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 **??**.

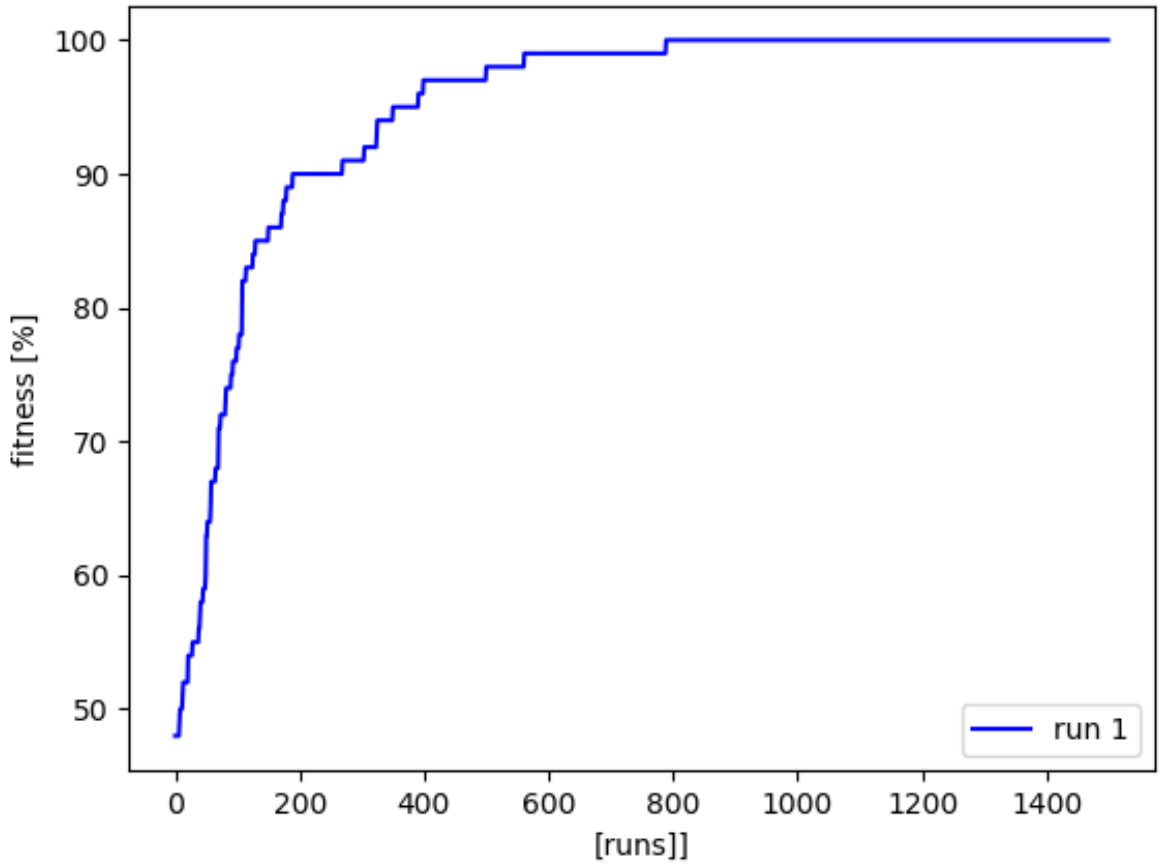


Figure 1: Performance of some genetic algorithm

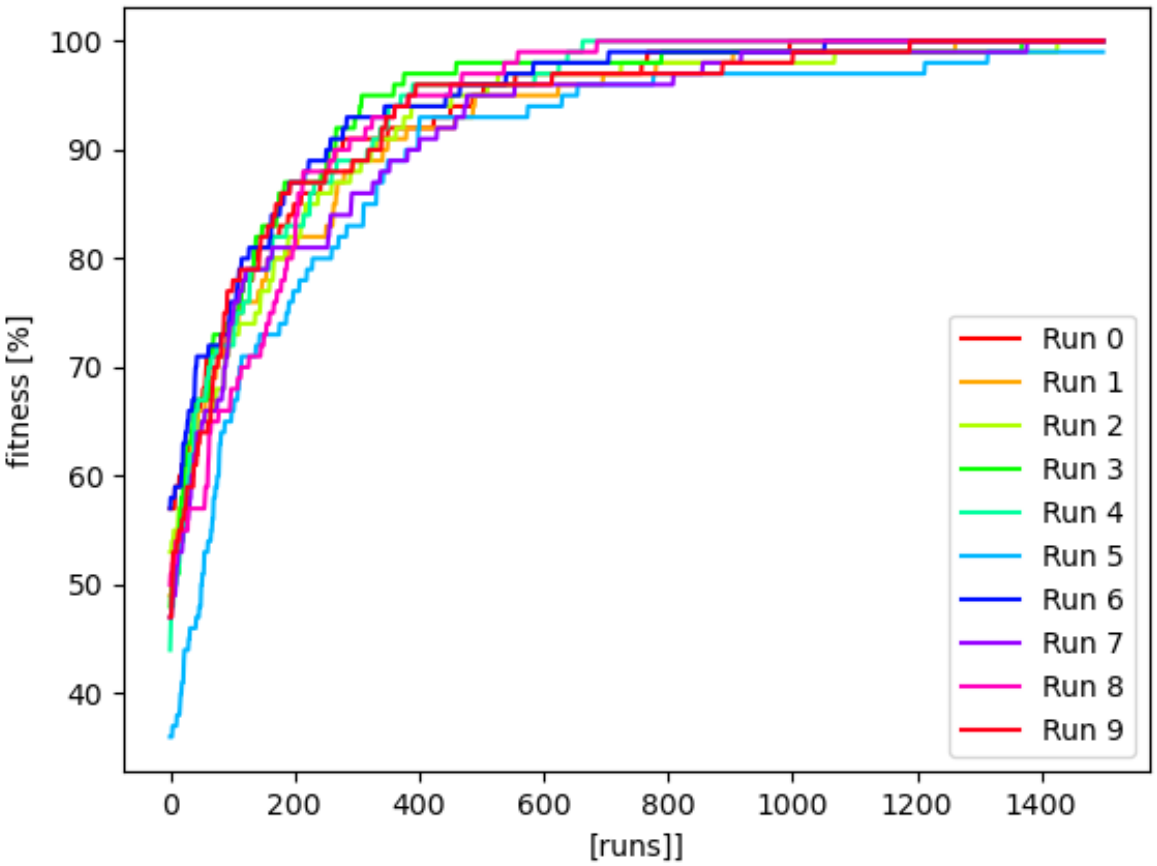


Figure 2: Performance of some genetic algorithm

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 = 3
main = Main()

notebook_names = ['AE4868_example_notebook_update20201025.ipynb']

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

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

# compile the latex report
main.compile_latex_report(project_nr)

##################################################################
############example code to illustrate python-latex  image sync #########
###############runs arbitrary genetic algorithm, can be deleted ############
##################################################################
# run a genetic algorithm to create some data for a plot.
print("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

# 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'''
        print(f'Running AE4868_example_notebook_update20201025.ipynb' )
        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 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

    def gen_bit_sequence(self):
```

```
        bits = []
        for _ in range(string_length):
            bits.append(True if random.randint(0, 1) == 1 else False)
        return bits

    def mutate_bit_sequence(self,sequence):
        retval = []
        for bit in sequence :
            do_mutation = random.random() <= mutation_chance
            if(do_mutation):
                retval.append(not bit)
            else:
                retval.append(bit)
        return retval

    #execute a run a
    def do_run_a(self):

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


    #execute a run c
    def do_run_c(self):
        seq = self.gen_bit_sequence()
        fitness = self.count(seq)
        results = [fitness]
        for run in range(max_iterations):
            new_seq = self.mutate_bit_sequence(seq)
            new_fitness = self.count(new_seq)
            seq = new_seq
            fitness = new_fitness
            results.append(max(results[-1], fitness))
        return results

    def do4b(self,project_nr):
        optimum_found = 0

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

        # perform computation
        for run in range(10):
            res = self.do_run_a()
            if res[-1] == string_length:
                optimum_found +=1

            # store computation data for plotting
            lineLabels.append(f'Run {run}')
            plotResult[run,:]=res;

```

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

Appendix python code that exports figures to latex

```
### 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()
```

Appendix python code that compiles the latex report to pdf

```python
# 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()
```

Appendix python code that runs the jupyter notebook(s)

```
# 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': 'notebooks/'}})
        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

November 9, 2020

[1]:
```
import os
#import ipyparams
#currentNotebook = ipyparams.notebook_name
#print(f'currentNotebook={currentNotebook}')
#currentNotebook = os.path.dirname(os.path.realpath(__file__))
#currentNotebook = os.path
#currentNotebook = os.getcwd()
#if not 'workbookDir' in globals():
#    workbookDir = os.getcwd()
#print('workbookDir: ' + workbookDir)
#currentNotebook = os.chdir(workbookDir)  # If you changed the current working␣
 ↪dir, this will take you back to the workbook dir.
#currentNotebook = %pwd
```

[ ]:
```
```

[2]:
```
################################################################################
# IMPORT STATEMENTS ############################################################
################################################################################
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/project3/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.
```

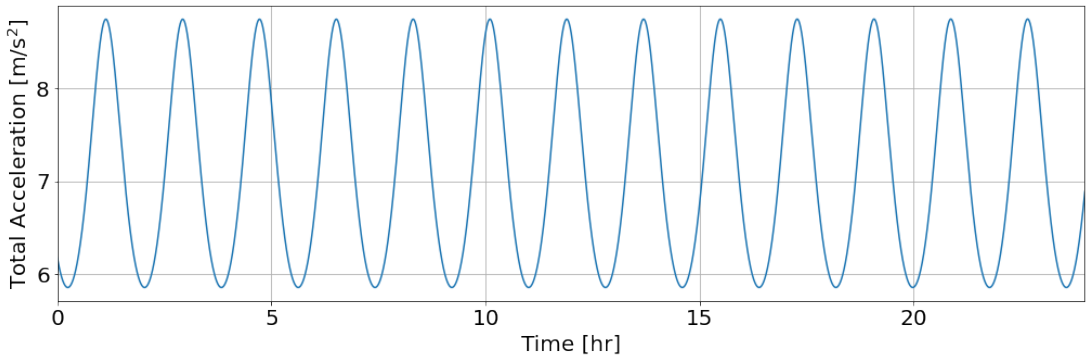
```
The initial position vector of Delfi-C3 is [km]:
[7037.48400133 3238.05901792 2150.7241875 ]
The initial velocity vector of Delfi-C3 is [km/s]:
[-1.46565763 -0.04095839  6.62279761]
After 86400.0 seconds the position vector of Delfi-C3 is [km]:
[-4602.79426676 -1421.16740978  5883.69740624]
And the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

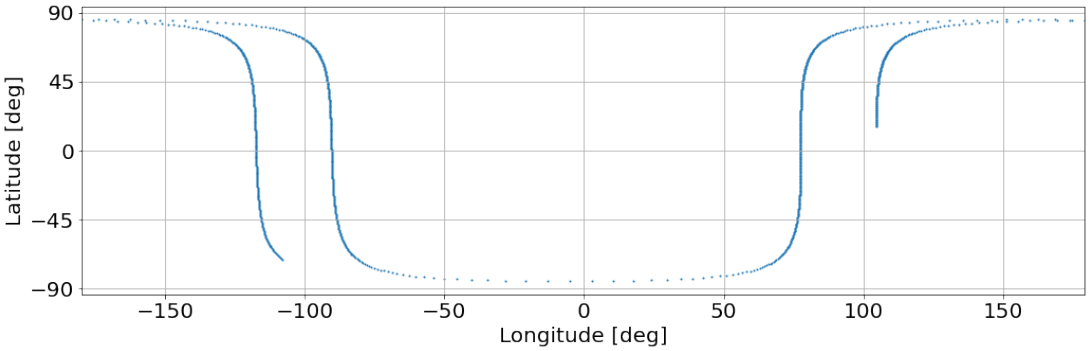
[3]:
```python
import os
from matplotlib import pyplot as plt

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

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

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

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

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



# Ground Track
```

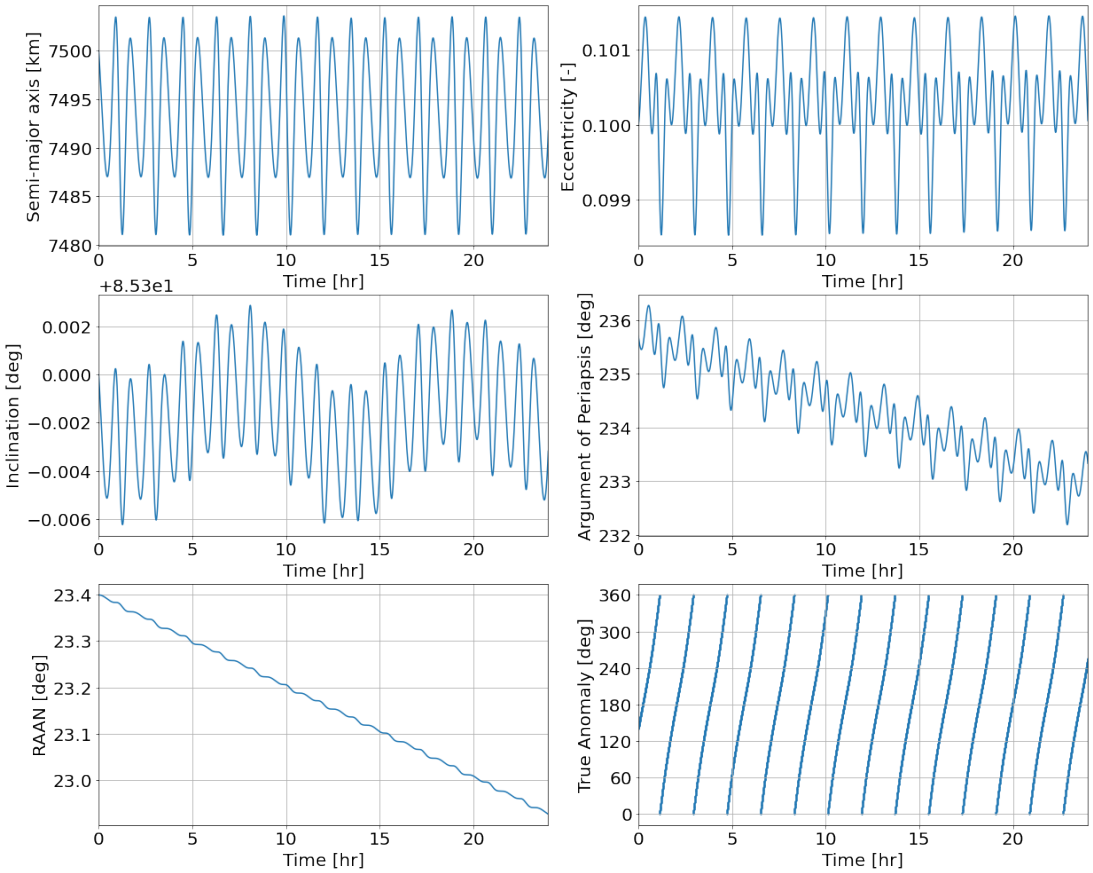
```
latitude = dependent_variable_list[:,9]
longitude = dependent_variable_list[:,10]

part = int(len(time)/24*3)
latitude = np.rad2deg( latitude[0:part] )
longitude = np.rad2deg( longitude[0:part] )
plt.figure( figsize=(17,5))
plt.grid()
plt.yticks(np.arange(-90, 91, step=45))
plt.scatter( longitude, latitude, s=1 )
plt.xlabel('Longitude [deg]')
plt.ylabel( 'Latitude [deg]')
plt.xlim( [min(longitude), max(longitude)] )
plt.savefig( fname = f'{latex_image_path}ground_track.png', bbox_inches='tight')

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

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

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

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

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

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

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

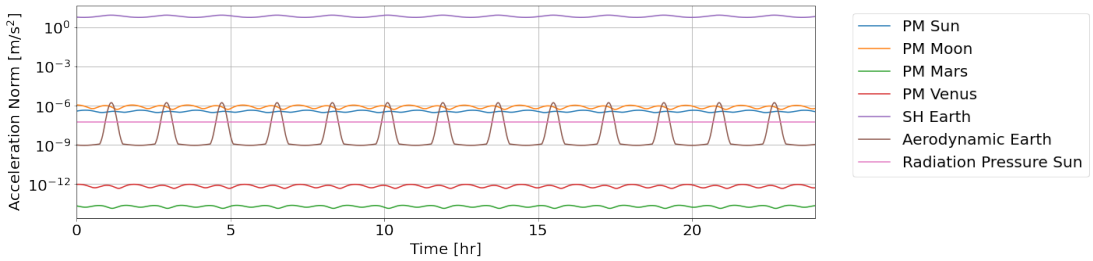
```
# True Anomaly
true_anomaly = [ np.rad2deg( element ) for element in kepler_elements[:,5] ]
ax6.scatter( time_hours, true_anomaly, s=1 )
ax6.set_ylabel( 'True Anomaly [deg]' )
ax6.set_yticks(np.arange(0, 361, step=60))

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

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

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

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

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

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

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

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

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

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

plt.grid()
plt.legend( bbox_to_anchor=(1.04,1) )
plt.xlim( [min(time_hours), max(time_hours)])
```

```
plt.yscale('log')
plt.xlabel( 'Time [hr]' )
plt.ylabel( 'Acceleration Norm [m/s$^2$]' )

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






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


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

[ ]: