

# Example to plot directly into latex

19-10-2019

## 1 Introduction

## 2 Genetic Algorithm Performance

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



Figure 1: Performance of some genetic algorithm

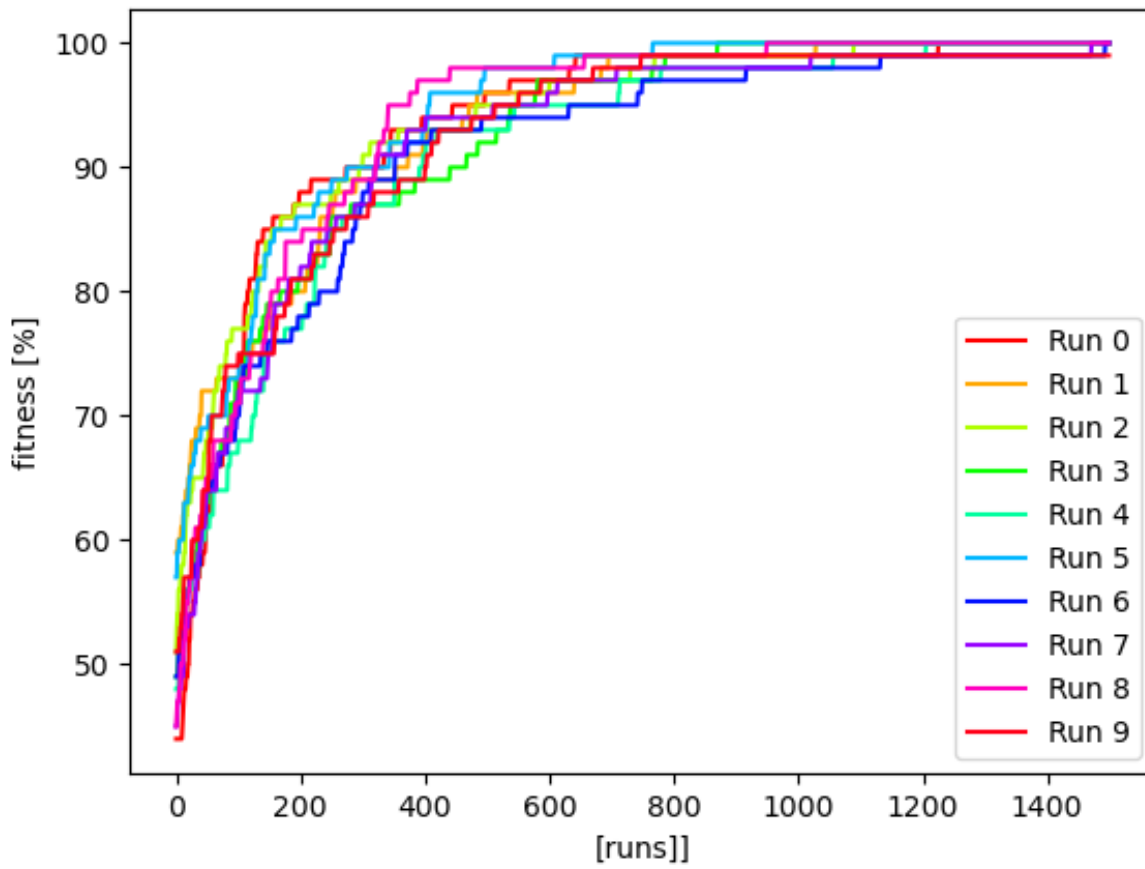


Figure 2: Performance of some genetic algorithm

## A Appendix \_\_main\_\_.py

---

```
1 import os
2 from .Main import Main
3
4 print(f'Hi, I\'ll be running the main code, and I\'ll let you know
   ↪ when I\'m done.')
5 project_nr = 2
6 main = Main()
7
8 notebook_names = ['AE4868_example_notebook_update20201025.ipynb']
9
10 # run the jupyter notebooks for assignment 1
11 main.run_jupyter_notebooks(project_nr, notebook_names)
12
13 # convert jupyter notebook for assignment 1 to pdf
14 main.convert_notebooks_to_pdf(project_nr, notebook_names)
15
16 # compile the latex report
17 main.compile_latex_report(project_nr)
18
19 #####
20 #####example code to illustrate python-latex image sync
   ↪ #####
21 #####runs arbitrary genetic algorithm, can be deleted
   ↪ #####
22 #####
23 # run a genetic algorithm to create some data for a plot.
24 print("now running a")
25 res = main.do_run_a()
26
27 # plot some graph with a single line, general form is:
28 # plt_tex.plotSingleLines(plt_tex,x,y,"x-axis label","y-axis label",
   ↪ lineLabels,"filename",legend_position,project_nr)
29 # main.plt_tex.plotSingleLine(plt_tex,range(0, len(res)),res,"[runs
   ↪ ]]", "fitness [%]", "run 1", "4a", 4, project_nr)
30
31 # run a genetic algorithm to create some data for another plot.
32 print("now running b")
33 main.do4b(project_nr)
34
35 # run a genetic algorithm to create some data for another plot.
36 print("now running 4c")
37 main.do4c(project_nr)
38
39 print(f'Done.')
```

---

## B Appendix Main.py

```
1 # Example code that creates plots directly in report
2 # Code is an implementation of a genetic algorithm
3 import random
4 from matplotlib import pyplot as plt
5 from matplotlib import lines
6 import matplotlib.pyplot as plt
7 import numpy as np
8
9 from .Compile_latex import Compile_latex
10 from .Plot_to_tex import Plot_to_tex as plt_tex
11 from .Run_jupyter_notebooks import Run_jupyter_notebook
12
13 # define global variables for genetic algorithm example
14 string_length = 100
15 mutation_chance= 1.0/string_length
16 max_iterations = 1500
17
18 class Main:
19
20     def __init__(self):
21         self.run_jupyter_notebook = Run_jupyter_notebook()
22         pass
23
24
25     def run_jupyter_notebooks(self,project_nr ,notebook_names):
26         '''runs a jupyter notebook'''
27         notebook_path = f'code/project{project_nr}/src/'
28
29         for notebook_name in notebook_names:
30             self.run_jupyter_notebook.run_notebook(f'{notebook_path}{
31                 ↪ notebook_name}')
32
33     def convert_notebooks_to_pdf(self,project_nr ,notebook_names):
34         '''converts a jupyter notebook to pdf'''
35         notebook_path = f'code/project{project_nr}/src/'
36
37         for notebook_name in notebook_names:
38             self.run_jupyter_notebook.convert_notebook_to_pdf(f'{
39                 ↪ notebook_path}{notebook_name}')
40
41     def compile_latex_report(self,project_nr):
42         '''compiles latex code to pdf'''
43         compile_latex =Compile_latex(project_nr ,'main.tex')
44
45         #####
46         #####example code to illustrate python-latex image sync
47         ↪ #####
48         #####runs arbitrary genetic algorithm, can be deleted
49         ↪ #####
50         #####
51
52     def count(self,bits):
53         count = 0
54         for bit in bits:
55             if bit:
56                 count = count + 1
57         return count
58
59     def gen_bit_sequence(self):
60         bits = []
61         for _ in range(string_length):
```

```

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

```

```

117     # plt_tex.plotMultipleLines(plt_tex,x,y,"x-axis label","y-
    ↪ axis label",lineLabels,"filename",legend_position,
    ↪ project_nr)
118     plt_tex.plotMultipleLines(plt_tex,range(0, len(res)),
    ↪ plotResult,"[runs]","fitness [%]",lineLabels,"4b",4,
    ↪ project_nr)
119     print("total optimum found: {} out of {} runs".format(
    ↪ optimum_found,10))

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

```

---

## Appendix python code that exports figures to latex

```
1  ### Call this from another file, for project 11, question 3b:
2  ### from Plot_to_tex import Plot_to_tex as plt_tex
3  ### multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints),
   ↪ dtype=int); # actually fill with data
4  ### lineLabels = [] # add a label for each dataseries
5  ### plt_tex.plotMultipleLines(plt_tex,single_x_series,
   ↪ multiple_y_series,"x-axis label [units]","y-axis label [units
   ↪ ]",lineLabels,"3b",4,11)
6  ### 4b=filename
7  ### 4 = position of legend, e.g. top right.
8  ###
9  ### For a single line, use:
10 ### plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),
   ↪ dataseries,"x-axis label [units]","y-axis label [units]",
   ↪ lineLabel,"3b",4,11)
11
12 ### You can also plot a table directly into latex, see
   ↪ example_create_a_table(..)
13 ###
14 ### Then put it in latex with for example:
15 ### \begin{table}[H]
16 ###     \centering
17 ###     \caption{Results some computation.}\label{tab:some_computation
   ↪ }
18 ###     \begin{tabular}{|c|c|} % remember to update this to show all
   ↪ columns of table
19 ###         \hline
20 ###         \input{latex/project3/tables/q2.txt}
21 ###     \end{tabular}
22 ### \end{table}
23 import random
24 from matplotlib import lines
25 import matplotlib.pyplot as plt
26 import numpy as np
27 import os
28 class Plot_to_tex:
29
30     def __init__(self):
31         self.script_dir = self.get_script_dir()
32         print("Created main")
33
34     # plot graph (legendPosition = integer 1 to 4)
35     def plotSingleLine(self,x_path,y_series,x_axis_label,y_axis_label
   ↪ ,label,filename,legendPosition,project_nr):
36         fig=plt.figure();
37         ax=fig.add_subplot(111);
38         ax.plot(x_path,y_series,c='b',ls='-',label=label,fillstyle='
   ↪ none');
39         plt.legend(loc=legendPosition);
40         plt.xlabel(x_axis_label);
41         plt.ylabel(y_axis_label);
42         plt.savefig(os.path.dirname(__file__)+ '/../../../latex/
   ↪ project'+str(project_nr)+'/Images/'+filename+'.png');
43     #
   ↪ plt.show();
44
45     # plot graphs
46     def plotMultipleLines(self,x,y_series,x_label,y_label,label,
   ↪ filename,legendPosition,project_nr):
47         fig=plt.figure();
48         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],
60                 ↪ fillstyle='none', c=cmap(i)); # color
61
62     # configure plot layout
63     plt.legend(loc=legendPosition);
64     plt.xlabel(x_label);
65     plt.ylabel(y_label);
66     plt.savefig(os.path.dirname(__file__) + '/../.../latex/
67                 ↪ project'+str(project_nr)+'/Images/'+filename+'.png');
68
69     print(f'plotted lines')
70
71     # Generate random line colours
72     # Source: https://stackoverflow.com/questions/14720331/how-to-
73     ↪ generate-random-colors-in-matplotlib
74     def get_cmap(n, name='hsv'):
75         '''Returns a function that maps each index in 0, 1, ..., n-1
76         ↪ to a distinct
77         RGB color; the keyword argument name must be a standard mpl
78         ↪ colormap name.'''
79         return plt.cm.get_cmap(name, n)
80
81     def generateLineTypes(y_series):
82         # generate varying linetypes
83         typeOfLines = list(lines.lineStyles.keys())
84
85         while(len(y_series)>len(typeOfLines)):
86             typeOfLines.append("-.");
87
88         # remove void lines
89         for i in range(0, len(y_series)):
90             if (typeOfLines[i]=='None'):
91                 typeOfLines[i]='-'
92             if (typeOfLines[i]=='):'):
93                 typeOfLines[i]=':'
94             if (typeOfLines[i]==' '):
95                 typeOfLines[i]='--'
96         return typeOfLines
97
98     # Create a table with: table_matrix = np.zeros((4,4), dtype=object
99     ↪ ) and pass it to this object
100     def put_table_in_tex(self, table_matrix, filename, project_nr):
101         cols = np.shape(table_matrix)[1]
102         format = "%s"
103         for col in range(1, cols):
104             format = format + " & %s"
105         format = format + ""
106         plt.savetxt(os.path.dirname(__file__) + '/../.../latex/
107                     ↪ project"+str(project_nr)+"/tables/"+filename+".txt",
108                     ↪ table_matrix, delimiter=' & ', fmt=format, newline='
109                     ↪ \\ \\ \\ \\ \\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 empty
    ↪ 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 regardless 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()

```

---

## Appendix python code that compiles the latex report to pdf

---

```
1 # runs a jupyter notebook and converts it to pdf
2
3 import os
4 import shutil
5 import nbformat
6 from nbconvert.preprocessors import ExecutePreprocessor
7
8 class Compile_latex:
9
10     def __init__(self, project_nr, latex_filename):
11         self.script_dir = self.get_script_dir()
12         relative_dir = f'latex/project{project_nr}/'
13         self.compile_latex(relative_dir, latex_filename)
14         self.clean_up_after_compilation(latex_filename)
15         self.move_pdf_into_latex_dir(relative_dir, latex_filename)
16
17     # runs jupyter notebook
18     def compile_latex(self, relative_dir, latex_filename):
19         os.system(f'pdflatex {relative_dir}{latex_filename}')
20
21     def clean_up_after_compilation(self, latex_filename):
22         latex_filename_without_extention = latex_filename[:-4]
23         print(f'latex_filename_without_extention={
24             ↪ latex_filename_without_extention}')
25         self.delete_file_if_exists(f'{
26             ↪ latex_filename_without_extention}.aux')
27         self.delete_file_if_exists(f'{
28             ↪ latex_filename_without_extention}.log')
29         self.delete_file_if_exists(f'texput.log')
30
31     def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
32         pdf_filename = f'{latex_filename[:-4]}.pdf'
33         destination= f'{self.get_script_dir()}/../../{relative_dir
34             ↪ }{pdf_filename}'
35
36         try:
37             shutil.move(pdf_filename, destination)
38         except:
39             print("Error while moving file ", pdf_filename)
40
41     def delete_file_if_exists(self, filename):
42         try:
43             os.remove(filename)
44         except:
45             print(f'Error while deleting file: {filename} but that is
46                 ↪ not too bad because the intention is for it to not
47                 ↪ be there.')
48
49     def get_script_dir(self):
50         ''' returns the directory of this script regardless of from
51             ↪ which level the code is executed '''
52         return os.path.dirname(__file__)
53
54 if __name__ == '__main__':
55     main = Compile_latex()
```

---

## Appendix python code that runs the jupyter notebook(s)

---

```
1 # runs a jupyter notebook and converts it to pdf
2
3 import os
4 import nbformat
5 from nbconvert.preprocessors import ExecutePreprocessor
6
7 class Run_jupyter_notebook:
8
9     def __init__(self):
10         self.script_dir = self.get_script_dir()
11         print("Created main")
12
13     # runs jupyter notebook
14     def run_notebook(self, notebook_filename):
15
16         # Load your notebook
17         with open(notebook_filename) as f:
18             nb = nbformat.read(f, as_version=4)
19
20         # Configure
21         ep = ExecutePreprocessor(timeout=600, kernel_name='python3')
22
23         # Execute
24         ep.preprocess(nb, {'metadata': {'path': f'{self.
25             ↪ get_script_dir()}/.././.././'}})
26
27         # Save output notebook
28         with open(notebook_filename, 'w', encoding='utf-8') as f:
29             nbformat.write(nb, f)
30
31     # converts jupyter notebook to pdf
32     def convert_notebook_to_pdf(self, notebook_filename):
33         os.system(f'jupyter nbconvert --to pdf {notebook_filename}')
34
35     def get_script_dir(self):
36         ''' returns the directory of this script regardless of from
37             ↪ which level the code is executed '''
38         return os.path.dirname(__file__)
39
40 if __name__ == '__main__':
41     main = Run_jupyter_notebook()
```

---

## Appendix Example Jupyter Notebook

# AE4868\_example\_notebook\_update20201025

November 9, 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 2  
      latex_image_path = 'latex/project2/Images/'  
  
      # Load spice kernels.  
      spice_interface.load_standard_kernels()  
  
      # Set simulation start and end epochs.  
      simulation_start_epoch = 0.0  
      simulation_end_epoch = constants.JULIAN_DAY  
  
      #####  
      # CREATE ENVIRONMENT #####  
      #####  
  
      # Create default body settings for selected celestial bodies  
      bodies_to_create = ["Sun", "Earth", "Moon", "Mars", "Venus"]  
  
      # Create default body settings for bodies_to_create, with "Earth"/"J2000" as  
      # global frame origin and orientation. This environment will only be valid  
      # in the indicated time range  
      # [simulation_start_epoch --- simulation_end_epoch]  
      body_settings = environment_setup.get_default_body_settings(
```

```

bodies_to_create,
simulation_start_epoch,
simulation_end_epoch,
"Earth", "J2000")

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

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

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

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

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

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

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

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

# Define accelerations acting on Delfi-C3 by Sun and Earth.

```

```

accelerations_settings_delfi_c3 = dict(
    Sun=
    [
        propagation_setup.acceleration.cannonball_radiation_pressure(),
        propagation_setup.acceleration.point_mass_gravity()
    ],
    Earth=
    [
        propagation_setup.acceleration.spherical_harmonic_gravity(5, 5),
        propagation_setup.acceleration.aerodynamic()
    ])

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

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

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

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

# Set initial conditions for the Asterix satellite that will be
# propagated in this simulation. The initial conditions are given in
# Keplerian elements and later on converted to Cartesian elements.
earth_gravitational_parameter = bodies.get_body( "Earth" ).
↳gravitational_parameter
initial_state = conversion.keplerian_to_cartesian(
    gravitational_parameter=earth_gravitational_parameter,
    semi_major_axis=7500.0E3,
    eccentricity=0.1,
    inclination=np.deg2rad(85.3),
    argument_of_periapsis=np.deg2rad(235.7),
    longitude_of_ascending_node=np.deg2rad(23.4),
    true_anomaly=np.deg2rad(139.87)
)

# Define list of dependent variables to save.

```

```

dependent_variables_to_save = [
    propagation_setup.dependent_variable.total_acceleration( "Delfi-C3" ),
    propagation_setup.dependent_variable.keplerian_state( "Delfi-C3", "Earth" ),
    propagation_setup.dependent_variable.latitude( "Delfi-C3", "Earth" ),
    propagation_setup.dependent_variable.longitude( "Delfi-C3", "Earth"),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",
↪ "Sun"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",
↪ "Moon"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",
↪ "Mars"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.point_mass_gravity_type, "Delfi-C3",
↪ "Venus"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.spherical_harmonic_gravity_type,
↪ "Delfi-C3", "Earth"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.aerodynamic_type, "Delfi-C3", "Earth"
    ),
    propagation_setup.dependent_variable.single_acceleration_norm(
        propagation_setup.acceleration.cannonball_radiation_pressure_type,
↪ "Delfi-C3", "Sun"
    )
]

# Create propagation settings.
propagator_settings = propagation_setup.propagator.translational(
    central_bodies,
    acceleration_models,
    bodies_to_propagate,
    initial_state,
    simulation_end_epoch,
    output_variables = dependent_variables_to_save
)

# Create numerical integrator settings.
fixed_step_size = 10.0

```



```

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

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

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

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

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

```

```

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

```

```

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

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

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

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

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

time_hours = [ t / 3600 for t in time]
# Total Acceleration
plt.figure( figsize=(17,5))
plt.grid()
plt.plot( time_hours , total_acceleration )
plt.xlabel('Time [hr]')
plt.ylabel( 'Total Acceleration [m/s2]' )
plt.xlim( [min(time_hours), max(time_hours)] )
plt.savefig( fname = f'{latex_image_path}total_acceleration.png',
    ↳ bbox_inches='tight')

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

part = int(len(time)/24*3)
latitude = np.rad2deg( latitude[0:part] )
longitude = np.rad2deg( longitude[0:part] )
plt.figure( figsize=(17,5))
plt.grid()
plt.yticks(np.arange(-90, 91, step=45))
plt.scatter( longitude, latitude, s=1 )

```

```

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

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

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

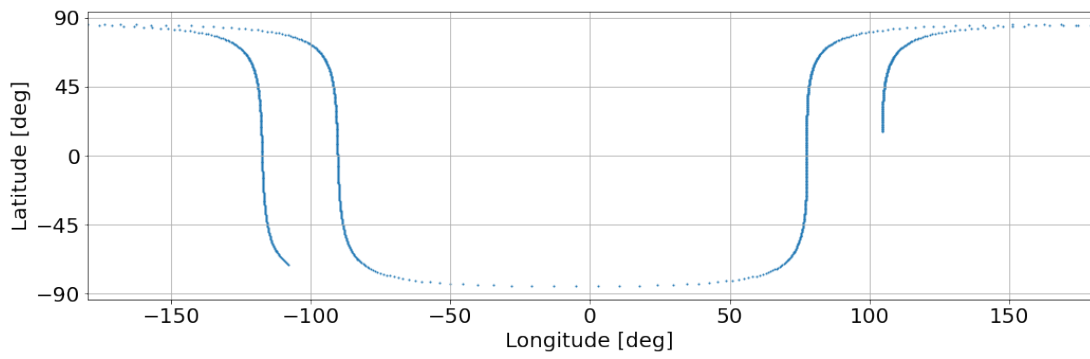
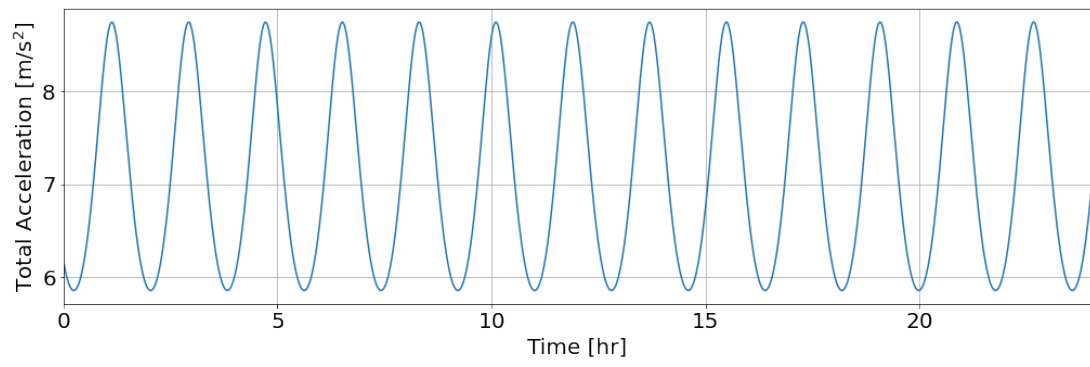
# Aerodynamic Acceleration Earth
acceleration_norm_aero_earth = dependent_variable_list[:, 16]
plt.plot( time_hours, acceleration_norm_aero_earth, label='Aerodynamic Earth')

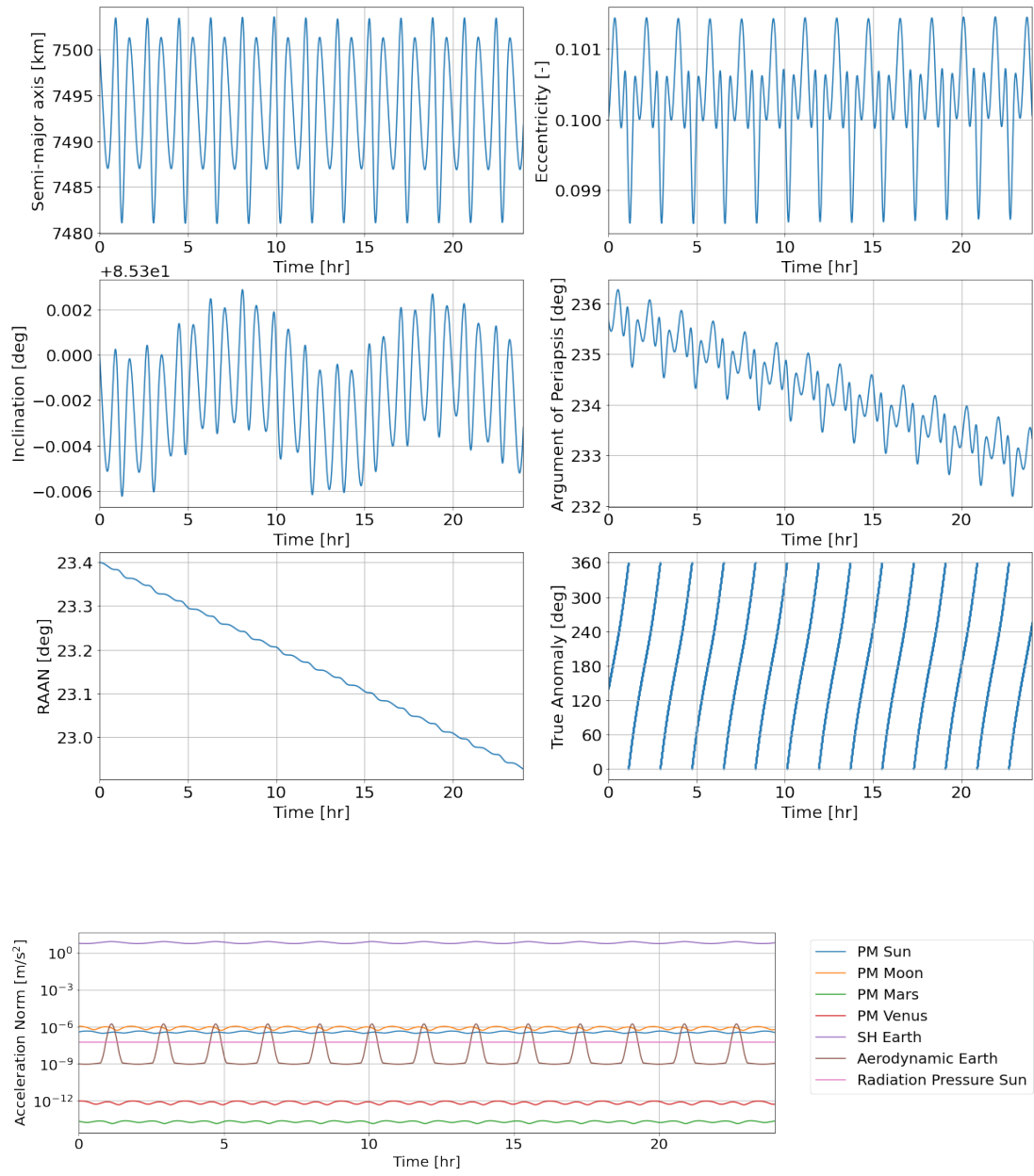
# Cannonball Radiation Pressure Acceleration Sun
acceleration_norm_rp_sun = dependent_variable_list[:, 17]
plt.plot( time_hours, acceleration_norm_rp_sun, label='Radiation Pressure Sun')

plt.grid()
plt.legend( bbox_to_anchor=(1.04,1) )
plt.xlim( [min(time_hours), max(time_hours)] )
plt.yscale('log')
plt.xlabel( 'Time [hr]' )
plt.ylabel( 'Acceleration Norm [m/s2]' )

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

```





[ ]: