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 **??**.

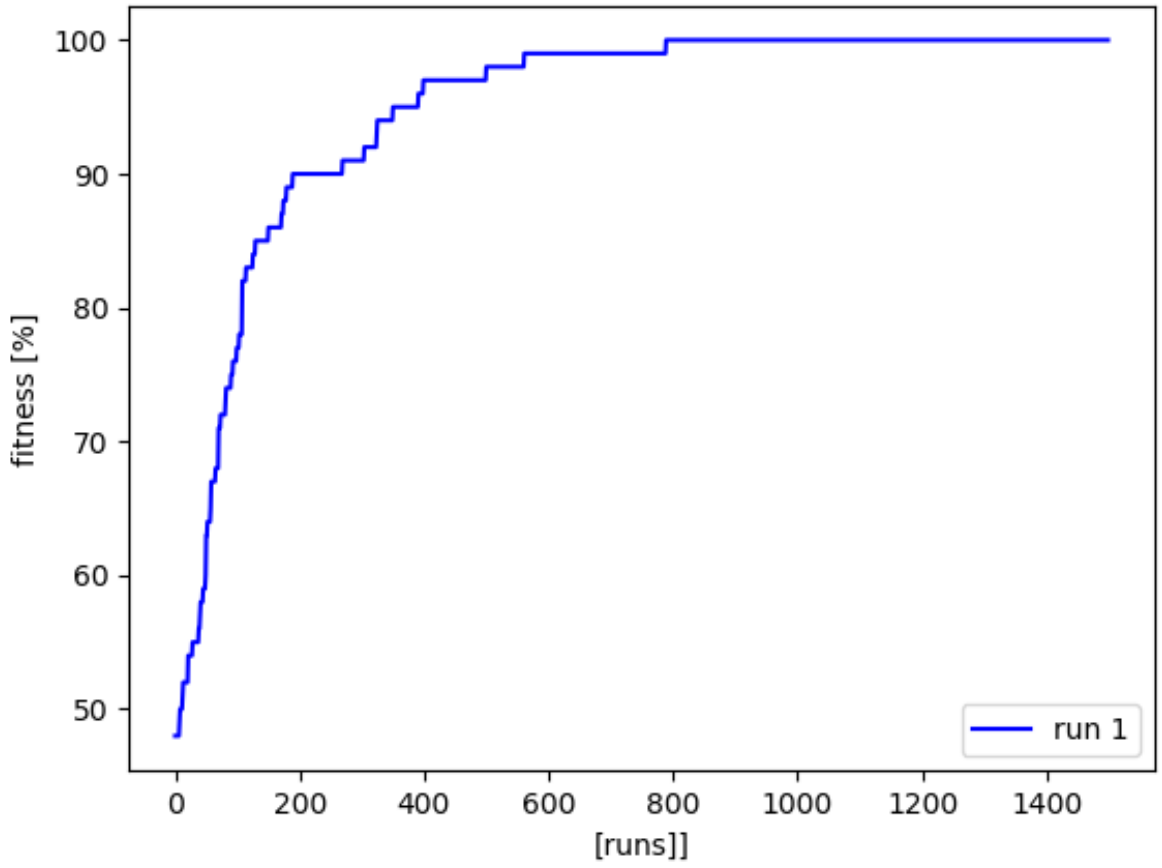


Figure 1: Performance of some genetic algorithm

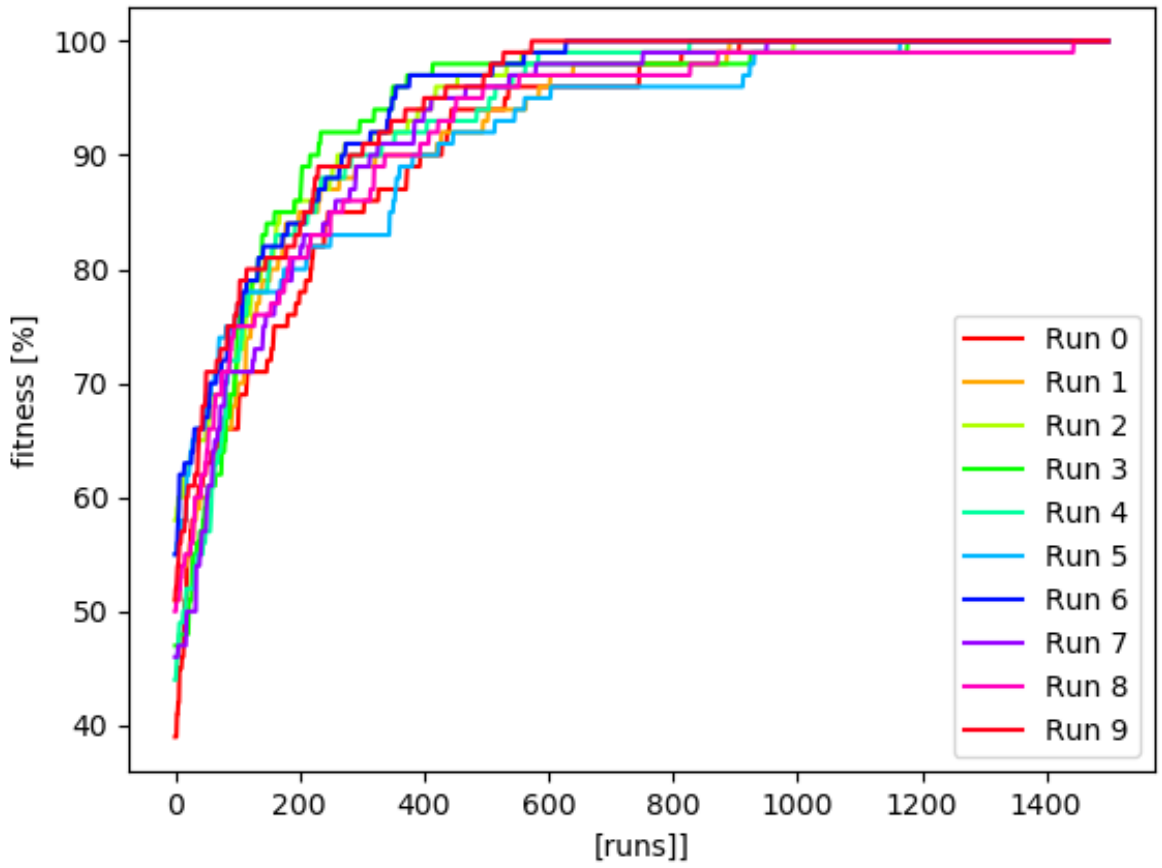


Figure 2: Performance of some genetic algorithm

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('main.tex', project_nr)

    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(main_latex_filename, project_nr):
        script_dir = get_script_dir()
        relative_dir = f'latex/project{project_nr}/'
        appendix_dir = script_dir+'/../../../'+relative_dir+'Appendices/'
        path_to_main_latex_file = f'{script_dir}/../../../{relative_dir}/{main_latex_filename}'
        root_dir = script_dir[0:script_dir.rfind(f'code/project{project_nr}')]

        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(python_filepaths, appendix_dir, '.py', project_nr, root_dir)
        notebook_pdf_files_already_included_in_appendices = get_code_files_already_included_in_appendices(compiled_notebook_pdf_filepaths, appendix_dir, '.ipynb', project_nr, root_dir)

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

        created_python_appendix_filenames = create_appendices_with_code(appendix_dir, missing_python_files_in_appendices, '.py',  project_nr,  root_dir)
        created_notebook_appendix_filenames = create_appendices_with_code(appendix_dir, missing_notebook_files_in_appendices, '.ipynb', project_nr, root_dir)

        appendices = get_list_of_appendix_files(appendix_dir, compiled_notebook_pdf_filepaths, python_filepaths)

        main_tex_code, start_index, end_index, appendix_tex_code = get_appendix_tex_code(path_to_main_latex_file)
        # assumes non-included non-code appendices should not be included:
        non_code_appendices, main_non_code_appendix_inclusion_lines =  get_order_of_non_code_appendices_in_main(appendices, appendix_tex_code)

```

```
32          python_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, filter_appendices_by_type(appendices
               ↪ , 'python')))
33          sorted_created_python_appendices = sort_python_appendices(
               ↪ filter_appendices_by_type(appendices, 'python'))
34          sorted_python_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, sorted_created_python_appendices))
35
36          notebook_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, filter_appendices_by_type(appendices
               ↪ , 'notebook')))
37          sorted_created_notebook_appendices = sort_notebook_appendices
               ↪ (filter_appendices_by_type(appendices, 'notebook'))
38          sorted_notebook_appendix_filenames = list(map(lambda x: x.
               ↪ appendix_filename, sorted_created_notebook_appendices))
39
40          appendix_latex_code = create_appendices_latex_code(
               ↪ main_non_code_appendix_inclusion_lines,
               ↪ sorted_created_notebook_appendices, project_nr,
               ↪ sorted_created_python_appendices)
41
42          updated_main_tex_code = substitute_appendix_code(
               ↪ main_tex_code, start_index, end_index,
               ↪ appendix_latex_code)
43
44          overwrite_content_to_file(updated_main_tex_code,
               ↪ path_to_main_latex_file)
45
46
47  def create_appendices_latex_code(
       ↪ main_non_code_appendix_inclusion_lines, notebook_appendices,
       ↪ project_nr, python_appendices):
48      ''' creates the appendix text for main.'''
49      main_appendix_inclusion_lines =
           ↪ main_non_code_appendix_inclusion_lines
50      for appendix in python_appendices:
51          line = update_appendix_tex_code(appendix.appendix_filename,
               ↪ project_nr)
52          main_appendix_inclusion_lines.append(line)
53
54      for appendix in notebook_appendices:
55          line = update_appendix_tex_code(appendix.appendix_filename,
               ↪ project_nr)
56          main_appendix_inclusion_lines.append(line)
57      return main_appendix_inclusion_lines
58
59
60  def filter_appendices_by_type(appendices, appendix_type):
61      ''' Returns the list of appendices of certain type from a list of
           ↪  appendix objects.'''
62      return_appendices = []
63      for appendix in appendices:
64          if appendix.appendix_type == appendix_type:
65              return_appendices.append(appendix)
66      return return_appendices
67
68
69  def sort_python_appendices(appendices):
70      ''' First puts __main__.py, followed by main.py followed by a-z
           ↪ code files.'''
71      return_appendices = []
```

```
72      for appendix in appendices: # first get appendix containing
           ↪ __main__.py
73          if (appendix.code_filename=="__main__.py") or (appendix.
               ↪ code_filename=="__Main__.py"):
74              return_appendices.append(appendix)
75              appendices.remove(appendix)
76      for appendix in appendices: # second get appendix containing main
           ↪ .py
77          if (appendix.code_filename=="main.py") or (appendix.
               ↪ code_filename=="Main.py"):
78              return_appendices.append(appendix)
79              appendices.remove(appendix)
80      return_appendices
81
82      # Filter remaining appendices in order of a-z
83      filtered_remaining_appendices = [i for i in appendices if i.
           ↪ code_filename is not None]
84      appendices_sorted_a_z = filter_list_on_property(
           ↪ filtered_remaining_appendices)
85      return return_appendices+appendices_sorted_a_z
86
87
88  def sort_notebook_appendices(appendices):
89      ''' Sorts notebooks on a-z pdf filenames.'''
90      return_appendices = []
91      filtered_remaining_appendices = [i for i in appendices if i.
           ↪ code_filename is not None]
92      appendices_sorted_a_z = filter_list_on_property(
           ↪ filtered_remaining_appendices)
93      return return_appendices+appendices_sorted_a_z
94
95
96  def filter_list_on_property(appendices):
97      ''' Returns a list based on the property: code_filename'''
98      attributes = list(map(lambda x: x.code_filename, appendices))
99      sorted_indices = sorted(range(len(attributes)), key=lambda k:
           ↪ attributes[k])
100     sorted_list = []
101     for i in sorted_indices:
102         sorted_list.append(appendices[i])
103     return sorted_list
104
105
106 def get_order_of_non_code_appendices_in_main(appendices,
       ↪ appendix_tex_code):
107     ''' Scans the lines of appendices in the main code, and returns
           ↪ the lines that
108     of appendices that do not contain code, in specified order.'''
109     non_code_appendices = []
110     non_code_appendix_lines = []
111     appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
112     for line in appendix_tex_code:
113         appendix_filename = get_filename_from_latex_appendix_line(
               ↪ appendices, line)
114
115         # Check if line is not commented
116         if not appendix_filename is None:
117             if not line_is_commented(line,appendix_filename):
118                 appendix = get_appendix_from_filename(appendices,
                       ↪ appendix_filename)
119                 if appendix.appendix_type == "no_code":
120                     non_code_appendices.append(appendix)
```

```
                    non_code_appendix_lines.append(line)
    return non_code_appendices, non_code_appendix_lines


def get_filename_from_latex_appendix_line(appendices, appendix_line):
    for filename in list(map(lambda appendix: appendix.appendix_filename, appendices)):
        if filename in appendix_line:
            return filename


def get_appendix_from_filename(appendices, appendix_filename):
    for appendix in appendices:
        if appendix_filename == appendix.appendix_filename:
            return appendix


def get_compiled_notebook_paths(script_dir):
    ''' Returns the list of jupiter notebook filepaths that were compiled successfully'''
    notebook_filepaths= get_filenames_in_dir('.ipynb', script_dir)
    compiled_notebook_filepaths = []

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

        # swap file extension
        notebook_filepath = notebook_filepath.replace('.ipynb','.pdf' )

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


def get_list_of_appendix_files(appendix_dir, absolute_notebook_filepaths, absolute_python_filepaths):
    ''' Returns a list with all the appendix files with .tex extension.'''
    appendices = []
    appendices_paths = get_filenames_in_dir('.tex', appendix_dir)

    for appendix_filepath in appendices_paths:
        appendix_type = "no_code"
        appendix_filecontent = read_file(appendix_filepath)
        line_nr_python_file_inclusion = get_line_of_latex_command(appendix_filecontent, "\pythonexternal{")
        line_nr_notebook_file_inclusion = get_line_of_latex_command(appendix_filecontent, "\includepdf[pages=")
        if  line_nr_python_file_inclusion > -1:
            appendix_type = "python"
            # get python filename
            line = appendix_filecontent[line_nr_python_file_inclusion ]
            filename = get_filename_from_latex_inclusion_command(line , '.py', "\pythonexternal{")
            appendices.append(Appendix(appendix_filepath, appendix_filecontent, appendix_type, filename, line ))
        if line_nr_notebook_file_inclusion > -1:
            appendix_type = "notebook"
```

```
172                 line = appendix_filecontent[
                        ↪ line_nr_notebook_file_inclusion]
173                 filename = get_filename_from_latex_inclusion_command(
                        ↪ line, '.pdf', "\includepdf[pages=")
174                 appendices.append(Appendix(appendix_filepath,
                        ↪ appendix_filecontent, appendix_type, filename, line
                        ↪ ))
175             else:
176                 appendices.append(Appendix(appendix_filepath,
                        ↪ appendix_filecontent, appendix_type))
177     return appendices
178
179
180 def get_filename_from_latex_inclusion_command(appendix_line,
        ↪ extension, start_substring):
181     ''' returns the filename in a latex inclusion command that is
            ↪ located in an appendix.
182     The inclusion command includes a python code or jupiter notebook
            ↪ pdf.'''
183     start_index = appendix_line.index(start_substring)
184     end_index = appendix_line.index(extension)
185     return get_filename_from_dir(appendix_line[start_index:end_index+
            ↪ len(extension)])
186
187
188 def get_filenames_in_dir(extension, path, excluded_files=None):
189     '''Returns a list of the relative paths to all files within the
            ↪ code/projectX/src/ folder that match
190     the given file extension.'''
191     filepaths=[]
192     for r, d, f in os.walk(path):
193         for file in f:
194             if file.endswith(extension):
195                 if (excluded_files is None) or ((not excluded_files
                        ↪ is None) and (not file in excluded_files)):
196                     filepaths.append(r+'/'+file)
197     return filepaths
198
199
200 def get_code_files_already_included_in_appendices(absolute_filepaths,
        ↪  appendix_dir, extension, project_nr, root_dir):
201     ''' Returns a list of filepaths that are already properly
            ↪ included in some appendix of this projectX,'''
202     appendix_files = get_filenames_in_dir('.tex', appendix_dir)
203     contained_codes = []
204     for code_filepath in absolute_filepaths:
205         for appendix_filepath in appendix_files:
206             appendix_filecontent = read_file(appendix_filepath)
207             line_nr = check_if_appendix_contains_file(
                        ↪ appendix_filecontent, code_filepath, extension,
                        ↪ project_nr, root_dir)
208             if line_nr>-1:
209                 # add filepath to list of files that are already in
                        ↪ the appendices
210                 contained_codes.append(Appendix_with_code(
                        ↪ code_filepath,
211                 appendix_filepath,
212                 appendix_filecontent,
213                 line_nr,
214                 '.py'))
215     return contained_codes
216
```

```
217
218 def check_if_appendix_contains_file(appendix_content, code_filepath,
    ↪ extension, project_nr, root_dir):
219     ''' scans an appendix content to determine whether it contains a
        ↪ substring that
220     includes the python code file.'''
221     # convert code_filepath to the inclusion format in latex format
222     latex_relative_filepath = f'latex/project{project_nr}/../../{
        ↪ code_filepath[len(root_dir):]}'
223     latex_command = get_latex_inclusion_command(extension,
        ↪ latex_relative_filepath)
224     return get_line_of_latex_command(appendix_content, latex_command)
225
226
227 def get_line_of_latex_command(appendix_content, latex_command):
228     ''' Returns the line number of a latex command if it is found.
        ↪ Returns -1 otherwise.'''
229     # check if the file is in the latex code
230     line_nr = 0
231     for line in appendix_content:
232         if latex_command in line:
233             if line_is_commented(line,latex_command):
234                 commented=True
235             else:
236                 return line_nr
237         line_nr=line_nr+1
238     return -1
239
240
241 def line_is_commented(line, target_substring):
242     ''' Returns true if a line is commented, returns false otherwise
        ↪ '''
243     left_of_command = line[:line.rfind(target_substring)]
244     if '%' in left_of_command:
245         return True
246     return False
247
248
249 def get_latex_inclusion_command(extension,
    ↪ latex_relative_filepath_to_codefile):
250     if extension==".py":
251         left = "\pythonexternal{"
252         right = "}"
253         latex_command = f'{left}{latex_relative_filepath_to_codefile
            ↪ }{right}'
254     elif extension==".ipynb":
255
256         left = "\includepdf[pages=-]{"
257         right = "}"
258         latex_command = f'{left}{latex_relative_filepath_to_codefile
            ↪ }{right}'
259     return latex_command
260
261
262 def read_file(filepath):
263     ''' Reads content of a file and returns it as a list of strings
        ↪ '''
264     with open(filepath) as f:
265         content = f.readlines()
266     return content
267
268
```

```
269 def get_code_files_not_yet_included_in_appendices(code_filepaths,
      ↪ contained_codes, extension):
270     ''' Returns a list of filepaths that are not yet properly
          ↪ included in some appendix of this projectX,'''
271     contained_filepaths = list(map(lambda contained_file:
          ↪ contained_file.code_filepath, contained_codes))
272     not_contained = []
273     for filepath in code_filepaths:
274         if not filepath in contained_filepaths:
275             not_contained.append(filepath)
276     return not_contained
277
278
279 def create_appendices_with_code(appendix_dir, code_filepaths,
      ↪ extension, project_nr, root_dir):
280     ''' Creates the latex appendix files in with relevant codes
          ↪ included.'''
281     appendix_filenames = []
282     appendix_reference_index = 0
283
284     for code_filepath in code_filepaths:
285         latex_relative_filepath = f'latex/project{project_nr}/../../{
              ↪ code_filepath[len(root_dir):]}'
286         content = []
287         filename = get_filename_from_dir(code_filepath)
288         content = create_section(appendix_reference_index, filename,
              ↪ content)
289         inclusion_command = get_latex_inclusion_command(extension,
              ↪ latex_relative_filepath)
290         content.append(inclusion_command)
291         overwrite_content_to_file(content, f'{appendix_dir}
              ↪ Auto_generated_{extension[1:]}_App{
              ↪ appendix_reference_index}.tex', False)
292         appendix_filenames.append(f'Auto_generated_{extension[1:]}
              ↪ _App{appendix_reference_index}.tex')
293         appendix_reference_index = appendix_reference_index+1
294     return appendix_filenames
295
296
297 def create_section(appendix_reference_index, code_filename, content):
298     # write section
299     left ="\section{Appendix "
300     middle = code_filename.replace("_","\_")
301     right = "}\label{app:"
302     end = "}" # TODO: update appendix reference index
303     content.append(f'{left}{middle}{right}{appendix_reference_index}{
          ↪ end}')
304     return content
305
306
307 def overwrite_content_to_file(content, filepath, content_has_newlines
      ↪ =True):
308     ''' Writes the content of an appendix to a new appendix'''
309     with open(filepath,'w') as f:
310         for line in content:
311             if content_has_newlines:
312                 f.write(line)
313             else:
314                 f.write(line+'\n')
315
316
317 def get_appendix_tex_code(main_latex_filename):
```

```
318     ''' gets the latex appendix code from the main tex file.'''
319     main_tex_code = read_file(main_latex_filename)
320     start =  "\\begin{appendices}"
321     end = "\end{appendices}"
322     start_index = get_index_of_substring_in_list(main_tex_code, start
          ↪ )+1
323     end_index = get_index_of_substring_in_list(main_tex_code, end)
324     return main_tex_code, start_index, end_index, main_tex_code[
          ↪ start_index:end_index]
325
326
327 def get_index_of_substring_in_list(lines, target_substring):
328     for i in range(0, len(lines)):
329         if target_substring in lines[i]:
330             if not line_is_commented(lines[i], target_substring):
331                 return i
332
333
334 def update_appendix_tex_code(appendix_filename, project_nr):
335     ''' Includes the appendices as latex commands in the tex code
          ↪ string'''
336     left = "\input{latex/project"
337     middle = "/Appendices/"
338     right = "} \\newpage\n"
339     return f'{left}{project_nr}{middle}{appendix_filename}{right}'
340
341
342 def substitute_appendix_code(main_tex_code, start_index, end_index,
      ↪ updated_appendices_tex_code):
343     ''' Replaces the old latex code that include the appendices with
          ↪ the new latex
344     commands that include the appendices in the latex report.'''
345     updated_main_tex_code = main_tex_code[0:start_index]+
          ↪ updated_appendices_tex_code+main_tex_code[end_index:]
346     return updated_main_tex_code
347
348
349 def get_filename_from_dir(path):
350     return path[path.rfind("/")+1:]
351
352
353 def get_script_dir():
354     ''' returns the directory of this script regardles of from which
          ↪ level the code is executed '''
355     return os.path.dirname(__file__)
356
357
358 class Appendix_with_code:
359     ''' stores in which appendix file and accompanying line number in
          ↪  the appendix in which a code file is
360     already included. Does not take into account whether this
          ↪ appendix is in the main tex file or not'''
361     def __init__(self, code_filepath, appendix_filepath,
          ↪ appendix_content, file_line_nr, extension):
362         self.code_filepath = code_filepath
363         self.appendix_filepath = appendix_filepath
364         self.appendix_content = appendix_content
365         self.file_line_nr = file_line_nr
366         self.extension = extension
367
368
369 class Appendix:
```

```
370     ''' stores in appendix files and type of appendix.'''
371     def __init__(self, appendix_filepath, appendix_content,
            ↪ appendix_type, code_filename=None, appendix_inclusion_line=
            ↪ None):
372         self.appendix_filepath = appendix_filepath
373         self.appendix_filename = get_filename_from_dir(self.
                ↪ appendix_filepath)
374         self.appendix_content = appendix_content
375         self.appendix_type = appendix_type # TODO: perform validation
                ↪  of input values
376         self.code_filename = code_filename
377         self.appendix_inclusion_line = appendix_inclusion_line
```

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
```

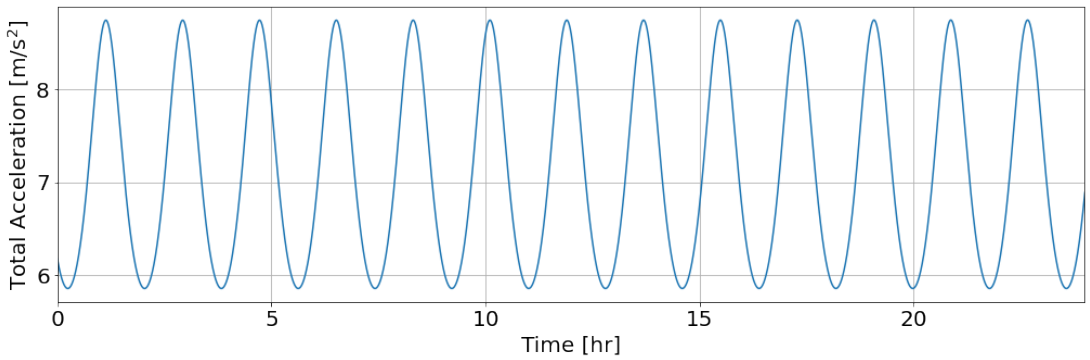
```
integrator_settings = propagation_setup.integrator.runge_kutta_4(
    simulation_start_epoch,
    fixed_step_size
)

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

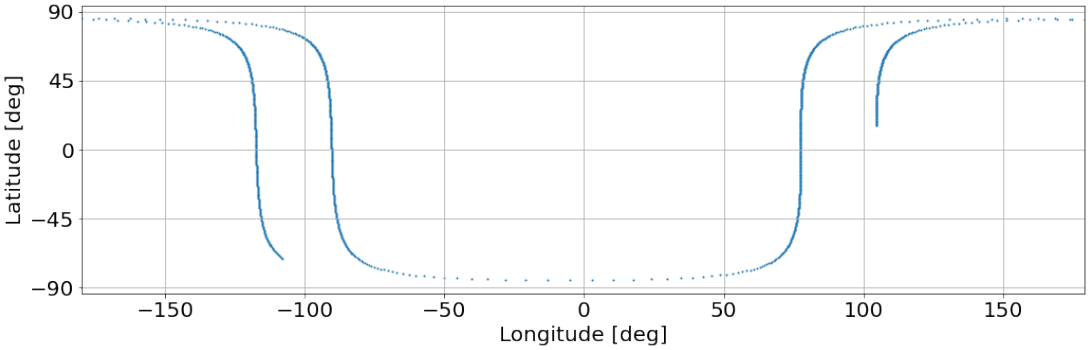
```
Single Earth-Orbiting Satellite Example.
The initial position vector of Delfi-C3 is [km]:
[7037.48400133 3238.05901792 2150.7241875 ]
The initial velocity vector of Delfi-C3 is [km/s]:
[-1.46565763 -0.04095839  6.62279761]
After 86400.0 seconds the position vector of Delfi-C3 is [km]:
[-4602.79426676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

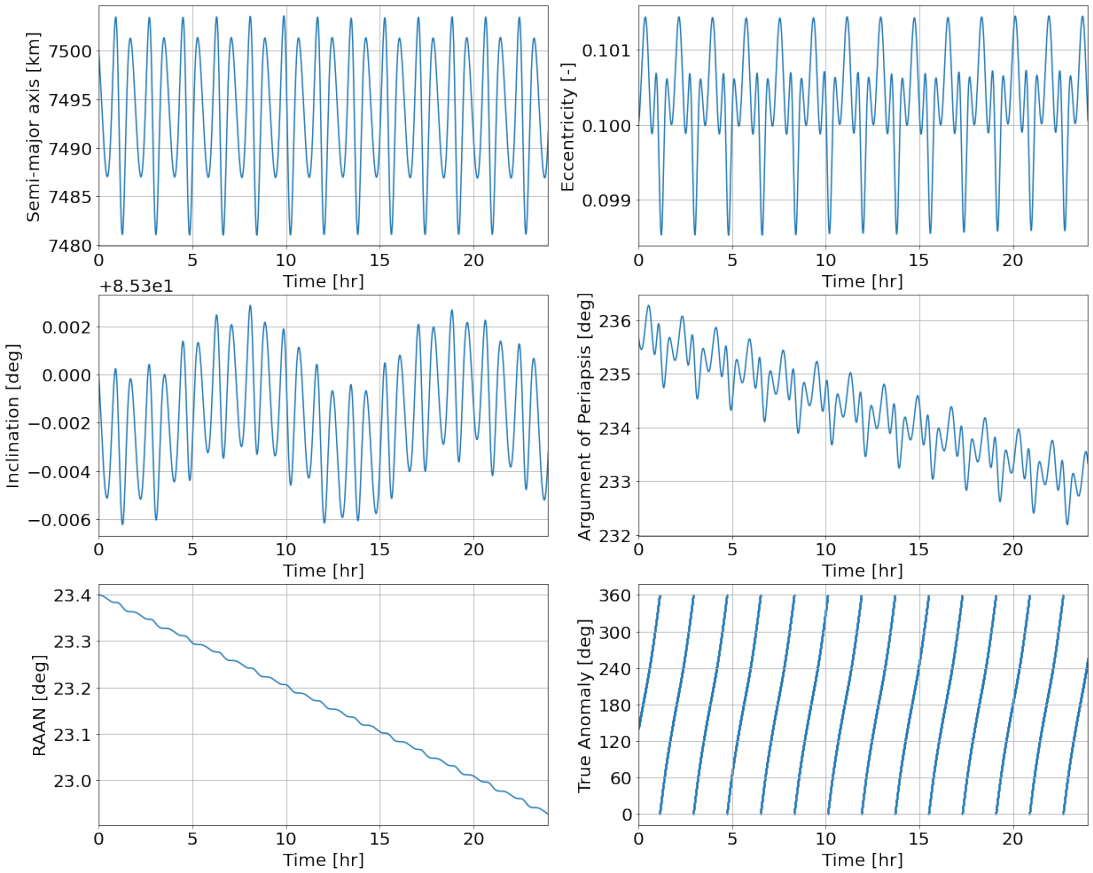
[3]:
```
import os
from matplotlib import pyplot as plt

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 )
```

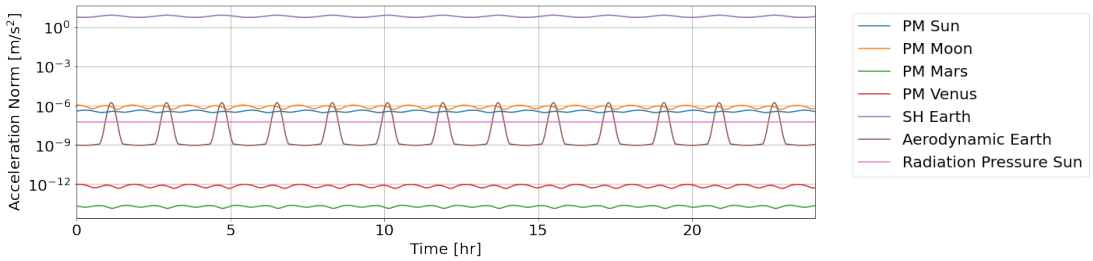
```
plt.xlabel('Longitude [deg]')
plt.ylabel( 'Latitude [deg]')
plt.xlim( [min(longitude), max(longitude)] )
plt.savefig( fname = f'{latex_image_path}ground_track.png', bbox_inches='tight')

# 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
```

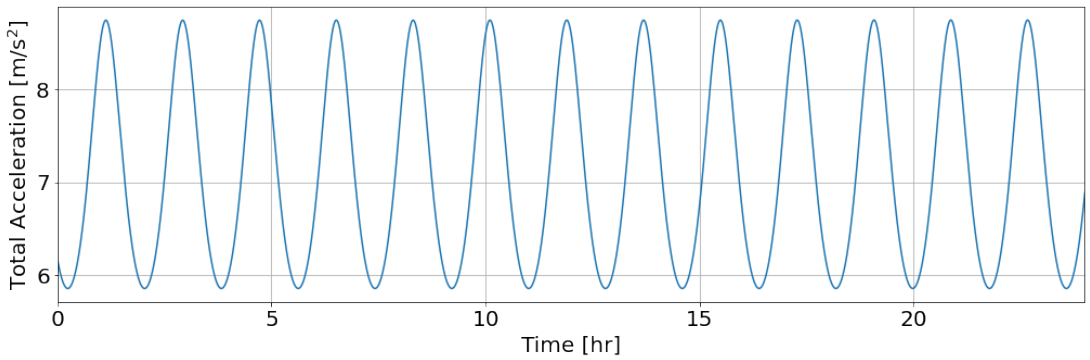
```
integrator_settings = propagation_setup.integrator.runge_kutta_4(
    simulation_start_epoch,
    fixed_step_size
)

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

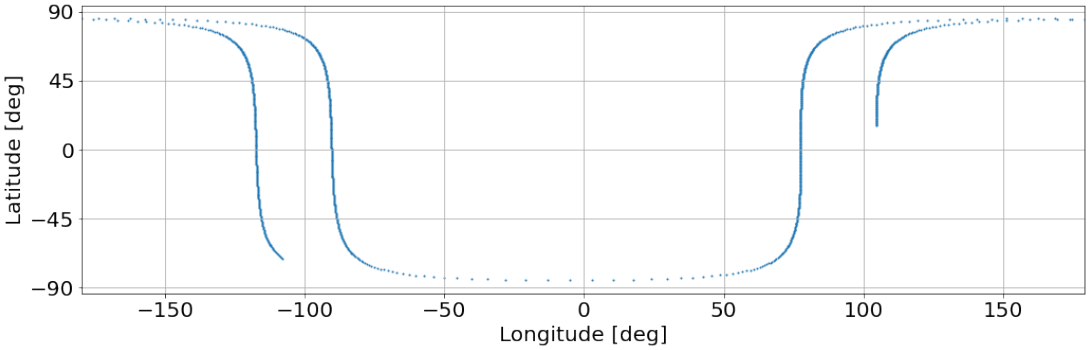
```
Single Earth-Orbiting Satellite Example.
The initial position vector of Delfi-C3 is [km]:
[7037.48400133 3238.05901792 2150.7241875 ]
The initial velocity vector of Delfi-C3 is [km/s]:
[-1.46565763 -0.04095839  6.62279761]
After 86400.0 seconds the position vector of Delfi-C3 is [km]:
[-4602.79426676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

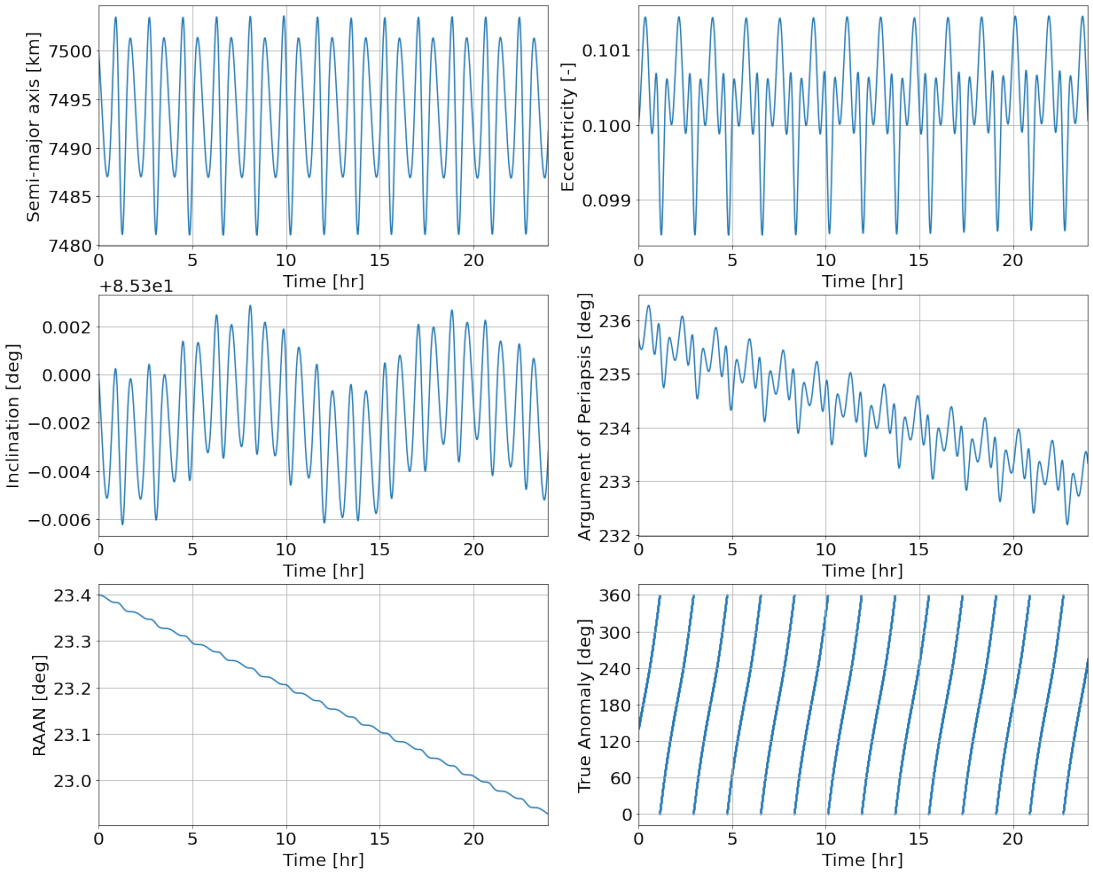
[3]:
```
import os
from matplotlib import pyplot as plt

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 )
```

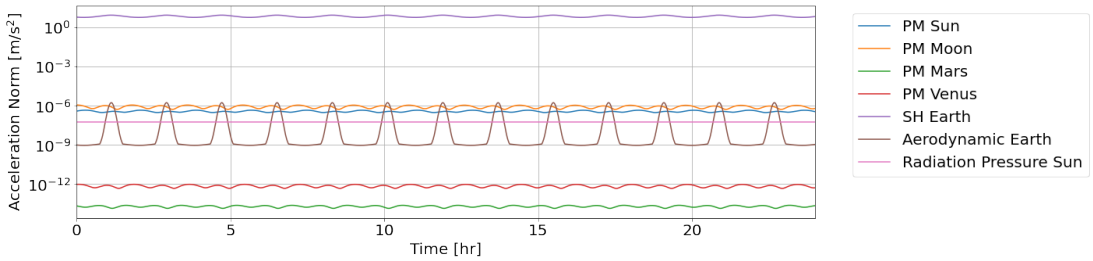
```
plt.xlabel('Longitude [deg]')
plt.ylabel( 'Latitude [deg]')
plt.xlim( [min(longitude), max(longitude)] )
plt.savefig( fname = f'{latex_image_path}ground_track.png', bbox_inches='tight')

# 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

[ ]: