
17 class Main:
18     """Runs jupyter notebooks, then compiles them to pdf
19     Exports those notebook pdfs to the latex of this project
20     nr, then compiles the latex report to pdf.
21
22     Als runs a genetic algorithm in conventional .py files
23     and exports them to the latex report, to illustrate the
24     functionality of the python and latex integration.
25
26     Note that the latex is already compiled before the
27     genetic algorith (GA) is ran, so these results of the GA
28     are one version behind the latex pdf report.
29     """
30
31     def __init__(self):
32         self.run_jupyter_notebook = Run_jupyter_notebook()
33         pass
34
35
36     def run_jupyter_notebooks(self, project_nr, notebook_names):
37         """calls a method that runs each jupyter notebook in the list
38         ↪ of incoming notebook names
39
40         :param project_nr: the numberr identifying which project is
41         ↪ being ran and compiled
42         :param notebook_names: list of strings with the names of the
43         ↪ notebooks that need to be ran
44
45         """
46         notebook_path = f'code/project{project_nr}/src/'
47
48         for notebook_name in notebook_names:
49             self.run_jupyter_notebook.run_notebook(f'{notebook_path}{
50             ↪ notebook_name}')
51
52
53     def convert_notebooks_to_pdf(self, project_nr, notebook_names):
54         """calls a method that converts each jupyter notebook in the
55         ↪ list of incoming notebook names
56
57         :param project_nr: the numberr identifying which project is
58         ↪ being ran and compiled
59         :param notebook_names: list of strings with the names of the
60         ↪ notebooks that need to be ran
```

```

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

```

```

108         else:
109             retval.append(bit)
110     return retval
111
112
113 #execute a run a
114 def do_run_a(self):
115     """Performs a run of the genetic algorithm, like simulating
116         ↪ evolution
117         and returns the fitness of the population.
118     """
119
120     seq = self.gen_bit_sequence()
121     fitness = self.count(seq)
122     results = [fitness]
123     for run in range(max_iterations-1):
124         new_seq = self.mutate_bit_sequence(seq)
125         new_fitness = self.count(new_seq)
126         if new_fitness > fitness:
127             seq = new_seq
128             fitness = new_fitness
129         results.append(max(results[-1], fitness))
130     return results
131
132 #execute a run c
133 def do_run_c(self):
134     """Performs a run of the genetic algorithm, like simulating
135         ↪ evolution
136         and returns the fitness of the population.
137     """
138
139     seq = self.gen_bit_sequence()
140     fitness = self.count(seq)
141     results = [fitness]
142     for run in range(max_iterations):
143         new_seq = self.mutate_bit_sequence(seq)
144         new_fitness = self.count(new_seq)
145         seq = new_seq
146         fitness = new_fitness
147         results.append(max(results[-1], fitness))
148     return results
149
150 def do4b(self, project_nr):
151     """Performs a run of the genetic algorithm, like simulating
152         ↪ evolution
153         and exports the optimum fitness of the population per
154         ↪ generation
155         as an image to the latex report of the incoming project nr.
156
157     :param project_nr: the numberr identifying which project is
158         ↪ being ran and compiled
159
160     """
161     optimum_found = 0
162
163     # generate plot data
164     plotResult = np.zeros((10, max_iterations), dtype=int);
165     lineLabels = []
166
167     # perform computation
168     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
174         ↪ dataseries (representing the lines))
175         # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
176         ↪ axis label",lineLabels,"filename",legend_position,
177         ↪ project_nr)
178         plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
179         ↪ plotResult,"[runs]", "fitness [%]",lineLabels,"4b",4,
180         ↪ project_nr)
181         print("total optimum found: {} out of {} runs".format(
182         ↪ optimum_found,10))
183
184     def do4c(self,project_nr):
185         """Performs a run of the genetic algorithm, like simulating
186         ↪ evolution
187         and exports the optimum fitness of the population per
188         ↪ generation
189         as an image to the latex report of the incoming project nr.
190
191         :param project_nr: the numberr identifying which project is
192         ↪ being ran and compiled
193
194         """
195         optimum_found = 0
196
197         # generate plot data
198         plotResult = np.zeros((10,max_iterations+1), dtype=int);
199         lineLabels = []
200
201         # perform computation
202         for run in range(10):
203             res = self.do_run_c()
204             if res[-1] == string_length:
205                 optimum_found +=1
206
207             # Store computation results for plot
208             lineLabels.append(f'Run {run}')
209             plotResult[run,:]=res;
210
211         # plot multiple lines into report (res is an array of
212         ↪ dataseries (representing the lines))
213         # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
214         ↪ axis label",lineLabels,"filename",legend_position,
215         ↪ project_nr)
216         plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
217         ↪ plotResult,"[runs]", "fitness [%]",lineLabels,"4c",4,
218         ↪ project_nr)
219
220         print("total optimum found: {} out of {} runs".format(
221         ↪ optimum_found, 10))
222
223     def addTwo(self,x):
224         """adds two to the incoming integer and returns the result of
225         ↪ 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

```
1 from matplotlib import lines
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import os
5 import random
6
7
8 class Plot_to_tex:
9     """Plots incoming images and/or tables to a latex report with a
10         ↪ certain layout."""
11     """
12     Example of how to include an exported table into your latex
13         ↪ report.
14
15     \begin{table}[H]
16         \centering
17         \caption{Results some computation.}\label{tab:
18             ↪ some_computation}
19         \begin{tabular}{|c|c|} % remember to update this to show all
20             ↪ columns of table
21             \hline
22             \input{latex/project3/tables/q2.txt}
23         \end{tabular}
24     \end{table}
25     """
26     def __init__(self):
27         self.script_dir = self.get_script_dir()
28
29     def plotSingleLine(self,x_path,y_series,x_axis_label,y_axis_label
30         ↪ ,label,filename,legendPosition,project_nr):
31         """Outputs a plot with a single line to a latex report
32
33         :param x_path: x coordinates of a line
34         :param y_series: y coordinates of a line
35         :param x_axis_label: label of x axis
36         :param y_axis_label: label of y axis
37         :param label: string describing the line (label)
38         :param filename: filename of the image that is exported to
39             ↪ latex
40         :param legendPosition: integer in range 1 to 4 representing
41             ↪ the legend position (or string 'best')
42         :param project_nr: the number identifying to which latex
43             ↪ project this image is exported
44
45         """
46         fig=plt.figure();
47         ax=fig.add_subplot(111);
48         ax.plot(x_path,y_series,c='b',ls='-',label=label,fillstyle='
49             ↪ none');
50         plt.legend(loc=legendPosition);
51         plt.xlabel(x_axis_label);
52         plt.ylabel(y_axis_label);
53         plt.savefig(os.path.dirname(__file__)+ '/../../../latex/
54             ↪ project'+str(project_nr)+'/Images/'+filename+'.png');
55         # plt.show();
56
57     def plotMultipleLines(self,x,y_series,x_label,y_label,label,
58         ↪ 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
58         ↳ latex
59     :param legendPosition: integer in range 1 to 4 representing
60         ↳ the legend position (or string 'best')
61     :param project_nr: the number identifying to which latex
62         ↳ project this image is exported
63
64     """
65     fig=plt.figure();
66     ax=fig.add_subplot(111);
67
68     # generate colours
69     cmap = self.get_cmap(len(y_series[:,0]))
70
71     # generate line types
72     lineTypes = self.generateLineTypes(y_series)
73
74     for i in range(0,len(y_series)):
75         # overwrite linetypes to single type
76         lineTypes[i] = "-"
77         ax.plot(x,y_series[i,:],ls=lineTypes[i],label=label[i],
78             ↳ fillstyle='none',c=cmap(i)); # color
79
80     # configure plot layout
81     plt.legend(loc=legendPosition);
82     plt.xlabel(x_label);
83     plt.ylabel(y_label);
84     plt.savefig(os.path.dirname(__file__)+ '/../../../latex/
85         ↳ project'+str(project_nr)+'/Images/'+filename+'.png');
86
87     print(f'plotted lines')
88
89 def get_cmap(n, name='hsv'):
90     """Returns a function that maps each index in 0, 1, ..., n-1
91         ↳ to a distinct
92         ↳ RGB color; the keyword argument name must be a standard mpl
93         ↳ colormap name.
94         ↳ Source: https://stackoverflow.com/questions/14720331/how-to-
95         ↳ generate-random-colors-in-matplotlib
96
97     :param n: number of lines that need a distinct colour
98     :param name: (Default value = 'hsv') the type of linecolour
99         ↳ palet, e.g. rainbow, grayscale etc
100
101     """
102     return plt.cm.get_cmap(name, n)
103
104 def generateLineTypes(y_series):
105     """Generates returns a list of a vissible line type for each
106         ↳ incoming line/y_series
107
108     :param y_series: list with list of y-coordinates representing
109         ↳ the lines

```

```

101
102
103 # generate varying linetypes
104 typeOfLines = list(lines.lineStyles.keys())
105
106 while(len(y_series)>len(typeOfLines)):
107     typeOfLines.append("-.");
108
109 # remove void lines
110 for i in range(0, len(y_series)):
111     if (typeOfLines[i]== 'None'):
112         typeOfLines[i]='-'
113     if (typeOfLines[i]== ''):
114         typeOfLines[i]=':'
115     if (typeOfLines[i]== ' '):
116         typeOfLines[i]='--'
117 return typeOfLines
118
119
120 def put_table_in_tex(self, table_matrix, filename, project_nr):
121     """Outputs a table into a latex report
122
123     :param table_matrix: numpy array with the table data
124     :param filename: filename of the table that is exported to
125         ↪ latex
126     :param project_nr: the number identifying to which latex
127         ↪ project this table is exported
128
129     """
130     cols = np.shape(table_matrix)[1]
131     format = "%s"
132     for col in range(1,cols):
133         format = format+" & %s"
134     format = format+"
135     plt.savetxt(os.path.dirname(__file__)+"/../../latex/
136         ↪ project"+str(project_nr)+"/tables/"+filename+".txt",
137         ↪ table_matrix, delimiter=' & ', fmt=format, newline='
138         ↪ \\\n \hline \n')
139
140
141 def example_create_a_table(self):
142     """Example code that generates the numpy array with
143     table data that can be exported to a latex table. Can
144     be modified to generate your own latex table"""
145     project_nr = "1"
146     table_name = "example_table_name"
147     rows = 2;
148     columns = 4;
149     table_matrix = np.zeros((rows,columns),dtype=object)
150     table_matrix[:,:]="" # replace the standard zeros with empty
151         ↪ cell
152     print(table_matrix)
153     for column in range(0,columns):
154         for row in range(0,rows):
155             table_matrix[row,column]=row+column
156     table_matrix[1,0]="example"
157     table_matrix[0,1]="grid sizes"
158
159     self.put_table_in_tex(table_matrix, table_name, project_nr)
160
161
162 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

```
1 from nbconvert.preprocessors import ExecutePreprocessor
2 import os
3 import shutil
4 import nbformat
5
6
7 class Compile_latex:
8     """Runs jupyter notebooks, converts them to pdf,
9     exports the notebook pdfs to latex and compiles the
10     latex report of the incoming project nr"""
11
12
13     def __init__(self, project_nr, latex_filename):
14         """Constructs attributes used throughout latex compilation
15
16         :param project_nr: the numberr identifying which project is
17             ↳ being ran and compiled
18         :param latex_filename: name of the main latex .tex file that
19             ↳ manages the latex document
20         """
21
22         self.script_dir = self.get_script_dir()
23         relative_dir = f'latex/project{project_nr}/'
24         self.compile_latex(relative_dir, latex_filename)
25         self.clean_up_after_compilation(latex_filename)
26         self.move_pdf_into_latex_dir(relative_dir, latex_filename)
27
28     def compile_latex(self, relative_dir, latex_filename):
29         """Executes a commandline line to compile the latex report
30
31         :param relative_dir: the relative dir towards the latex main
32             ↳ .tex file
33         :param latex_filename: name of the main latex .tex file that
34             ↳ manages the latex document
35         """
36
37         os.system(f'pdflatex {relative_dir}{latex_filename}')
38
39     def clean_up_after_compilation(self, latex_filename):
40         """Removes the unneeded files that were generated during
41             ↳ latex to pdf compilation.
42
43         :param latex_filename: name of the main latex .tex file that
44             ↳ manages the latex document
45         """
46
47         latex_filename_without_extention = latex_filename[:-4]
48         self.delete_file_if_exists(f'{
49             ↳ latex_filename_without_extention}.aux')
50         self.delete_file_if_exists(f'{
51             ↳ latex_filename_without_extention}.log')
52         self.delete_file_if_exists(f'texput.log')
53
54     def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
55         """Moves the compiled/generated pdf file from the root of
56             ↳ this repository to the
57             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 regardless 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)

```
1 from nbconvert.preprocessors import ExecutePreprocessor
2 import os
3 import nbformat
4
5 class Run_jupyter_notebook:
6     """runs a list of jupyter notebooks and converts it to pdf"""
7
8
9     def __init__(self):
10         self.script_dir = self.get_script_dir()
11
12
13     def run_jupyter_notebooks(self, project_nr, notebook_names):
14         """runs a jupyter notebook in this directory
15
16         :param project_nr: the numberr identifying which project is
17             ↪ being ran and compiled
18         :param notebook_names: list of strings with the names of the
19             ↪ notebooks that need to be ran
20
21         """
22         notebook_path = f'code/project{project_nr}/src/'
23
24         for notebook_name in notebook_names:
25             self.run_jupyter_notebook.run_notebook(f'{notebook_path}{
26                 ↪ notebook_name}')
27
28
29     def convert_notebooks_to_pdf(self, project_nr, notebook_names):
30         """converts a jupyter notebook to pdf
31
32         :param project_nr: the numberr identifying which project is
33             ↪ being ran and compiled
34         :param notebook_names: list of strings with the names of the
35             ↪ notebooks that need to be ran
36
37         """
38         notebook_path = f'code/project{project_nr}/src/'
39
40         for notebook_name in notebook_names:
41             self.run_jupyter_notebook.convert_notebook_to_pdf(f'{
42                 ↪ notebook_path}{notebook_name}')
43
44
45     def compile_latex_report(self, project_nr):
46         """compiles latex code to pdf
47
48         :param project_nr: the numberr identifying which project is
49             ↪ being ran and compiled
50
51         """
52         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
72         ↪ to pdf
73
74         :param notebook_filename: the name of the notebook that needs
75         ↪ to be compiled to pdf
76
77         """
78         os.system(f'jupyter nbconvert --to pdf {notebook_filename}')
79
80     def get_script_dir(self):
81         """returns the directory of this script regardless of from
82         ↪ which level the code is executed"""
83         return os.path.dirname(__file__)
84
85 if __name__ == '__main__':
86     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  
      latex_image_path = '../..../latex/project1/Images/'  
  
      # 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/s2]')
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')

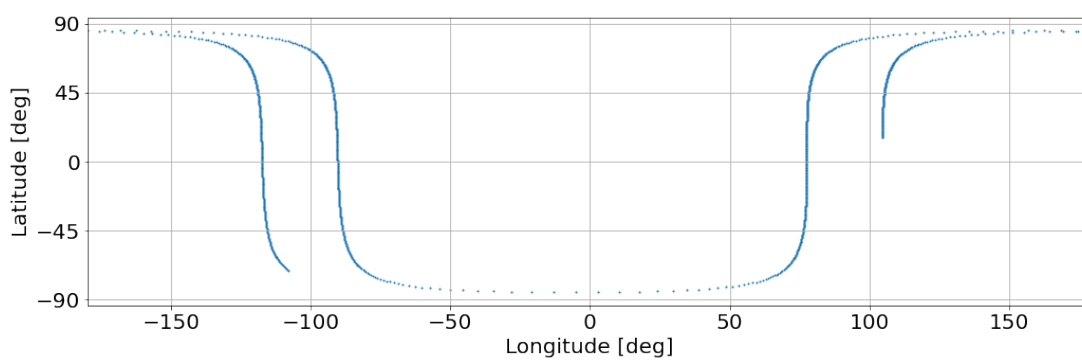
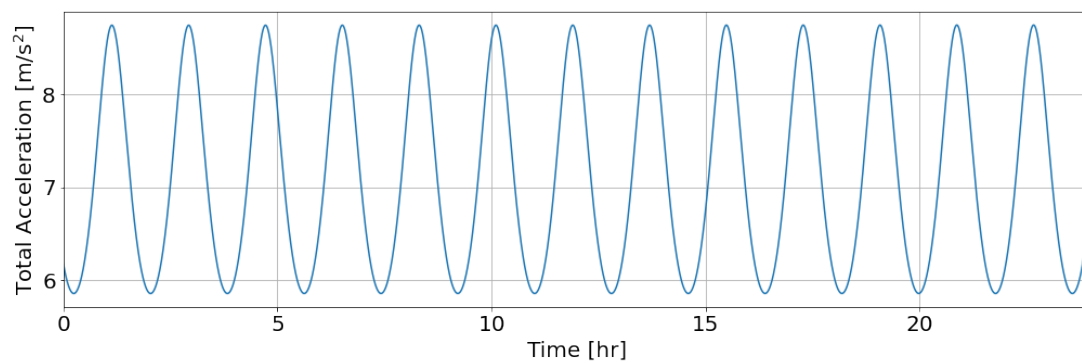
# Aerodynamic Acceleration Earth
acceleration_norm_aero_earth = dependent_variable_list[:, 16]
plt.plot( time_hours, acceleration_norm_aero_earth, label='Aerodynamic Earth')

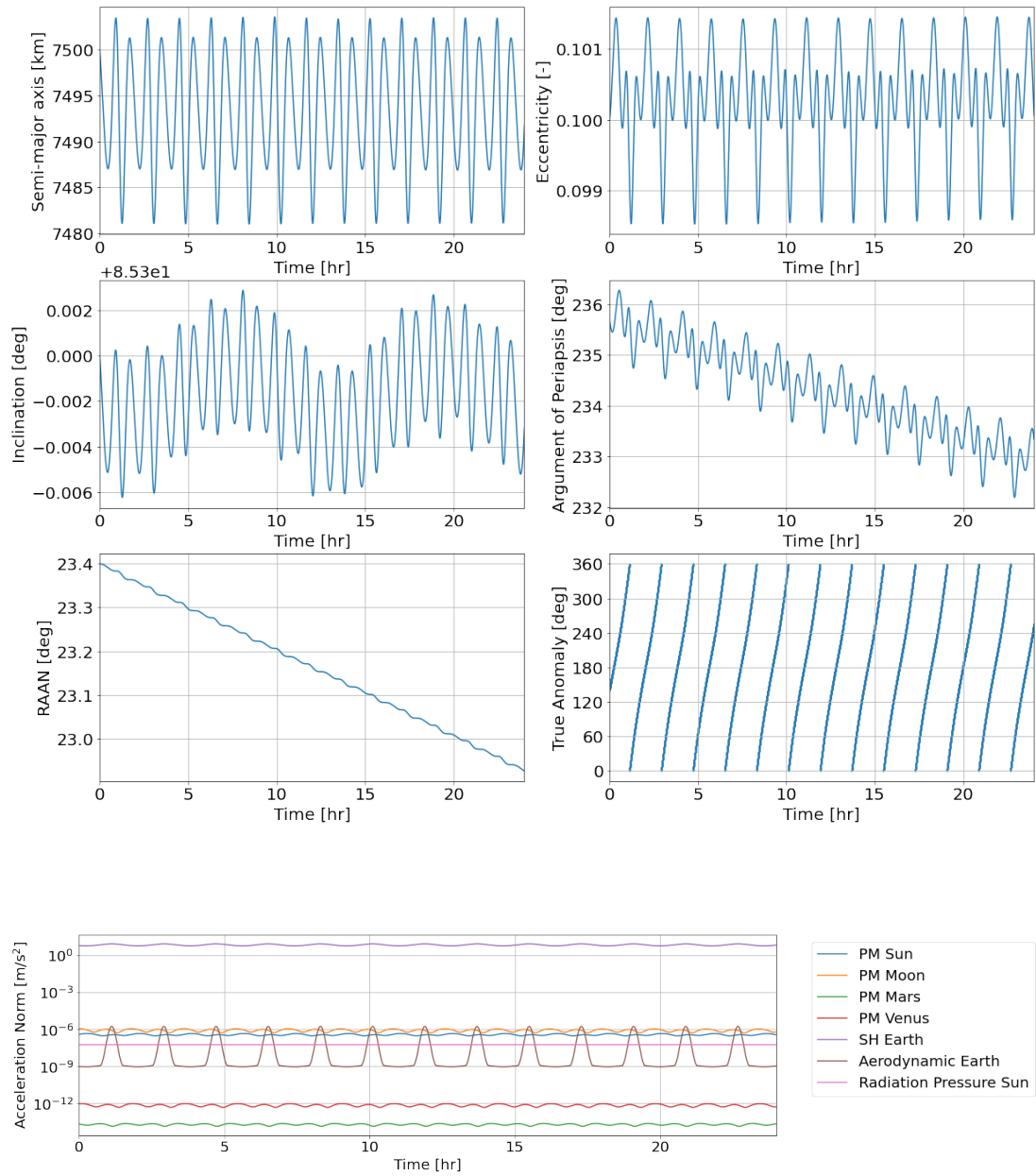
# Cannonball Radiation Pressure Acceleration Sun
acceleration_norm_rp_sun = dependent_variable_list[:, 17]
plt.plot( time_hours, acceleration_norm_rp_sun, label='Radiation Pressure Sun')

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/s2]' )

plt.savefig( fname = f'{latex_image_path}acceleration_norms.png',
    ↳bbox_inches='tight')
#plt.savefig('acceleration_norms.png', bbox_inches='tight')

```





[]: