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 **??**.

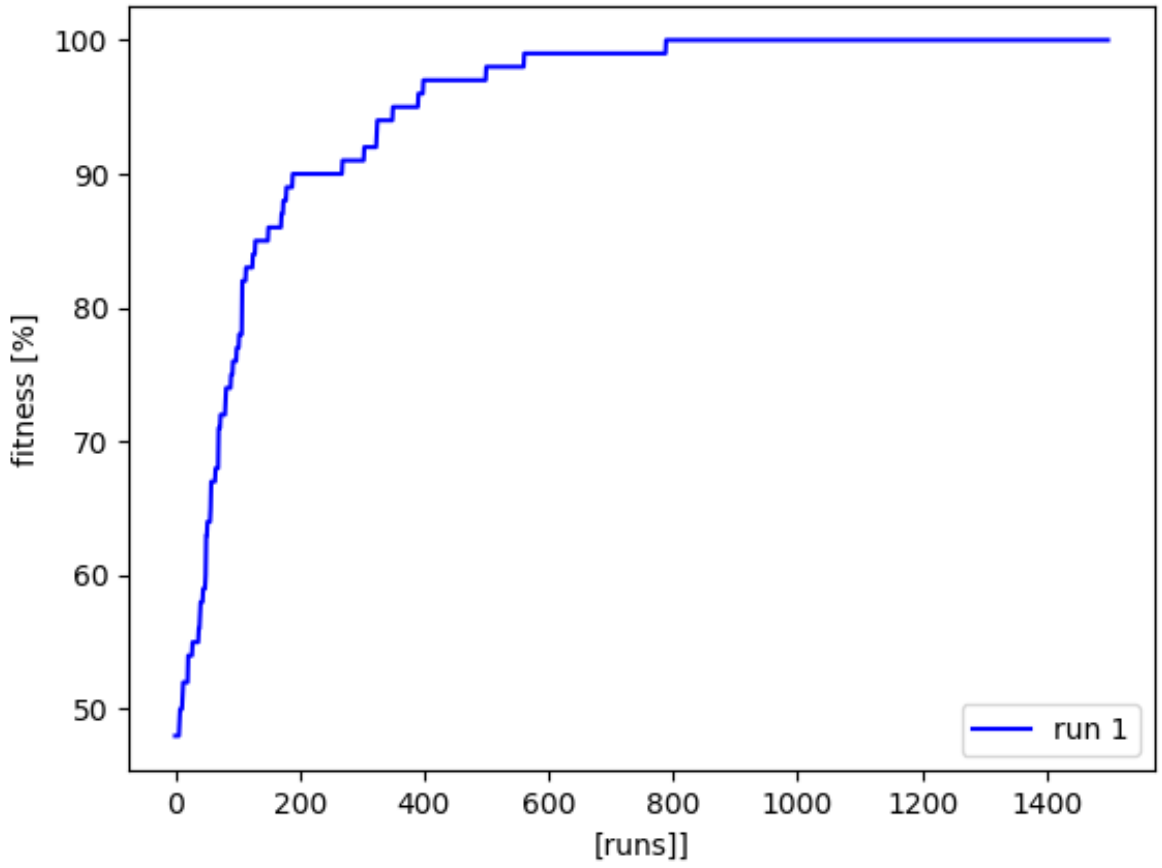


Figure 1: Performance of some genetic algorithm

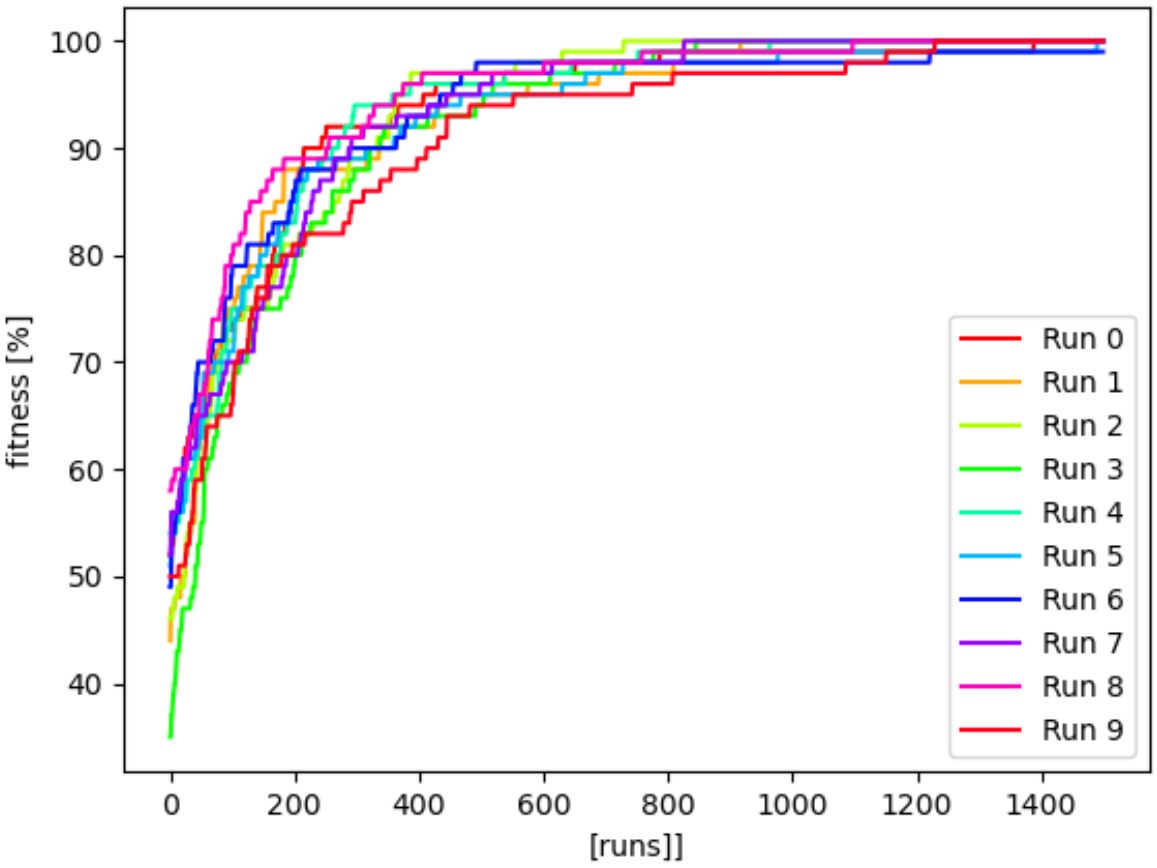


Figure 2: Performance of some genetic algorithm

A Appendix __main__.py

```
import os
from .Main import Main

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']
notebook_names = []# TODO: re-enable

# 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)

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

# 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("now running a")
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("now running b")
main.do4b(project_nr)

# run a genetic algorithm to create some data for another plot.
print("now running 4c")
main.do4c(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 .Run_jupyter_notebooks import Run_jupyter_notebook
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.run_jupyter_notebook = Run_jupyter_notebook()
        pass


    def run_jupyter_notebooks(self,project_nr,notebook_names):
        '''runs a jupyter notebook'''
        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'''
        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 export_code_to_latex(self, project_nr):
        export_code_to_latex(project_nr, 'main.tex')

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

    ##############################################################
    ############example code to illustrate python-latex  image sync #########
    ###############runs arbitrary genetic algorithm, can be deleted ###########
    ##############################################################
    def count(self,bits):
        count = 0
        for bit in bits:
            if bit:
                count = count + 1
        return count
```

```
57
58      def gen_bit_sequence(self):
59          bits = []
60          for _ in range(string_length):
61              bits.append(True if random.randint(0, 1) == 1 else False)
62          return bits
63
64      def mutate_bit_sequence(self,sequence):
65          retval = []
66          for bit in sequence :
67              do_mutation = random.random() <= mutation_chance
68              if(do_mutation):
69                  retval.append(not bit)
70              else:
71                  retval.append(bit)
72          return retval
73
74      #execute a run a
75      def do_run_a(self):
76
77          seq = self.gen_bit_sequence()
78          fitness = self.count(seq)
79          results = [fitness]
80          for run in range(max_iterations-1):
81              new_seq = self.mutate_bit_sequence(seq)
82              new_fitness = self.count(new_seq)
83              if new_fitness > fitness:
84                  seq = new_seq
85                  fitness = new_fitness
86              results.append(max(results[-1],fitness))
87          return results
88
89
90      #execute a run c
91      def do_run_c(self):
92          seq = self.gen_bit_sequence()
93          fitness = self.count(seq)
94          results = [fitness]
95          for run in range(max_iterations):
96              new_seq = self.mutate_bit_sequence(seq)
97              new_fitness = self.count(new_seq)
98              seq = new_seq
99              fitness = new_fitness
100             results.append(max(results[-1], fitness))
101         return results
102
103     def do4b(self,project_nr):
104         optimum_found = 0
105
106         # generate plot data
107         plotResult = np.zeros((10,max_iterations), dtype=int);
108         lineLabels = []
109
110         # perform computation
111         for run in range(10):
112             res = self.do_run_a()
113             if res[-1] == string_length:
114                 optimum_found +=1
115
116             # store computation data for plotting
117             lineLabels.append(f'Run {run}')
118             plotResult[run,:]=res;
```

```
119
120         # plot multiple lines into report (res is an array of
                ↪ dataseries (representing the lines))
121         # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
                ↪ axis label",lineLabels,"filename",legend_position,
                ↪ project_nr)
122         plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
                ↪ plotResult,"[runs]]","fitness [%]",lineLabels,"4b",4,
                ↪ project_nr)
123         print("total optimum found: {} out of {} runs".format(
                ↪ optimum_found,10))
124
125     def do4c(self,project_nr):
126         optimum_found = 0
127
128         # generate plot data
129         plotResult = np.zeros((10,max_iterations+1), dtype=int);
130         lineLabels = []
131
132         # perform computation
133         for run in range(10):
134             res = self.do_run_c()
135             if res[-1] == string_length:
136                 optimum_found +=1
137
138             # Store computation results for plot
139             lineLabels.append(f'Run {run}')
140             plotResult[run,:]=res;
141
142         # plot multiple lines into report (res is an array of
                ↪ dataseries (representing the lines))
143         # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
                ↪ axis label",lineLabels,"filename",legend_position,
                ↪ project_nr)
144         plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
                ↪ plotResult,"[runs]]","fitness [%]",lineLabels,"4c",4,
                ↪ project_nr)
145
146         print("total optimum found: {} out of {} runs".format(
                ↪ optimum_found, 10))
147
148     def addTwo(self,x):
149         ''' adds two to the incoming integer and returns the result
                ↪ of the computation.'''
150         return x+2
151
152 if __name__ == '__main__':
153     # initialize main class
154     main = Main()
```

C Appendix Compile_latex.py

```
# 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()
```

D Appendix Export_code_to_latex.py

```
# 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(project_nr,latex_filename):
        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}/{latex_filename}'
        root_dir = script_dir[0:script_dir.rfind(f'code/project{
           ↪ project_nr}')]

        python_filepaths = get_filenames_in_dir('py',script_dir, ['
           ↪ __init__.py'])
        compiled_notebook_pdf_filepaths = get_compiled_notebook_paths
           ↪ (script_dir)

        python_files_already_included_in_appendices =
           ↪ get_code_files_already_included_in_appendices('.py',
           ↪ python_filepaths, appendix_dir, project_nr, root_dir)
        #print(f'\n\npython_files_already_included_in_appendices={
           ↪ list(map(lambda x: x.filepath,
           ↪ python_files_already_included_in_appendices))}')
        notebook_pdf_files_already_included_in_appendices =
           ↪ get_code_files_already_included_in_appendices('.ipynb',
           ↪  compiled_notebook_pdf_filepaths, appendix_dir,
           ↪ project_nr, root_dir)
        #appendices = get_filenames_in_dir('.tex', appendix_dir)

        missing_python_files_in_appendices =
           ↪ get_code_files_not_yet_included_in_appendices('.py',
           ↪ python_files_already_included_in_appendices,
           ↪ python_filepaths)
        missing_notebook_files_in_appendices =
           ↪ get_code_files_not_yet_included_in_appendices('.pdf',
           ↪ notebook_pdf_files_already_included_in_appendices,
           ↪ compiled_notebook_pdf_filepaths)

        created_python_appendix_filenames =
           ↪ create_appendices_with_code('.py',
           ↪ missing_python_files_in_appendices, appendix_dir,
           ↪ project_nr, root_dir)
        created_notebook_appendix_filenames =
           ↪ create_appendices_with_code('.ipynb',
           ↪ missing_notebook_files_in_appendices, appendix_dir,
           ↪ project_nr, root_dir)
        # create_appendices_with_notebook_pdfs()
        appendices = get_list_of_appendix_files(appendix_dir,
           ↪ python_filepaths, compiled_notebook_pdf_filepaths,
           ↪ project_nr, root_dir)
        #print(f'appendices={appendices}')

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

```
34          non_code_appendices, non_code_appendix_lines =
               ↪ get_order_of_non_code_appendices_in_main(
               ↪ appendix_tex_code,appendices) # assumes non-included
               ↪ non-code appendices should not be included.
35        # create intended order of appendices
36
37          # TODO: include appendices even if they are not newly created
               ↪  but still missing in main
38          python_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, get_appendices_of_type(appendices, '
               ↪ python')))
39          sorted_created_python_appendices = sort_python_appendices(
               ↪ get_appendices_of_type(appendices, 'python'))
40          sorted_python_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, sorted_created_python_appendices))
41          print(f'sorted_python_appendix_filenames ={
               ↪ sorted_python_appendix_filenames }')
42
43          notebook_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, get_appendices_of_type(appendices, '
               ↪ notebook')))
44          sorted_created_notebook_appendices = sort_notebook_appendices
               ↪ (get_appendices_of_type(appendices, 'notebook'))
45          sorted_notebook_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, sorted_created_notebook_appendices))
46          print(f'sorted_notebook_appendix_filenames ={
               ↪ sorted_notebook_appendix_filenames}')
47
48
49          appendix_latex_code = create_appendices_latex_code(
               ↪ non_code_appendix_lines,
               ↪ sorted_created_python_appendices,
               ↪ sorted_created_notebook_appendices, project_nr)
50
51          #updated_appendices_tex_code = update_appendix_tex_code(
               ↪ appendix_tex_code,created_python_appendix_filenames,
               ↪ project_nr)
52          #updated_appendices_tex_code = update_appendix_tex_code(
               ↪ updated_appendices_tex_code,
               ↪ created_notebook_appendix_filenames, project_nr)
53
54          updated_main_tex_code = substitute_appendix_code(
               ↪ main_tex_code, start_index, end_index,
               ↪ appendix_latex_code)
55
56          overwrite_content_to_file(path_to_main_latex_file,
               ↪ updated_main_tex_code)
57
58  def create_appendices_latex_code(non_code_lines, python_appendices,
       ↪ notebook_appendices, project_nr):
59      ''' creates the appendix text for main.'''
60      appendix_latex_code = non_code_lines
61      for appendix in python_appendices:
62          line = update_appendix_tex_code(appendix.appendix_filename,
               ↪ project_nr)
63          appendix_latex_code.append(line)
64
65      for appendix in notebook_appendices:
66          line = update_appendix_tex_code(appendix.appendix_filename,
               ↪ project_nr)
67          appendix_latex_code.append(line)
68      print(f'appendix_latex_code={appendix_latex_code}')
```

```
    return appendix_latex_code

def get_appendices_of_type(appendices, appendix_type):
    ''' Returns the list of appendices of certain type from a list of appendix objects.'''
    return_appendices = []
    for appendix in appendices:
        if appendix.appendix_type == appendix_type:
            return_appendices.append(appendix)
    return return_appendices

def sort_python_appendices(appendices):
    ''' First puts __main__.py, followed by main.py followed by a-z code files.'''
    return_appendices = []
    for appendix in appendices:
        print(f'appendix.appendix_filename={appendix.filename_of_code_file}')
        if (appendix.filename_of_code_file=="__main__.py") or (appendix.filename_of_code_file=="__Main__.py"):
            return_appendices.append(appendix)
            appendices.remove(appendix)
    for appendix in appendices:
        if (appendix.filename_of_code_file=="main.py") or (appendix.filename_of_code_file=="Main.py"):
            return_appendices.append(appendix)
            appendices.remove(appendix)
    return_appendices

    filtered_remaining_appendices = [i for i in appendices if i.filename_of_code_file is not None]
    print(f'filtered_remaining_appendices={list(map(lambda x: x.filename_of_code_file, filtered_remaining_appendices))}')
    appendices_sorted_a_z = filter_list_on_property(filtered_remaining_appendices)
    print(f'sorted_a_z={list(map(lambda x: x.filename_of_code_file, appendices_sorted_a_z))}')
    return return_appendices+appendices_sorted_a_z


def sort_notebook_appendices(appendices):
    ''' Sorts notebooks on a-z pdf filenames.'''
    return_appendices = []
    filtered_remaining_appendices = [i for i in appendices if i.filename_of_code_file is not None]
    print(f'filtered_remaining_appendices={list(map(lambda x: x.filename_of_code_file, filtered_remaining_appendices))}')
    appendices_sorted_a_z = filter_list_on_property(filtered_remaining_appendices)
    print(f'sorted_a_z={list(map(lambda x: x.filename_of_code_file, appendices_sorted_a_z))}')
    return return_appendices+appendices_sorted_a_z


def filter_list_on_property(some_list):
    ''' Returns a list based on the property: filename_of_code_file '''
    # TODO: pass property to generalize method
    # customObjects.sort(key=lambda x: x.date, reverse=True) gives error of Nonetype on this list
    attributes = list(map(lambda x: x.filename_of_code_file, some_list))
```

```
115     sorted_indices = sorted(range(len(attributes)), key=lambda k:
          ↪ attributes[k])
116     sorted_list = []
117     for i in sorted_indices:
118         sorted_list.append(some_list[i])
119     return sorted_list
120 
121 def get_order_of_non_code_appendices_in_main(appendix_tex_code,
      ↪ appendices):
122     ''' Scans the lines of appendices in the main code, and returns
          ↪ the lines that
123     of appendices that do not contain code, in specified order.'''
124     non_code_appendices = []
125     non_code_appendix_lines = []
126     appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
127     for line in appendix_tex_code:
128         appendix_filename = get_filename_from_latex_appendix_line(
              ↪ line, appendices)
129         print(f'appendix_filename={appendix_filename}')
130 
131         # Check if line is not commented
132         if not appendix_filename is None:
133             if not line_is_commented(line,appendix_filename):
134                 print(f'line={line}')
135                 appendix = get_appendix_from_filename(
                      ↪ appendix_filename, appendices)
136                 if appendix.appendix_type == "no_code":
137                     non_code_appendices.append(appendix)
138                     non_code_appendix_lines.append(line)
139     return non_code_appendices, non_code_appendix_lines
140 
141 
142 def get_filename_from_latex_appendix_line(line,appendices):
143     for filename in list(map(lambda appendix: appendix.
          ↪ appendix_filename, appendices)):
144         if filename in line:
145             return filename
146 
147 def get_appendix_from_filename(appendix_filename,appendices):
148     for appendix in appendices:
149         if appendix_filename == appendix.appendix_filename:
150             return appendix
151 
152 def get_compiled_notebook_paths(script_dir):
153     ''' Returns the list of jupiter notebook filepaths that were
          ↪ compiled successfully'''
154     notebook_filepaths= get_filenames_in_dir('.ipynb', script_dir)
155     compiled_notebook_filepaths = []
156 
157     # check if the jupyter notebooks were compiled
158     for notebook_filepath in notebook_filepaths:
159 
160         # swap file extension
161         notebook_filepath = notebook_filepath.replace('.ipynb','.pdf'
              ↪ )
162 
163         # check if file exists
164         if os.path.isfile(notebook_filepath):
165             compiled_notebook_filepaths.append(notebook_filepath)
166     return compiled_notebook_filepaths
167 
168 
```

```
169 def get_list_of_appendix_files(appendix_dir,
    ↪ absolute_python_filepaths, absolute_notebook_filepaths,
    ↪ project_nr, root_dir):
170     ''' Returns a list with all the appendix files with .tex
        ↪ extension.'''
171     appendices = []
172     appendices_paths = get_filenames_in_dir('.tex', appendix_dir)
173 
174     for appendix_filepath in appendices_paths:
175         appendix_type = "no_code"
176         appendix_filecontent = read_file(appendix_filepath)
177         line_nr_python_file_inclusion = get_line_of_latex_command(
            ↪ appendix_filecontent, "\pythonexternal{")
178         line_nr_notebook_file_inclusion = get_line_of_latex_command(
            ↪ appendix_filecontent, "\includepdf[pages=")
179         if  line_nr_python_file_inclusion > -1:
180             appendix_type = "python"
181             # get python filename
182             line = appendix_filecontent[line_nr_python_file_inclusion
                ↪ ]
183             filename = get_filename_from_latex_inclusion_command('.py
                ↪ ', line, "\pythonexternal{")
184             appendices.append(Appendix(appendix_filepath,
                ↪ appendix_filecontent, appendix_type, filename, line
                ↪ ))
185         if line_nr_notebook_file_inclusion > -1:
186             appendix_type = "notebook"
187             line = appendix_filecontent[
                ↪ line_nr_notebook_file_inclusion]
188             filename = get_filename_from_latex_inclusion_command('.
                ↪ pdf', line, "\includepdf[pages=")
189             appendices.append(Appendix(appendix_filepath,
                ↪ appendix_filecontent, appendix_type, filename, line
                ↪ ))
190         else:
191             appendices.append(Appendix(appendix_filepath,
                ↪ appendix_filecontent, appendix_type))
192     #return get_filenames_in_dir('.tex', appendix_dir)
193     return appendices
194 
195 def get_filename_from_latex_inclusion_command(extension, line,
    ↪ start_substring):
196     ''' returns the filename in a latex inclusion command.'''
197     start_index = line.index(start_substring)
198     end_index = line.index(extension)
199     return get_filename_from_dir(line[start_index:end_index+len(
        ↪ extension)])
200 
201 def get_filenames_in_dir(extension, folder, excluded_files=None):
202     '''Returns a list of the relative paths to all files within the
        ↪ code/projectX/src/ folder that match
203     the given file extension.'''
204     filepaths=[]
205     for r, d, f in os.walk(folder):
206         for file in f:
207             if file.endswith(extension):
208                 if (excluded_files is None) or ((not excluded_files
                    ↪ is None) and (not file in excluded_files)):
209                     filepaths.append(r+'/'+file)
210     return filepaths
211 
212 # def check_if_is_excluded_file(filename,excluded_files):
```

```
213     # ''' Retruns true if the file is in the excluded file list,
            ↪ returns false otherwise.'''
214     # if filename in
215
216 def get_code_files_already_included_in_appendices(extension,
        ↪ absolute_filepaths, appendix_dir, project_nr, root_dir):
217     ''' Returns a list of filepaths that are already properly
            ↪ included in some appendix of this projectX,'''
218     # TODO: change search string for python and jupyter notebook
219     appendix_files = get_filenames_in_dir('.tex', appendix_dir)
220     contained_codes = []
221     for code_filepath in absolute_filepaths:
222         for appendix_filepath in appendix_files:
223             appendix_filecontent = read_file(appendix_filepath)
224             line_nr = check_if_appendix_contains_file(extension,
                    ↪ code_filepath, appendix_filecontent, project_nr,
                    ↪ root_dir)
225             if line_nr>-1:
226                 # add filepath to list of files that are already in
                        ↪ the appendices
227                 contained_codes.append(Appendix_with_code(
                        ↪ code_filepath,
228                 appendix_filepath,
229                 appendix_filecontent,
230                 line_nr,
231                 '.py'))
232     return contained_codes
233
234
235 def check_if_appendix_contains_file(extension, code_filepath,
        ↪ appendix_content, project_nr, root_dir):
236     ''' scans an appendix content to determine whether it contains a
            ↪ substring that
237     includes the python code file.'''
238     # TODO: write tests
239     # convert code_filepath to the inclusion format in latex format
240     latex_relative_filepath = f'latex/project{project_nr}/../../{
            ↪ code_filepath[len(root_dir):]}' # TODO: rename to indicate
            ↪ filepath of what
241     latex_command = get_latex_inclusion_command(extension,
            ↪ latex_relative_filepath, project_nr)
242     return get_line_of_latex_command(appendix_content, latex_command)
243
244 def get_line_of_latex_command(appendix_content, latex_command):
245     ''' Returns the line number of a latex command if it is found.
            ↪ Returns -1 otherwise.'''
246     # check if the file is in the latex code
247     line_nr = 0
248     for line in appendix_content:
249         if latex_command in line:
250             if line_is_commented(line,latex_command):
251                 commented=True
252             else:
253                 return line_nr
254         line_nr=line_nr+1
255     return -1
256
257
258 def line_is_commented(line,target_substring):
259     ''' Returns true if a line is commented, returns false otherwise
            ↪ '''
260     left_of_command = line[:line.rfind(target_substring)]
```

```
261     if '%' in left_of_command:
262         return True
263     return False
264 
265     # return true with filename, line_number and line
266     # return false
267 
268 def get_latex_inclusion_command(extension, latex_relative_filepath,
    ↪ project_nr):
269     if extension==".py":
270         left = "\pythonexternal{"
271         right = "}"
272         latex_command = f'{left}{latex_relative_filepath}{right}'
273     elif extension==".ipynb":
274 
275         left = "\includepdf[pages=-]{"
276         right = "}"
277         latex_command = f'{left}{latex_relative_filepath}{right}'
278     return latex_command
279 
280 def read_file(filepath):
281     ''' Reads content of a file and returns it as a list of strings
        ↪ '''
282     with open(filepath) as f:
283         content = f.readlines()
284     # you may also want to remove whitespace characters like `\n` at
        ↪ the end of each line
285     #content = [x.strip() for x in content]
286     return content
287 
288 
289 def get_code_files_not_yet_included_in_appendices(extension,
    ↪ contained_codes, code_filepaths):
290     ''' Returns a list of filepaths that are not yet properly
        ↪ included in some appendix of this projectX,'''
291     contained_filepaths = list(map(lambda contained_file:
        ↪ contained_file.filepath, contained_codes))
292     not_contained = []
293     for filepath in code_filepaths:
294         if not filepath in contained_filepaths:
295             not_contained.append(filepath)
296     return not_contained
297 
298 
299 def create_appendices_with_code(extension, code_filepaths,
    ↪ appendix_dir, project_nr,root_dir):
300     ''' Creates the latex appendix files in with relevant codes
        ↪ included.'''
301     appendix_filenames = []
302     appendix_reference_index = 0
303 
304     for code_filepath in code_filepaths:
305         latex_relative_filepath = f'latex/project{project_nr}/../../{
            ↪ code_filepath[len(root_dir):]}' # TODO: rename to
            ↪ indicate filepath of what # TODO: move out of loop for
            ↪ lower complexity
306         content = []
307         filename = get_filename_from_dir(code_filepath)
308         content = create_section(content,filename,
            ↪ appendix_reference_index)
309         inclusion_command = get_latex_inclusion_command(extension,
            ↪ latex_relative_filepath, project_nr)
```

```
310             content.append(inclusion_command)
311             overwrite_content_to_file(f'{appendix_dir}Auto_generated_{
                    ↪ extension[1:]}_App{appendix_reference_index}.tex',
                    ↪ content, False)
312             appendix_filenames.append(f'Auto_generated_{extension[1:]}
                    ↪ _App{appendix_reference_index}.tex')
313             appendix_reference_index = appendix_reference_index+1
314         return appendix_filenames
315
316     def create_section(content,filename, appendix_reference_index):
317         # write section
318         left ="\section{Appendix "
319         middle = filename.replace("_","\_")
320         right = "}\label{app:"
321         end = "}" # TODO: update appendix reference index
322         content.append(f'{left}{middle}{right}{appendix_reference_index}{
                ↪ end}')
323         return content
324
325
326     def overwrite_content_to_file(filepath, content, has_newlines=True):
327         ''' Writes the content of an appendix to a new appendix'''
328         with open(filepath,'w') as f:
329             for line in content:
330                 if has_newlines:
331                     f.write(line)
332                 else:
333                     f.write(line+'\n')
334
335
336     def verify_notebook_pdf_exists(relative_file_path):
337         ''' Returns True if a compiled pdf of the listed Jupyter notebook
                ↪  exists
338         that can be included in the latex as appendix. Returns False
                ↪ otherwise.'''
339         pass
340
341
342     def get_list_of_appendices_with_code(code_format,relative_paths):
343         ''' Returns a list of all the appendices that are available that
                ↪ contain code'''
344         pass
345
346
347     def get_appendix_tex_code(main_filename):
348         ''' gets the latex appendix code from the main tex file.'''
349         main_tex_code = read_file(main_filename)
350         start =  '\\begin{appendices}' # TODO: scan for % in front
351         end = "\end{appendices}" # TODO: scan for % in front
352         start_index = get_index_of_substring_in_list(start,main_tex_code)
                ↪ +1
353         end_index = get_index_of_substring_in_list(end,main_tex_code)
354         return main_tex_code,start_index,end_index,main_tex_code[
                ↪ start_index:end_index]
355
356     def get_index_of_substring_in_list(substring, lines):
357         for i in range(0, len(lines)):
358             if substring in lines[i]:
359                 return i
360
361
362     def update_appendix_tex_code(appendix_filename, project_nr):
```

```
363     ''' Includes the appendices as latex commands in the tex code string'''
364     #return_lines = appendix_tex_code
365     #f'{appendix_dir}Auto_generated_{extension[1:]}App{appendix_reference_index}.tex',content, False)
366     left = "\input{latex/project"
367     middle = "/Appendices/"
368     right = "} \\newpage\n"
369     #return_lines.append(f'{left}{project_nr}{middle}{appendix_filename}{right}')
370     return f'{left}{project_nr}{middle}{appendix_filename}{right}'
371
372
373 def substitute_appendix_code(main_tex_code, start_index, end_index, updated_appendices_tex_code):
374     ''' Replaces the old latex code that include the appendices with the new latex
375     commands that include the appendices in the latex report.'''
376     updated_main_tex_code = main_tex_code[0:start_index]+updated_appendices_tex_code+main_tex_code[end_index:]
377     print(f'updated_main_tex_code={updated_main_tex_code}')
378     return updated_main_tex_code
379
380
381
382 def compile_latex(relative_dir,latex_filename):
383     os.system(f'pdflatex {relative_dir}{latex_filename}')
384
385 def clean_up_after_compilation(latex_filename):
386     latex_filename_without_extention = latex_filename[:-4]
387     delete_file_if_exists(f'{latex_filename_without_extention}.aux')
388     delete_file_if_exists(f'{latex_filename_without_extention}.log')
389     delete_file_if_exists(f'texput.log')
390
391 def move_pdf_into_latex_dir(relative_dir,latex_filename):
392     pdf_filename = f'{latex_filename[:-4]}.pdf'
393     destination= f'{get_script_dir()}/../../../{relative_dir}{pdf_filename}'
394
395     try:
396         shutil.move(pdf_filename, destination)
397     except:
398         print("Error while moving file ", pdf_filename)
399
400
401 def delete_file_if_exists(filename):
402     try:
403         os.remove(filename)
404     except:
405         print(f'Error while deleting file: {filename} but that is not too bad because the intention is for it to not be there.')
406
407 def get_filename_from_dir(path):
408     return path[path.rfind("/")+1:]
409
410 def get_script_dir():
411     ''' returns the directory of this script regardles of from which level the code is executed '''
412     return os.path.dirname(__file__)
413
414
```

```
415 class Appendix_with_code:
416     ''' stores in which appendix file and accompanying line number in
        ↪  the appendix in which a code file is
417     already included. Does not take into account whether this
        ↪ appendix is in the main tex file or not'''
418     def __init__(self, filepath,appendix_path,appendix_content,
        ↪ file_line_nr, extension):
419         self.filepath = filepath
420         self.appendix_path = appendix_path
421         self.appendix_content = appendix_content
422         self.file_line_nr = file_line_nr
423         self.extension = extension
424 
425 class Appendix:
426     ''' stores in appendix files and type of appendix.'''
427     # TODO: refactor remove the appendix_ cause that's what the
        ↪ object already implies
428     def __init__(self, appendix_path,appendix_content, appendix_type,
        ↪  filename_of_code_file=None, appendix_inclusion_line=None):
429         self.appendix_path = appendix_path
430         self.appendix_filename = get_filename_from_dir(self.
            ↪ appendix_path)
431         self.appendix_content = appendix_content
432         self.appendix_type = appendix_type # TODO: perform validation
            ↪  of input values
433         self.filename_of_code_file=filename_of_code_file
434         self.appendix_inclusion_line = appendix_inclusion_line
435 
436 class Appendix_without_code:
437     ''' stores in which appendix file that does not contain code.'''
438     def __init__(self, appendix_path,appendix_content):
439         self.appendix_path = appendix_path
440         self.appendix_content = appendix_content
```

E Appendix Plot_to_tex.py

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

    # plot graphs
    def plotMultipleLines(self,x,y_series,x_label,y_label,label, filename,legendPosition,project_nr):
        fig=plt.figure();
        ax=fig.add_subplot(111);
```

```
49
50          # generate colours
51          cmap = self.get_cmap(len(y_series[:,0]))
52
53          # generate line types
54          lineTypes = self.generateLineTypes(y_series)
55
56          for i in range(0,len(y_series)):
57              # overwrite linetypes to single type
58              lineTypes[i] = "-"
59              ax.plot(x,y_series[i,:],ls=lineTypes[i],label=label[i],
                    ↪ fillstyle='none',c=cmap(i)); # color
60
61          # configure plot layout
62          plt.legend(loc=legendPosition);
63          plt.xlabel(x_label);
64          plt.ylabel(y_label);
65          plt.savefig(os.path.dirname(__file__)+'/../../../latex/
                ↪ project'+str(project_nr)+'/Images/'+filename+'.png');
66
67          print(f'plotted lines')
68
69      # Generate random line colours
70      # Source: https://stackoverflow.com/questions/14720331/how-to-
            ↪ generate-random-colors-in-matplotlib
71      def get_cmap(n, name='hsv'):
72          '''Returns a function that maps each index in 0, 1, ..., n-1
                ↪ to a distinct
73          RGB color; the keyword argument name must be a standard mpl
                ↪ colormap name.'''
74          return plt.cm.get_cmap(name, n)
75
76      def generateLineTypes(y_series):
77          # generate varying linetypes
78          typeOfLines = list(lines.lineStyles.keys())
79
80          while(len(y_series)>len(typeOfLines)):
81              typeOfLines.append("-.");
82
83          # remove void lines
84          for i in range(0, len(y_series)):
85              if (typeOfLines[i]=='None'):
86                  typeOfLines[i]='-'
87              if (typeOfLines[i]==''):
88                  typeOfLines[i]=':'
89              if (typeOfLines[i]==' '):
90                  typeOfLines[i]='--'
91          return typeOfLines
92
93      # Create a table with: table_matrix = np.zeros((4,4),dtype=object
            ↪ ) and pass it to this object
94      def put_table_in_tex(self, table_matrix,filename,project_nr):
95          cols = np.shape(table_matrix)[1]
96          format = "%s"
97          for col in range(1,cols):
98              format = format+" & %s"
99          format = format+""
100         plt.savetxt(os.path.dirname(__file__)+"/../../../latex/
                ↪ project"+str(project_nr)+"/tables/"+filename+".txt",
                ↪ table_matrix, delimiter=' & ', fmt=format, newline='
                ↪ \\\\ \hline \n')
101
```

```
102     # replace this with your own table creation and then pass it to
            ↪ put_table_in_tex(..)
103     def example_create_a_table(self):
104         project_nr = "1"
105         table_name = "example_table_name"
106         rows = 2;
107         columns = 4;
108         table_matrix = np.zeros((rows,columns),dtype=object)
109         table_matrix[:,:]="" # replace the standard zeros with emtpy
                ↪ cell
110         print(table_matrix)
111         for column in range(0,columns):
112             for row in range(0,rows):
113                 table_matrix[row,column]=row+column
114         table_matrix[1,0]="example"
115         table_matrix[0,1]="grid sizes"
116 
117         self.put_table_in_tex(table_matrix,table_name,project_nr)
118 
119 
120     def get_script_dir(self):
121         ''' returns the directory of this script regardles of from
                ↪ which level the code is executed '''
122         return os.path.dirname(__file__)
123 
124 if __name__ == '__main__':
125     main = Plot_to_tex()
126     main.example_create_a_table()
```

F Appendix Run_jupyter_notebooks.py

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

import os
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor

class Run_jupyter_notebook:

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

    # runs jupyter notebook
    def run_notebook(self,notebook_filename):


        # Load your notebook
        with open(notebook_filename) as f:
            nb = nbformat.read(f, as_version=4)

        # Configure
        ep = ExecutePreprocessor(timeout=600, kernel_name='python3')

        # Execute
        ep.preprocess(nb, {'metadata': {'path': f'{self.
           ↪ get_script_dir()}/../../../'}})

        # Save output notebook
        with open(notebook_filename, 'w', encoding='utf-8') as f:
            nbformat.write(nb, f)

    # converts jupyter notebook to pdf
    def convert_notebook_to_pdf(self,notebook_filename):
        os.system(f'jupyter nbconvert --to pdf {notebook_filename}')

    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 = Run_jupyter_notebook()
```

Appendix Example Jupyter Notebook

AE4868_example_notebook_update20201025

December 26, 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
```

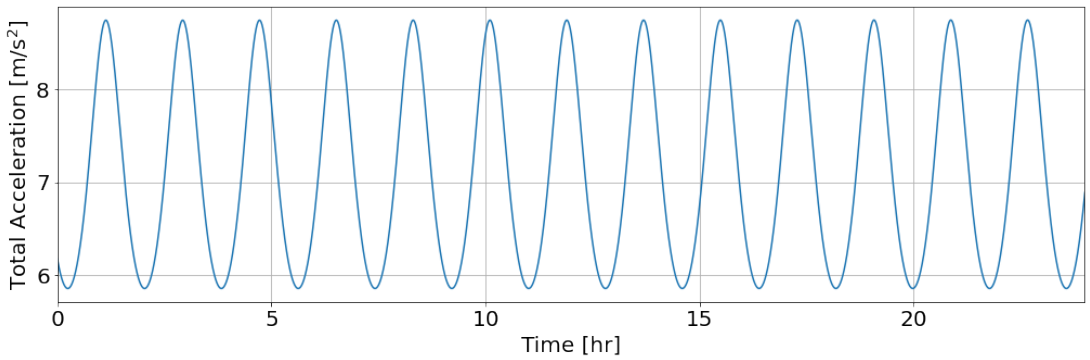
```
integrator_settings = propagation_setup.integrator.runge_kutta_4(
    simulation_start_epoch,
    fixed_step_size
)

################################################################################
# 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}
    """
)
```

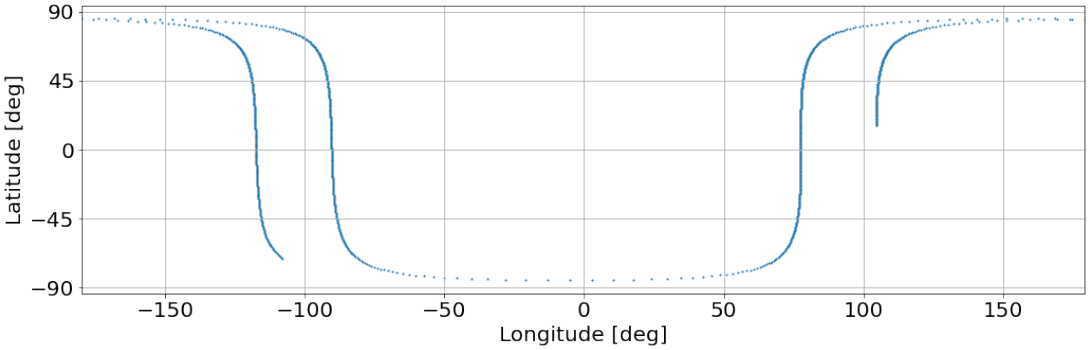
```
Single Earth-Orbiting Satellite Example.
The initial position vector of Delfi-C3 is [km]:
[7037.48400133 3238.05901792 2150.7241875 ]
The initial velocity vector of Delfi-C3 is [km/s]:
[-1.46565763 -0.04095839  6.62279761]
After 86400.0 seconds the position vector of Delfi-C3 is [km]:
[-4602.79426676 -1421.16740978  5883.69740624]
And the velocity vector of Delfi-C3 is [km/s]:
[-4.53846052 -2.36988263 -5.04163195]
```

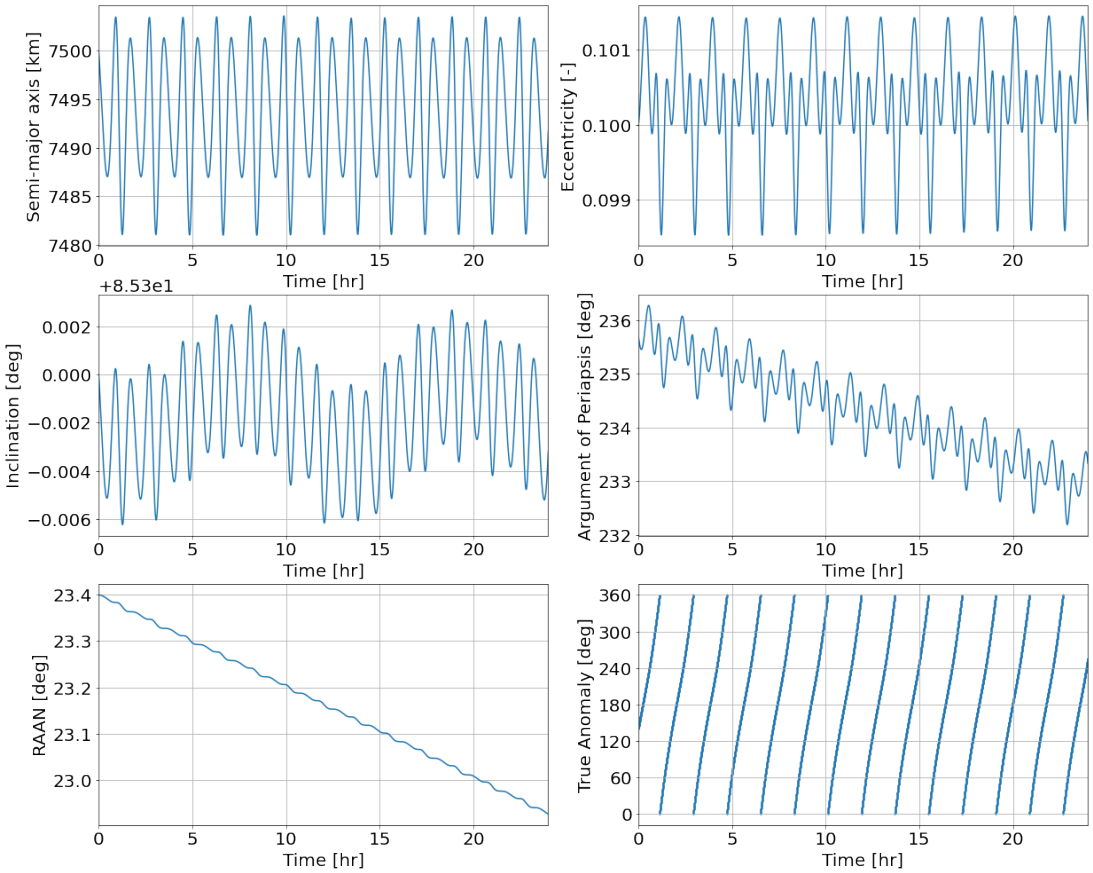
[3]:
```
import os
from matplotlib import pyplot as plt

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/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 )
```

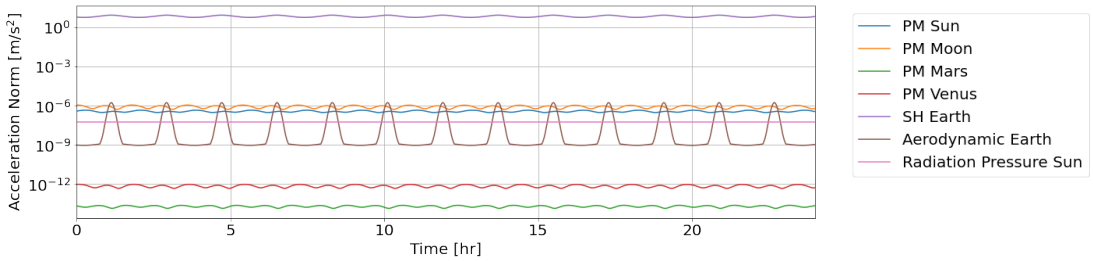
```
plt.xlabel('Longitude [deg]')
plt.ylabel( 'Latitude [deg]')
plt.xlim( [min(longitude), max(longitude)] )
plt.savefig( fname = f'{latex_image_path}ground_track.png', bbox_inches='tight')

# 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')
```


Total Acceleration [m/s^2]
8
7
6
0
5
10
15
20
Time [hr]


90
45
0
−45
−90
Latitude [deg]
−150
−100
−50
0
50
100
150
Longitude [deg]


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

[ ]:

G Appendix test_add.pdf

AE4868_example_notebook_update20201025

December 26, 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
```

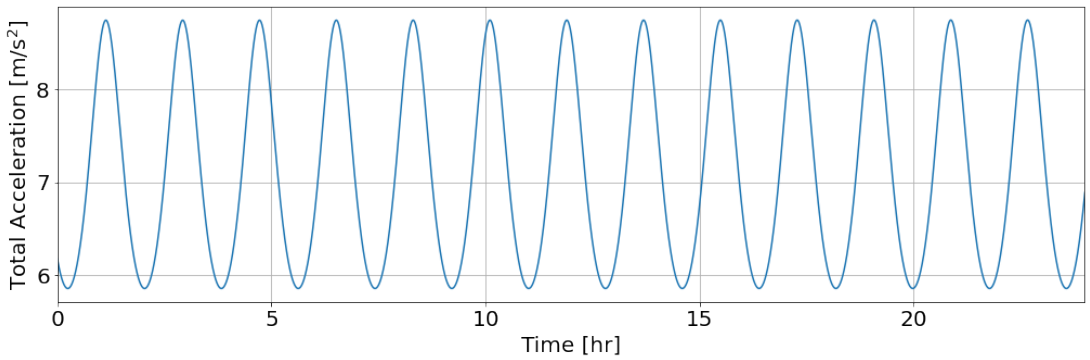
```
integrator_settings = propagation_setup.integrator.runge_kutta_4(
    simulation_start_epoch,
    fixed_step_size
)

################################################################################
# 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}
    """
)
```

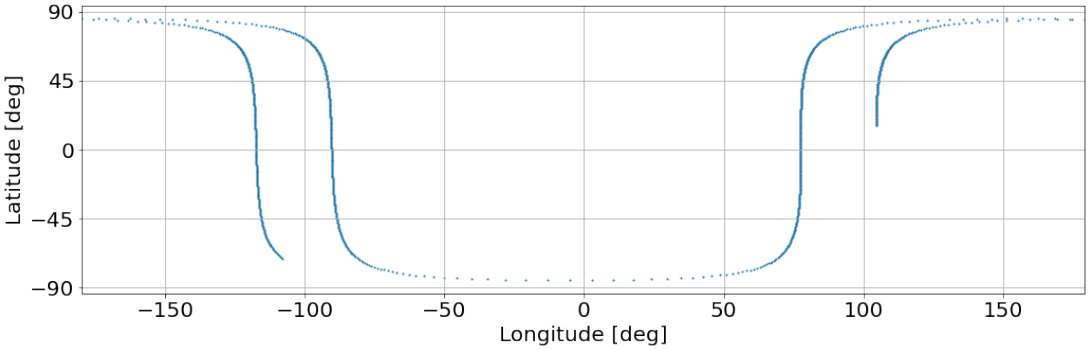
```
Single Earth-Orbiting Satellite Example.
The initial position vector of Delfi-C3 is [km]:
[7037.48400133 3238.05901792 2150.7241875 ]
The initial velocity vector of Delfi-C3 is [km/s]:
[-1.46565763 -0.04095839  6.62279761]
After 86400.0 seconds the position vector of Delfi-C3 is [km]:
[-4602.79426676 -1421.16740978  5883.69740624]
And the velocity vector of Delfi-C3 is [km/s]:
[-4.53846052 -2.36988263 -5.04163195]
```

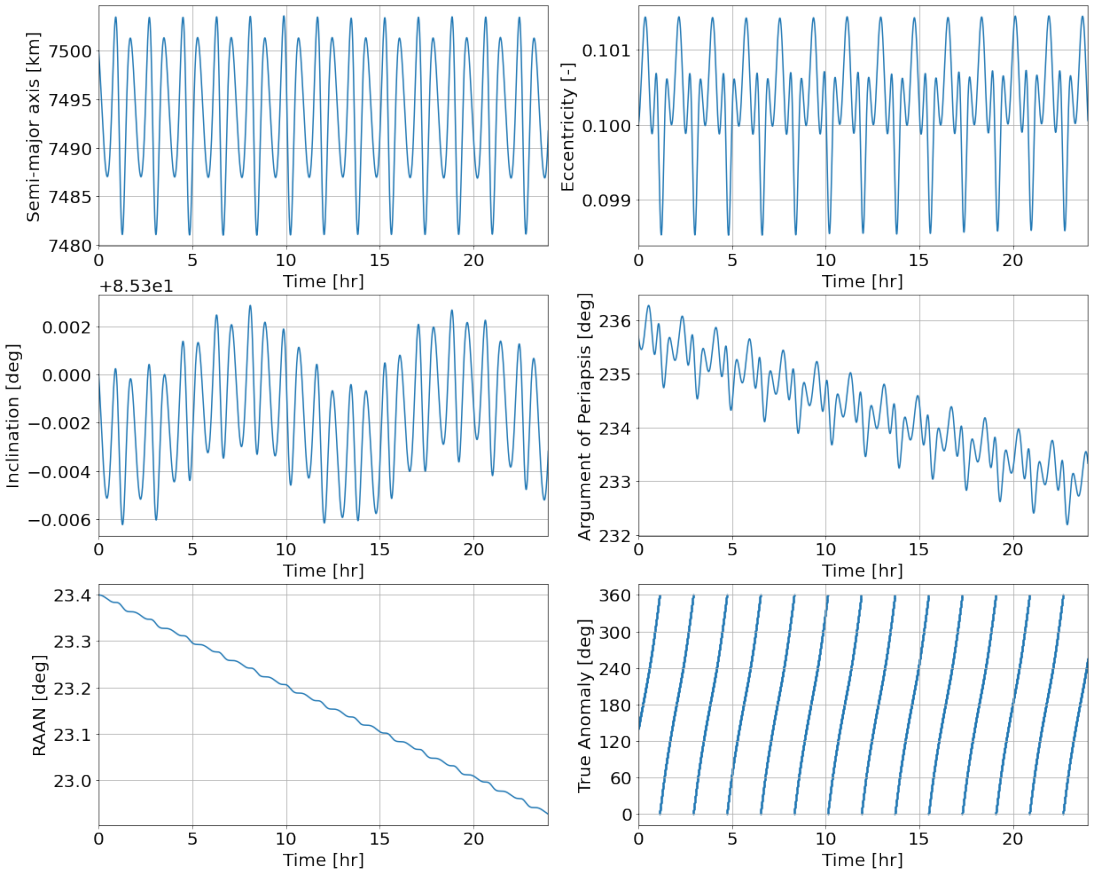
[3]:
```
import os
from matplotlib import pyplot as plt

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/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 )
```

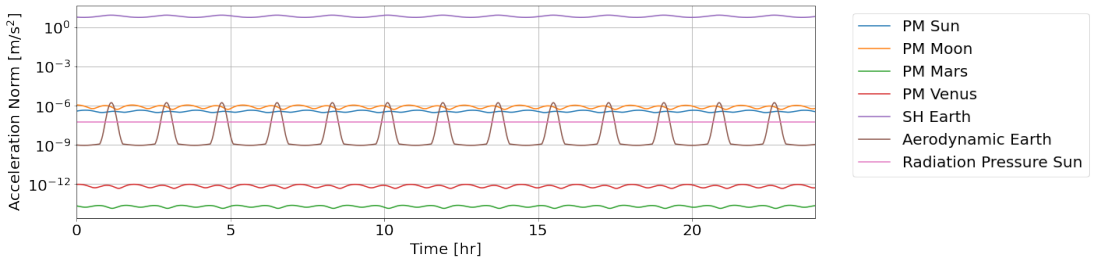
```
plt.xlabel('Longitude [deg]')
plt.ylabel( 'Latitude [deg]')
plt.xlim( [min(longitude), max(longitude)] )
plt.savefig( fname = f'{latex_image_path}ground_track.png', bbox_inches='tight')

# 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')
```


Total Acceleration [m/s^2]
8
7
6
0
5
10
15
20
Time [hr]


90
45
0
−45
−90
Latitude [deg]
−150
−100
−50
0
50
100
150
Longitude [deg]


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

[ ]: