LongitudinalHandsOnTracking_solutions

September 27, 2022

1 Longitudinal tracking simulations

S. Albright, H. Damerau, A. Lasheen, F. Tecker

1.1 Links

- Introductory CAS wesite: https://indico.cern.ch/event/1117526/
- Programme of the CAS: https://cas.web.cern.ch/sites/default/files/Timetable_Introductory2022_ver7.pdf
- Python software installation for transverse (and longitudinal) exercises: https://github.com/cerncas/hands-on-lattice-exercises/blob/master/Setup_Instructions.md
- Longitudinal hands-on, link to content and cheat sheets: https://indico.cern.ch/event/1117526/contributions/4978478/

1.2 Introduction

In this hands-on session we will experiment with particle tracking simulations.

The goal of the session is to write a tracking code to observe the evolution of the particles in the longitudinal phase space (ϕ, ΔE), for each turn in the machine.

The notebook is constructed with the following purpose in mind:

1. Compute basic machine parameters (the example of the CERN scSPS is taken).
2. Writing the equations of motion in the form of a python function, to track a single particle.
3. Record and observe the trajectory of a particle in the longitudinal phase space.
4. Extend the tracking code to work with many particles, acceleration, below and above transition.
5. Analyze the particle motion and compare with analytical evaluations of the bucket area, height, synchrotron frequency.
6. Observe the evolution of a bunch of particles, and simulate the injection of a bunch in a synchrotron by adjusting the RF parameters to match the bunch to the RF bucket.
7. BONUS: include more complex features like multiple RF systems, synchrotron radiation, in the tracking loop.

Along the exercises, you will be encouraged to use `support_functions`. These were designed to help you during the hands-on session by reducing the coding overhead. You can check the documentation of each function by calling `function?` in a new cell.

The available support functions are

support_functions.py
plot_phase_space_trajectory
plot_phase_space_distribution
synchrotron_tune
separatrix
run_animation
oscillation_spectrum
synchrotron_tune

1.3 Importing modules

[ ]:
```
# In this cell we import all modules that will be required for the computation
# You can add extra imports as you progress in the exercises
# Hint: use scipy.constants for elementary charge, speed of light, mass of a␣
↪proton...

import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import e, c, m_p
```

1.4 Basic accelerator and beam parameters

1.4.1 Parameters of the Super Proton Synchrotron (SPS) at CERN

Parameter	
Energy range	$E_{\text{kin}} = 26\,\text{GeV}...1300\,\text{GeV}$
Circumference	$2\pi R = 6911.5\,\text{m}$
Bending radius	$\rho = 741.3\,\text{m}$
Transition gamma	$\gamma_{\text{tr}} = 18.$
Acceleration time	$4\,\text{s}$
Harmonic number	4620
Maximum RF voltage	$V_{\text{rf}} = 15\,\text{MV}$
Longitudinal emittance per bunch	$\varepsilon_{\text{l}} = 0.6\,\text{eVs}$
Maximum bucket filling factor	$\varepsilon_{\text{l}}/A_{\text{bucket}} = 0.8$
Total beam intensity	$N = 1.6 \cdot 10^{14}$ protons ($2 \times 320\text{b} \times 2.5 \cdot 10^{11}$protons/bunch)

1.5 Exercise 1: Compute basic machine parameters

1. Compute the following parameters at the minimum/maximum energies
   - E, p
   - β, γ, T_{rev}, f_{rev}

- f_{rf}, T_{rf}
- α_c, η

2. Some reflexion and crosscheck with respect to yesterday's hands-on exercises
   - How large is the RF frequency sweep ?
   - What is the bucket length ?
   - Are we above/below transition ?

[ ]:
```
Ekin = 26e9  # 1.3e12

charge = 1
E0 = m_p*c**2./e
circumference = 6911.5
energy = Ekin + E0
momentum = np.sqrt(energy**2. - E0**2.)
beta = momentum/energy
gamma = energy/E0

t_rev = circumference/(beta*c)
f_rev = 1/t_rev

harmonic = 4620
voltage = 4.5e6
f_rf = harmonic*f_rev
t_rf = 1/f_rf

gamma_t = 18
alpha_c = 1/gamma_t**2.
eta = alpha_c - 1/gamma**2.

print("Beta: " +str(beta))
print("Gamma: " +str(gamma))
print("Revolution period: " +str(t_rev*1e6) + " mus")
print("RF frequency: " +str(f_rf/1e6) + " MHz")
print("RF period: " +str(t_rf*1e9) + " ns")
print("Momentum compaction factor: " +str(alpha_c))
print("Phase slippage factor: " +str(eta))
```

```
Beta: 0.9993932358608412
Gamma: 28.710512044511262
Revolution period: 23.0682794443392 mus
RF frequency: 200.27501449110963 MHz
RF period: 4.993134078861298 ns
Momentum compaction factor: 0.0030864197530864196
Phase slippage factor: 0.0018732596374893458
```

1.6 Exercise 2: Tracking with a single particle

1. Write functions to track the particle coordinates following longitudinal equations of motion

$$\phi_{n+1} = \phi_n + 2\pi h \eta \frac{\Delta E_n}{\beta^2 E}$$

$$\Delta E_{n+1} = \Delta E_n + qV \sin(\phi_{n+1}) - U_0$$

   - *Start with no acceleration or synchrotron radiation*
2. Define the initial coordinates of a particle in the $(\phi, \Delta E)$ phase space.
3. Simulate the evolution of the particle coordintates for few hundred turns and store the coordinates.
   - *You can pre-allocate a numpy array with* ***`np.zeros(n_turns)`*** *and fill the particle coordinates each turn*
   - *You can store the particle coordinates at each turn by appending the coordinates to a list*
4. Plot the evolution of the particle phase and energy vs. turn number, and the particle motion in longitudinal phase space
   - *The support function* ***`plot_phase_space_trajectory`*** *can be used*

[ ]:
```
# Tracking functions

def drift(phaseInitial, energyInitial, harmonic, eta, beta, energy):
    newPhase = phaseInitial + 2*np.pi*harmonic*eta*energyInitial/
 ↪(beta**2*energy)
    return newPhase

def kick(energyInitial, phaseInitial, charge, voltage):
    newEnergy = energyInitial + charge*voltage*np.sin(phaseInitial)
    return newEnergy
```

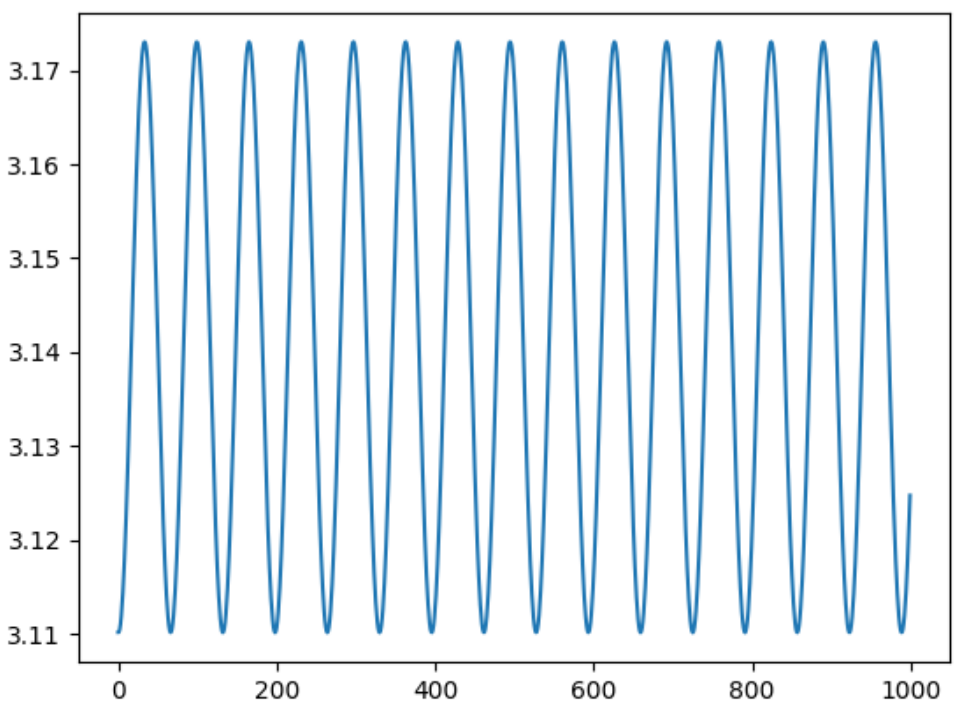
[ ]:
```
# Initial coordinates

particlePhase = 0.99*np.pi
particleEnergy = 0
```

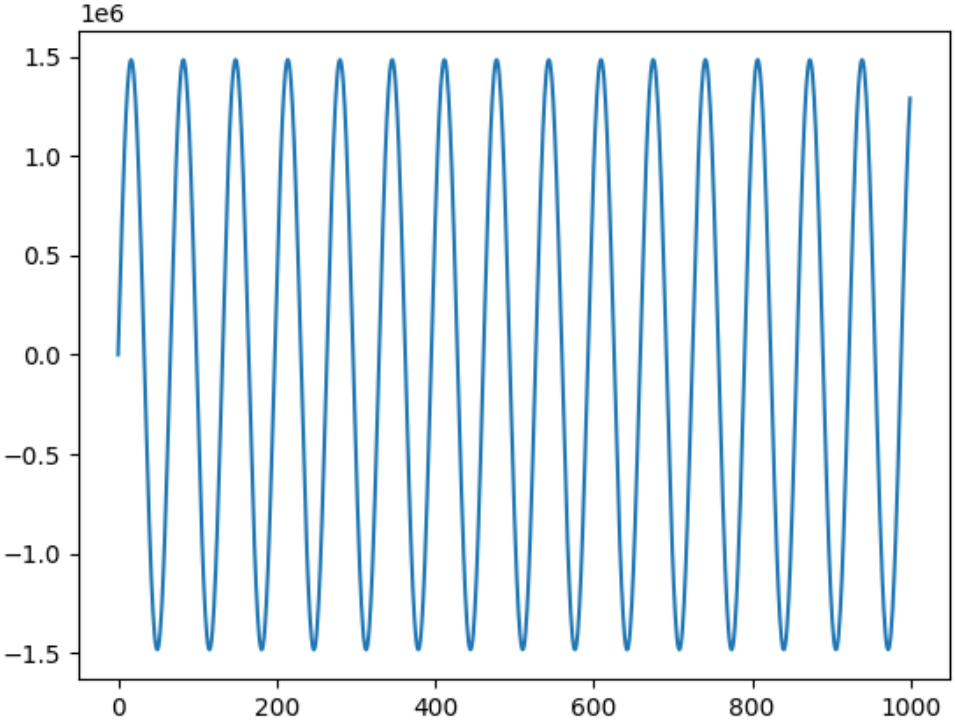
[ ]:
```
# Storing with list

n_turns = 1000

particleEnergyList = []
particlePhaseList = []
for i in range(n_turns):
    particleEnergyList.append(particleEnergy)
    particlePhaseList.append(particlePhase)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)
```

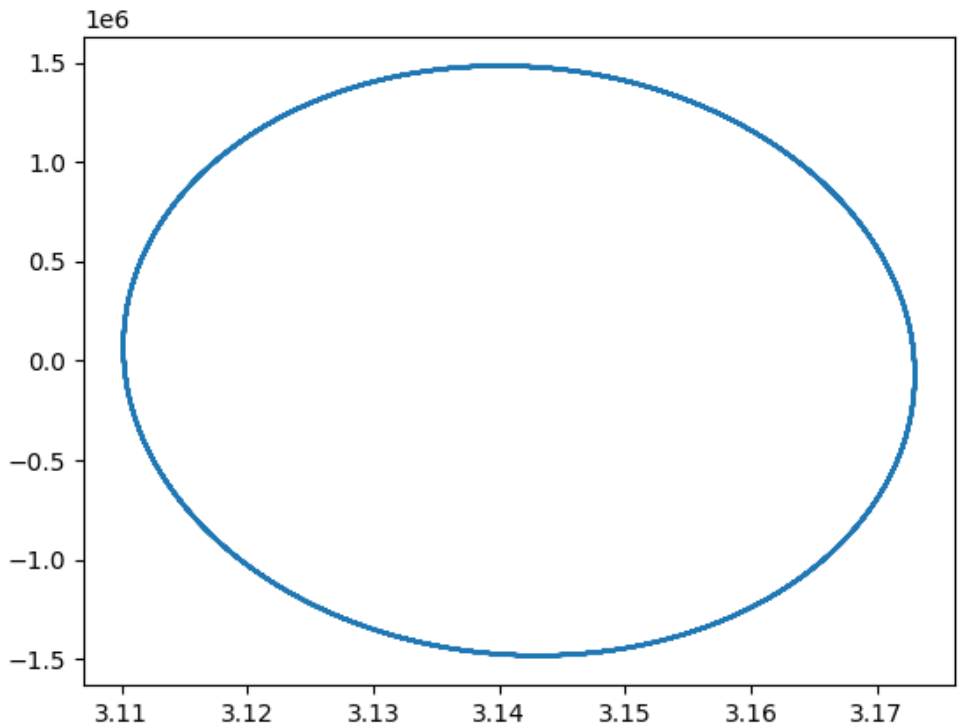
[ ]:
```
# Storing with np array

n_turns = 1000

particleEnergyArray = np.zeros(n_turns)
particlePhaseArray = np.zeros(n_turns)
for i in range(n_turns):
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhaseArray[i] = np.array(particlePhase)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)
```

[ ]:
```
plt.figure()
plt.plot(particlePhaseList)
plt.figure()
plt.plot(particleEnergyList)
plt.figure()
plt.plot(particlePhaseList, particleEnergyList)
```

[ ]: [<matplotlib.lines.Line2D at 0x215664593a0>]




1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
200
400
600
800
1000

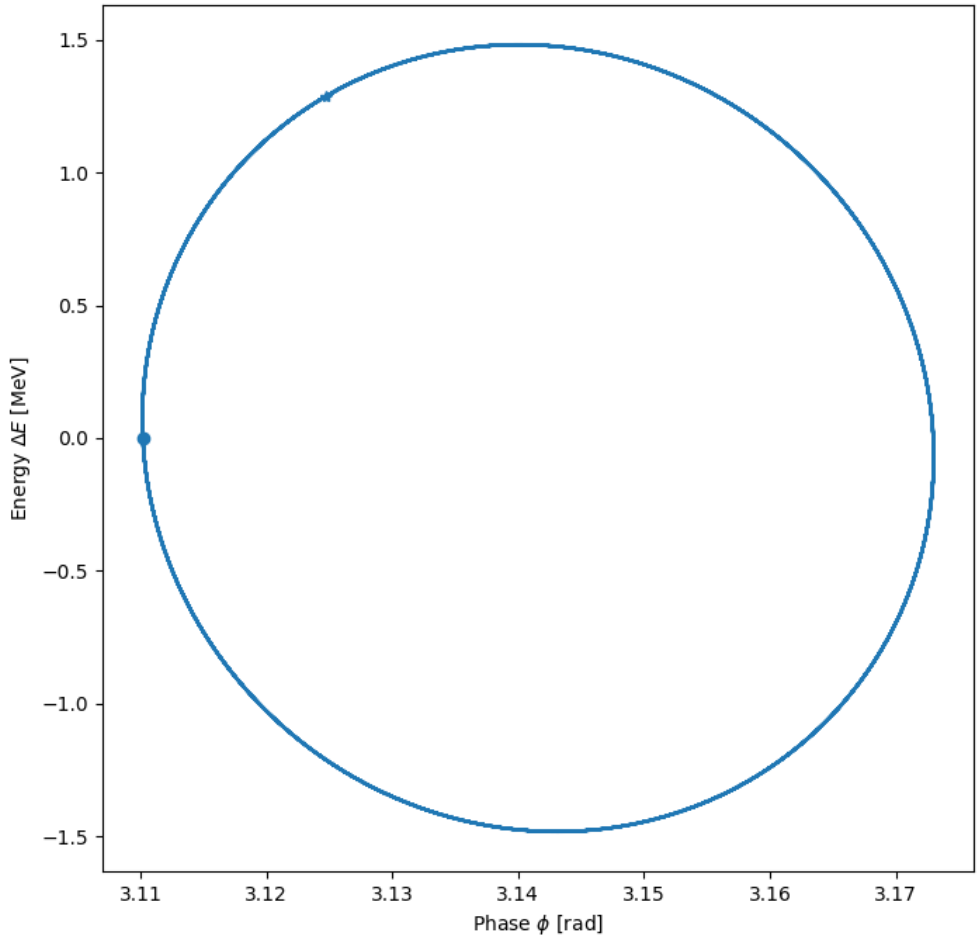


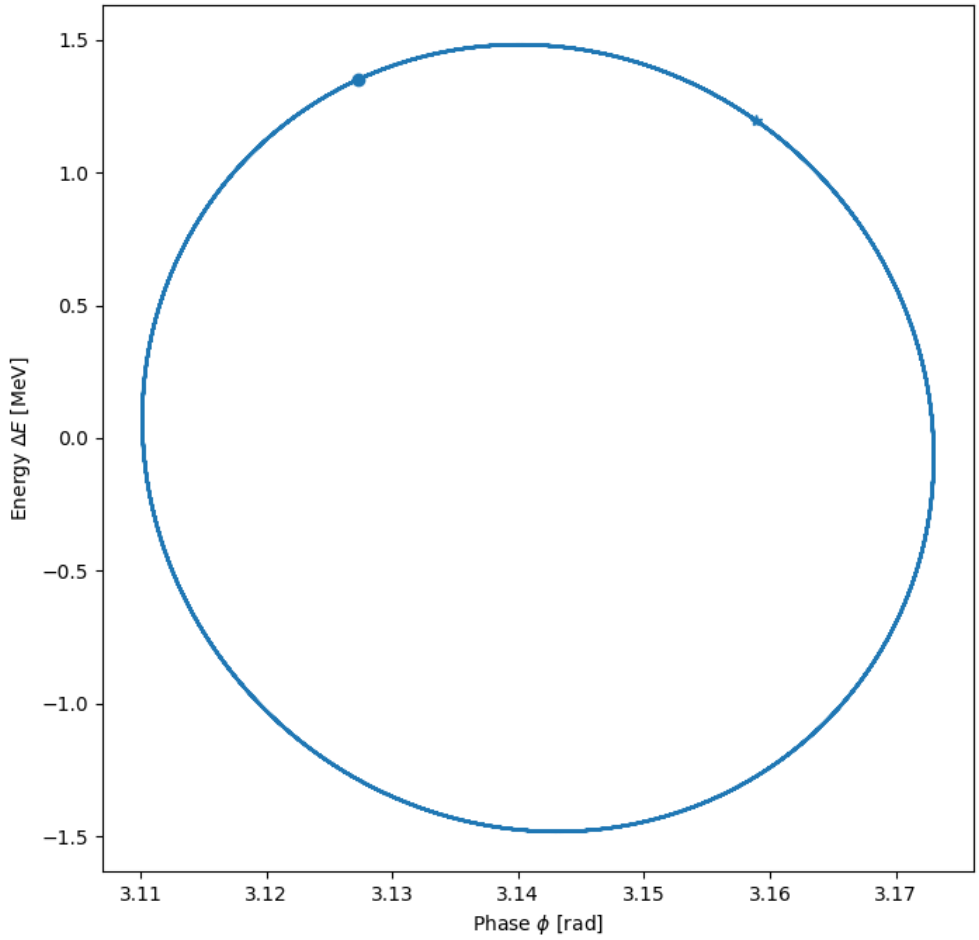
[ ]:
```
from support_functions import plot_phase_space_trajectory
plot_phase_space_trajectory?
```

```
Signature:
plot_phase_space_trajectory(
    phase_trajectory,
    energy_trajectory,
    phase_sep=None,
    separatrix_array=None,
    figname=None,
    draw_line=True,
    xlim=None,
    ylim=None,
)
Docstring:
This support function can be used to plot the trajectory of a particle
```

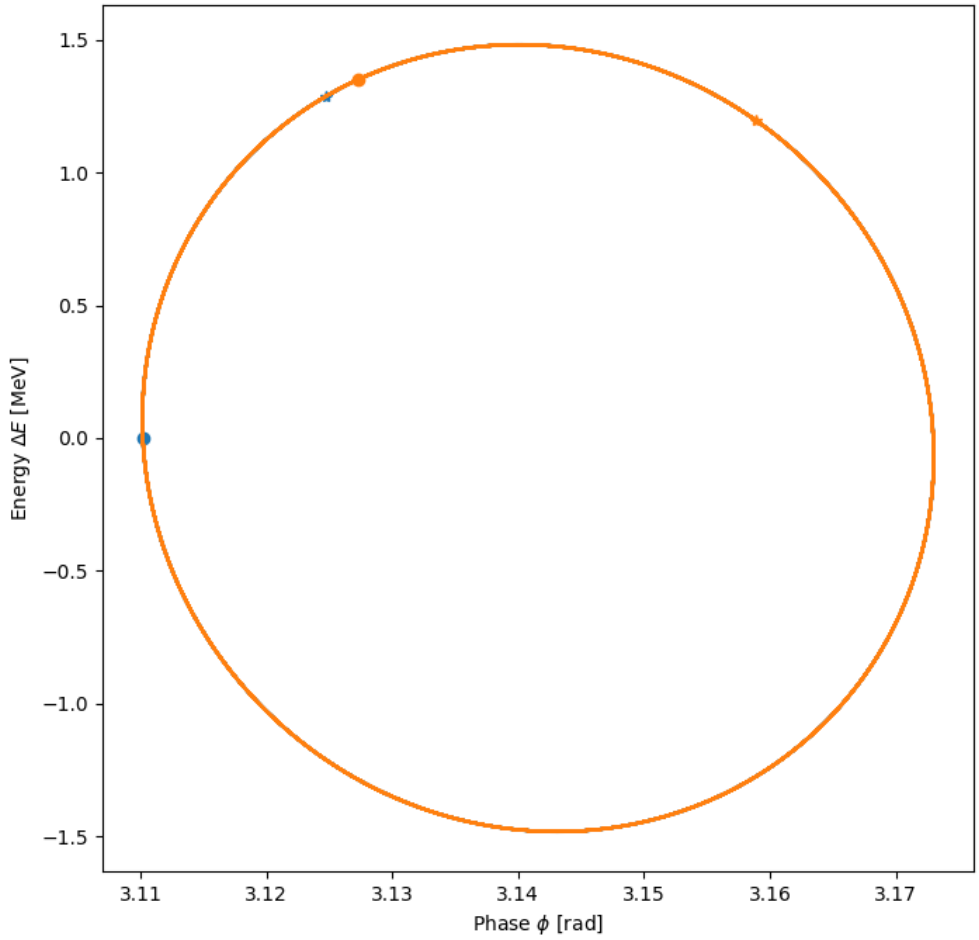
```
in the longitudinal phase space. The o marker is the starting coordinate,
the * marker is the end coordinate.

Parameters
----------
phase_trajectory : np.array or list
    The phase coordinates of a distribution of particles in [rad]
energy_trajectory : np.array or list
    The energy coordinates of a distribution of particles in [eV]
phase_sep : np.array
    The phase of the separatrix array in [rad], as output by the separatrix
function
separatrix_array : np.array
    The separatrix array in [eV], as output by the separatrix function
figname : str, optional
    The name of your nice figure, by default None
draw_line : bool, optional
    Hide the trajectory to plot only the start/end coordinates, by default True
xlim : tuple, optional
    The limits in phase for your nice plot in [rad], e.g. (-np.pi, np.pi), by
default None
ylim : tuple, optional
    The limits in energy for your nice plot [eV], e.g. (-1e6, 1e6), by default
None
File:
c:\users\alasheen\cernbox\storage\redaction\events\2022\cas\git\hands-
on\tracking\support_functions.py
Type:      function
```

[ ]:
```
plot_phase_space_trajectory(particlePhaseList, particleEnergyList)
plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray)
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Energy ΔE [MeV]
3.11
3.12
3.13
3.14
3.15
3.16
3.17
Phase ϕ [rad]



[ ]:
```
figname_traj_1 = 'trajectories 1'
plot_phase_space_trajectory(particlePhaseList, particleEnergyList,␣
 ↪figname=figname_traj_1)
plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray,␣
 ↪figname=figname_traj_1)
```



1.7 Exercise 3: Track with a few particles at different amplitudes

1. Repeat the same operations as in Exercise 2, starting with several particles
   - *You can simply add more particles with different variable names*
   - *You can generate numpy arrays representing several particles as* `np.array([phase_1, phase_2, phase_3])`
   - *You can store the trajectories of the particles in a pre-allocated array of size* `np.zeros((n_turns, n_particles))`
2. Track the evolution of particles including with large offsets in ϕ and ΔE
   - *What happens when particles are too far from the synchronous particle?*
3. Generate about 10 particles at $\Delta E = 0$ with different phases, and simulate for a few turns
   - *You can use* `np.linspace(phase_start, phase_end, n_particles)` *to linearly space particles in phase*

- *What can you observe regarding the velocity of the particles in phase space, vs. the maximum amplitude in phase?*

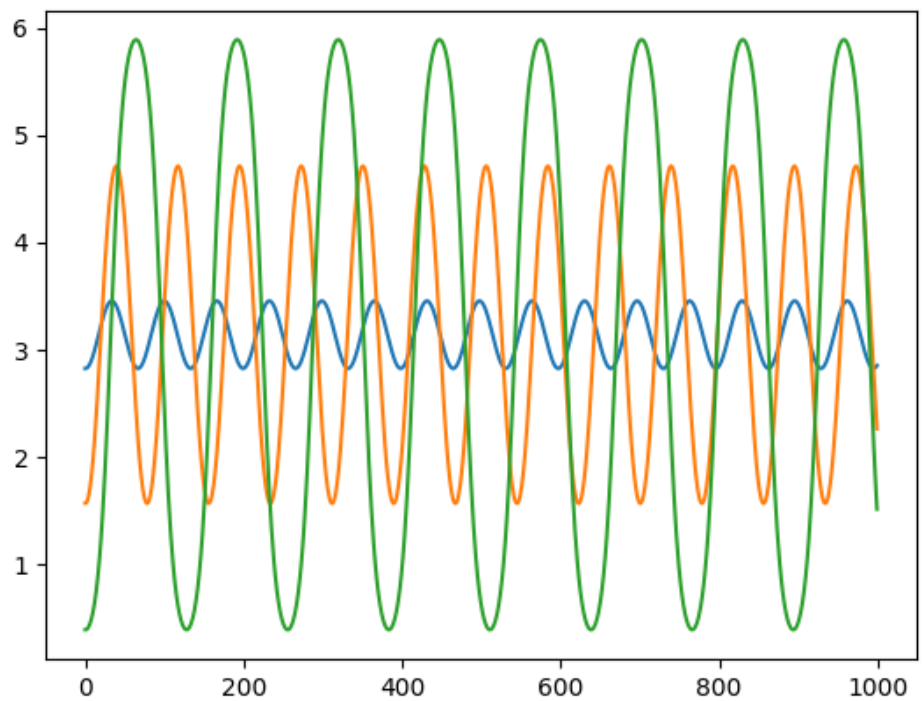
4. Plot the separatrix on top of your plot
   - *Use the* ***separatrix*** *support_function to generate the separatrix.*
   - *You can pass the separatrix to the* ***plot_phase_space_trajectory*** *to combine plots*

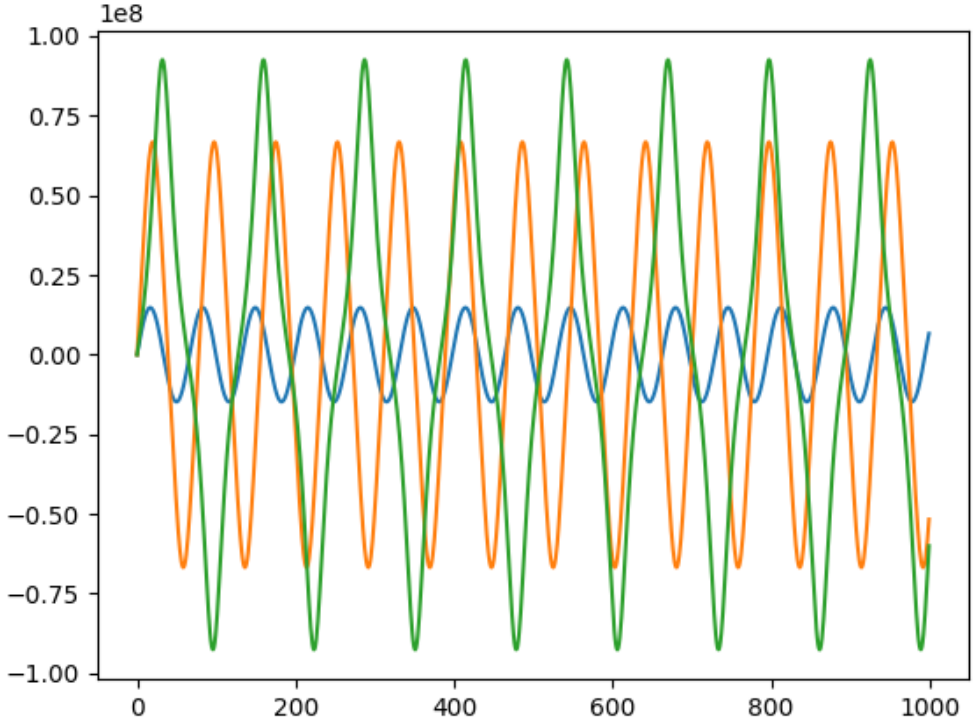
[ ]:
```
# Initial coordinates

particle1Phase = 0.9*np.pi
particle1Energy = 0

particle2Phase = np.pi/2
particle2Energy = 0

particle3Phase = np.pi/8
particle3Energy = 0
```

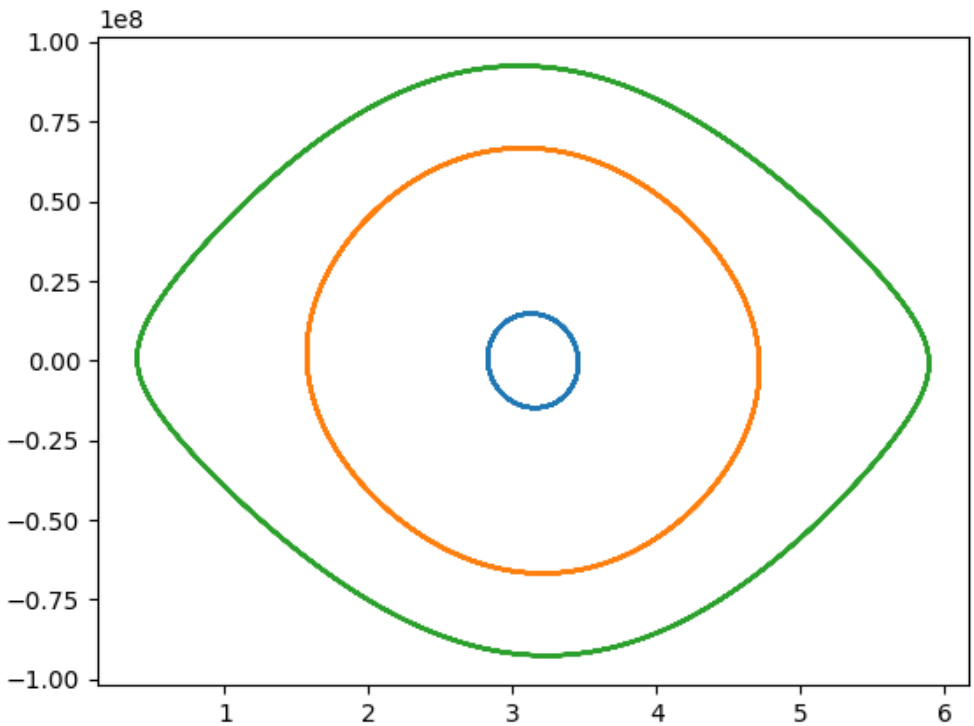
[ ]:
```
# Each particle treated separately

particle1EnergyList = []
particle2EnergyList = []
particle3EnergyList = []
particle1PhaseList = []
particle2PhaseList = []
particle3PhaseList = []
for i in range(n_turns):
    particle1EnergyList.append(particle1Energy)
    particle2EnergyList.append(particle2Energy)
    particle3EnergyList.append(particle3Energy)
    particle1PhaseList.append(particle1Phase)
    particle2PhaseList.append(particle2Phase)
    particle3PhaseList.append(particle3Phase)
    particle1Phase = drift(particle1Phase, particle1Energy, harmonic, eta,␣
 ↪beta, energy)
    particle2Phase = drift(particle2Phase, particle2Energy, harmonic, eta,␣
 ↪beta, energy)
    particle3Phase = drift(particle3Phase, particle3Energy, harmonic, eta,␣
 ↪beta, energy)
    particle1Energy = kick(particle1Energy, particle1Phase, charge, voltage)
    particle2Energy = kick(particle2Energy, particle2Phase, charge, voltage)
    particle3Energy = kick(particle3Energy, particle3Phase, charge, voltage)

plt.figure('energy evolution several')
plt.plot(particle1PhaseList)
plt.plot(particle2PhaseList)
plt.plot(particle3PhaseList)
plt.figure('phase evolution several')
```

```
plt.plot(particle1EnergyList)
plt.plot(particle2EnergyList)
plt.plot(particle3EnergyList)
plt.figure('phase space several')
plt.plot(particle1PhaseList, particle1EnergyList)
plt.plot(particle2PhaseList, particle2EnergyList)
plt.plot(particle3PhaseList, particle3EnergyList)
```

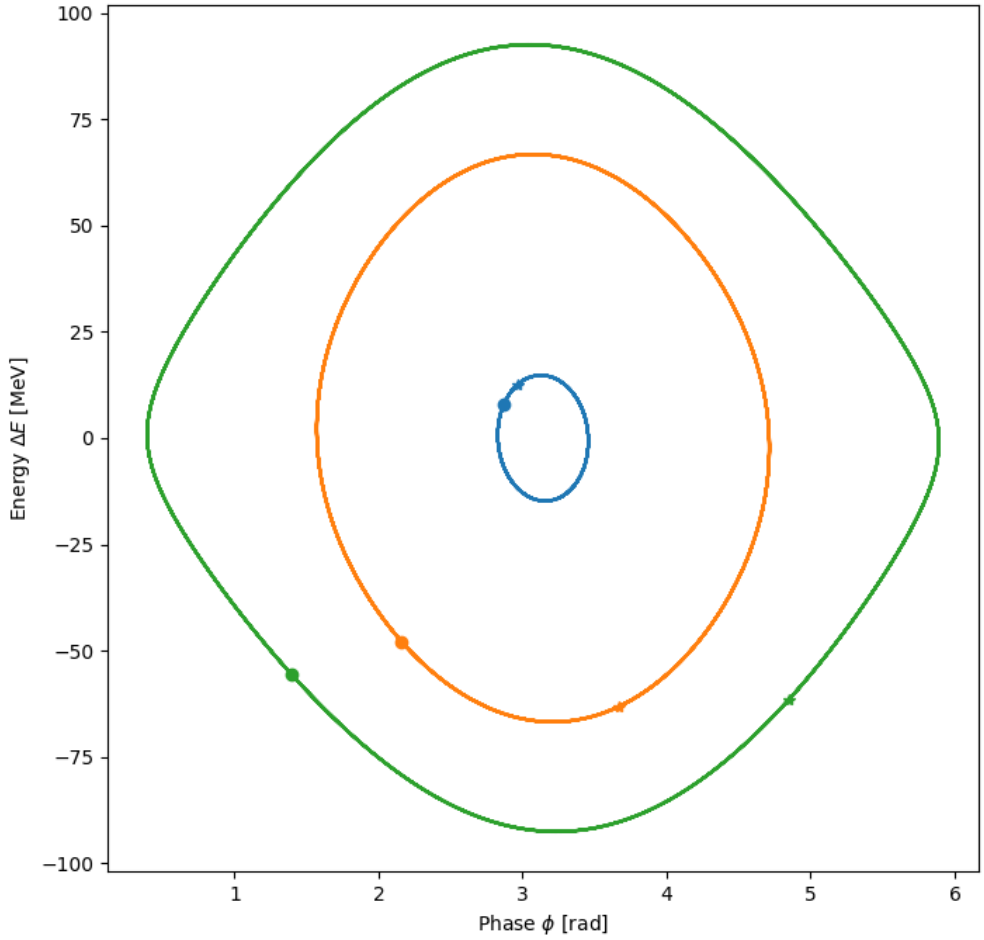
[ ]: [<matplotlib.lines.Line2D at 0x21566912cd0>]




1e8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
200
400
600
800
1000



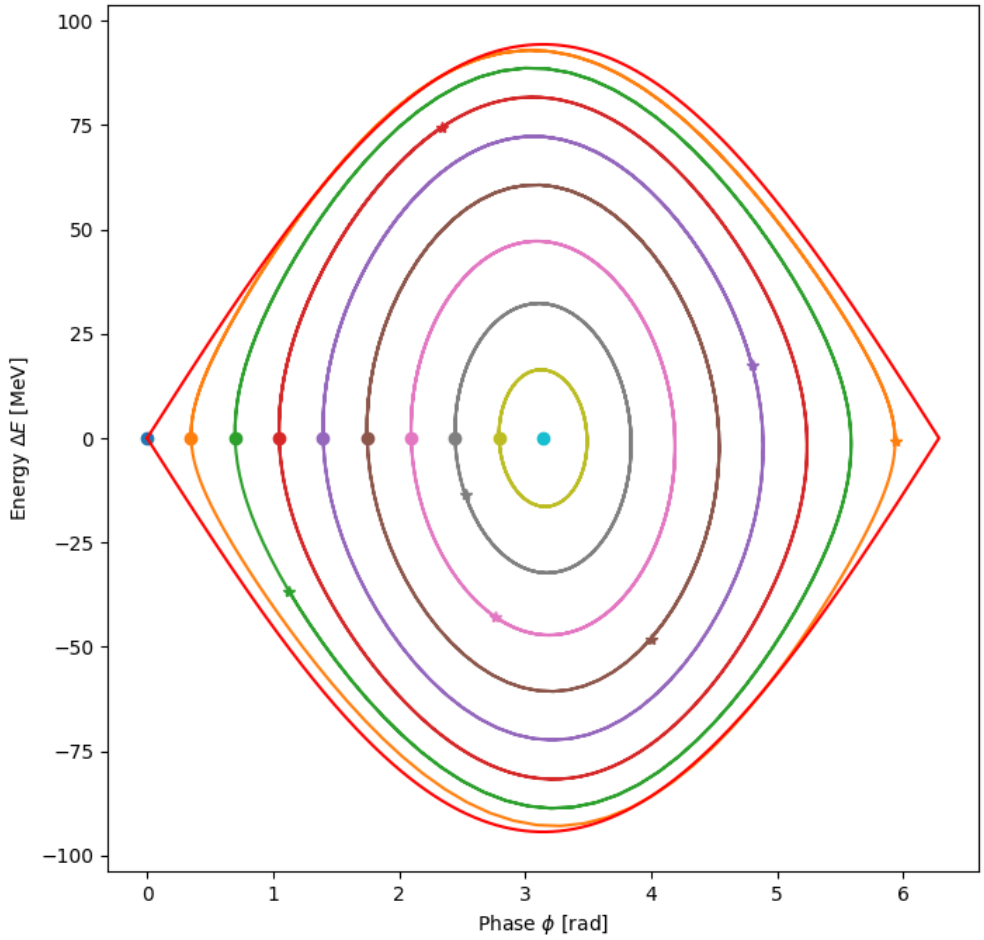
[ ]:
```
# Using numpy arrays

n_particles = 3

particlePhase = np.array([particle1Phase, particle2Phase, particle3Phase])
particleEnergy = np.array([particle1Energy, particle2Energy, particle3Energy])

particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)

plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray)
```



[ ]:
```
n_particles = 10
n_turns = 200

particlePhase = np.linspace(0, np.pi, n_particles)
particleEnergy = np.zeros(n_particles)

particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
```

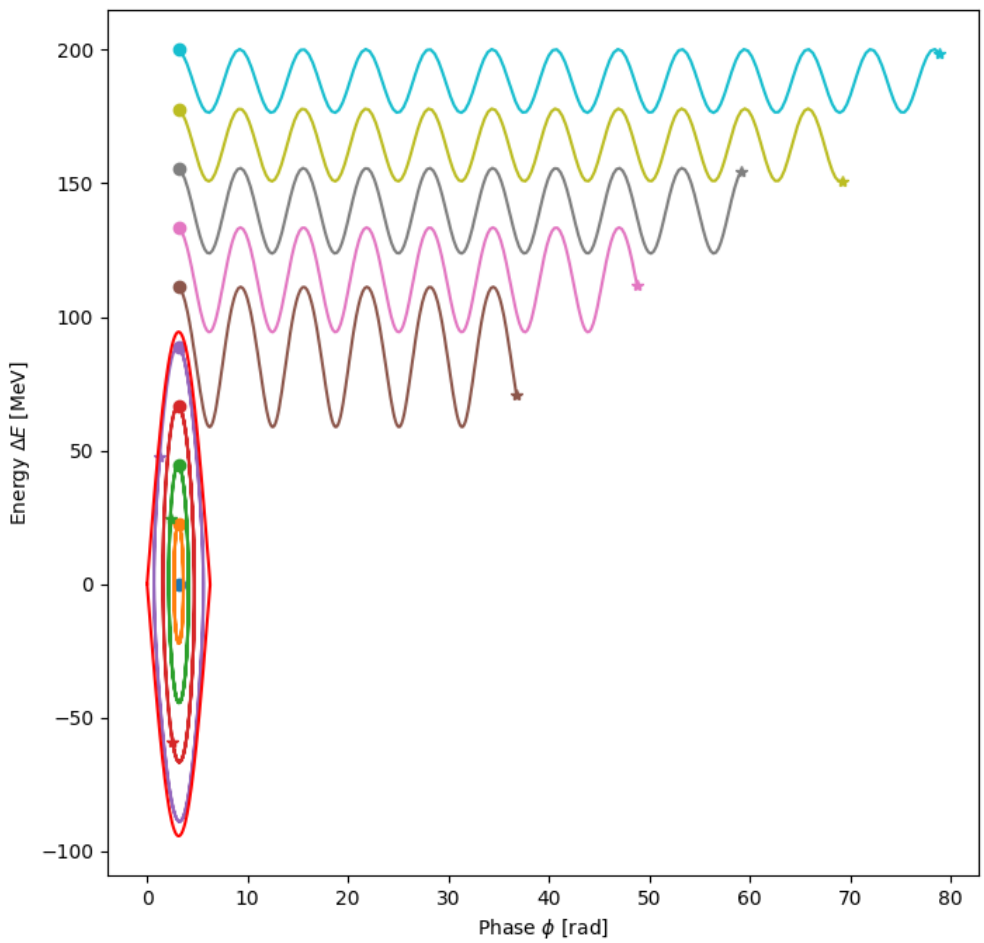
```
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)

from support_functions import separatrix
phase_array = np.linspace(0, 2*np.pi, 1000)
phase_sep, separatrix_array = separatrix(phase_array, f_rev, eta, beta, energy,␣
 ↪charge, voltage, harmonic)

plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray,
                            phase_sep=phase_sep,␣
 ↪separatrix_array=separatrix_array)
```

```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```



[ ]:
```
n_particles = 10
n_turns = 200

particlePhase = np.ones(n_particles) * np.pi
particleEnergy = np.linspace(0, 2e8, n_particles)

particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy,
                          harmonic, eta, beta, energy)
```

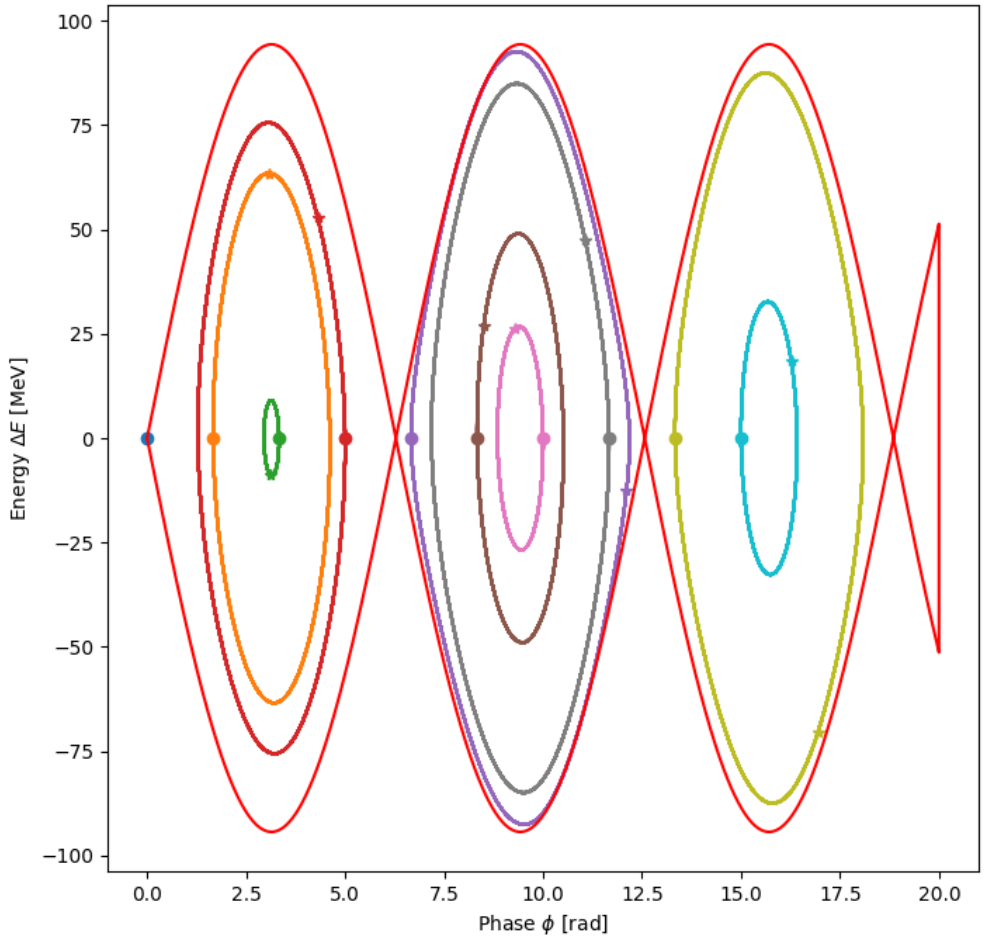
```
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)

plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray,
                            phase_sep=phase_sep,␣
 ↪separatrix_array=separatrix_array)
```



[ ]:
```
n_particles = 10
n_turns = 2000

particlePhase = np.linspace(0, 15, n_particles)
particleEnergy = np.zeros(n_particles)
```

```
particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy,
                          harmonic, eta, beta, energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)

phase_array = np.linspace(0, 20, 10000)
phase_sep, separatrix_array = separatrix(
    phase_array, f_rev, eta, beta, energy, charge, voltage, harmonic)

plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray,
                            phase_sep=phase_sep,␣
 ↪separatrix_array=separatrix_array)
```

```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```



1.8 Exercise 4: Acceleration

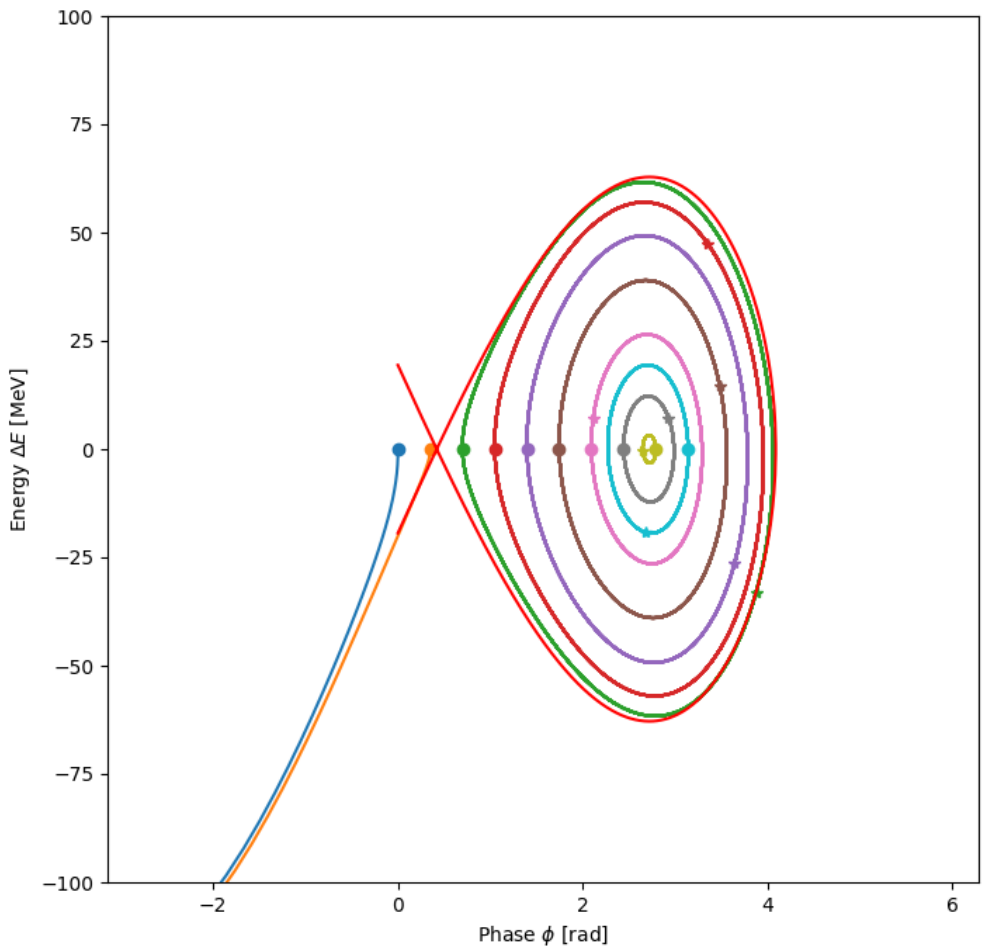
1. Track the particles by adding the acceleration term, which can be evaluated from the parameter table above
   - *For simplicity we will neglect here the variations in β, γ, T, ω…*
2. What is the influence on the particle trajectories ?
   - *You can repeat the same tests as in the previous exercises.*
3. What happens if the ramp rate is twice as fast? Or the voltage is halved?
4. What happens if the beam is decelerated instead?

[ ]:
```
def kick(energyInitial, phaseInitial, charge, voltage, acceleration=0):
    newEnergy = energyInitial + charge * voltage * \
        np.sin(phaseInitial) - acceleration
```

```
    return newEnergy

Ekin_top = 1.3e12
Ekin_bottom = 26e9
t_ramp = 4
U0 = (Ekin_top-Ekin_bottom)/t_ramp / (t_ramp/t_rev)
```

[ ]:
```
n_particles = 10
n_turns = 1000

particlePhase = np.linspace(0, np.pi, n_particles)
particleEnergy = np.zeros(n_particles)

particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage, U0)

phase_array = np.linspace(0, 2*np.pi, 1000)
phase_sep, separatrix_array = separatrix(
    phase_array, f_rev, eta, beta, energy, charge, voltage, harmonic,
    acceleration=U0)

plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray,
                            phase_sep=phase_sep,␣
 ↪separatrix_array=separatrix_array,
                            xlim=(-np.pi, 2*np.pi),
                            ylim=(-100, 100))
```

```
c:\Users\alasheen\cernbox\Storage\Redaction\Events\2022\CAS\git\hands-
on\tracking\support_functions.py:236: RuntimeWarning: invalid value encountered
in sqrt
  separatrix_array = np.sqrt(
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```



1.9 Exercise 5: On the other side of transition energy

1. What if the injection kinetic energy was 14 GeV instead of 26 GeV?
2. What does that change for the bucket and why?

[ ]:
```
Ekin = 14e9

charge = 1
E0 = m_p*c**2./e
circumference = 6911.5
energy = Ekin + E0
momentum = np.sqrt(energy**2. - E0**2.)
beta = momentum/energy
```

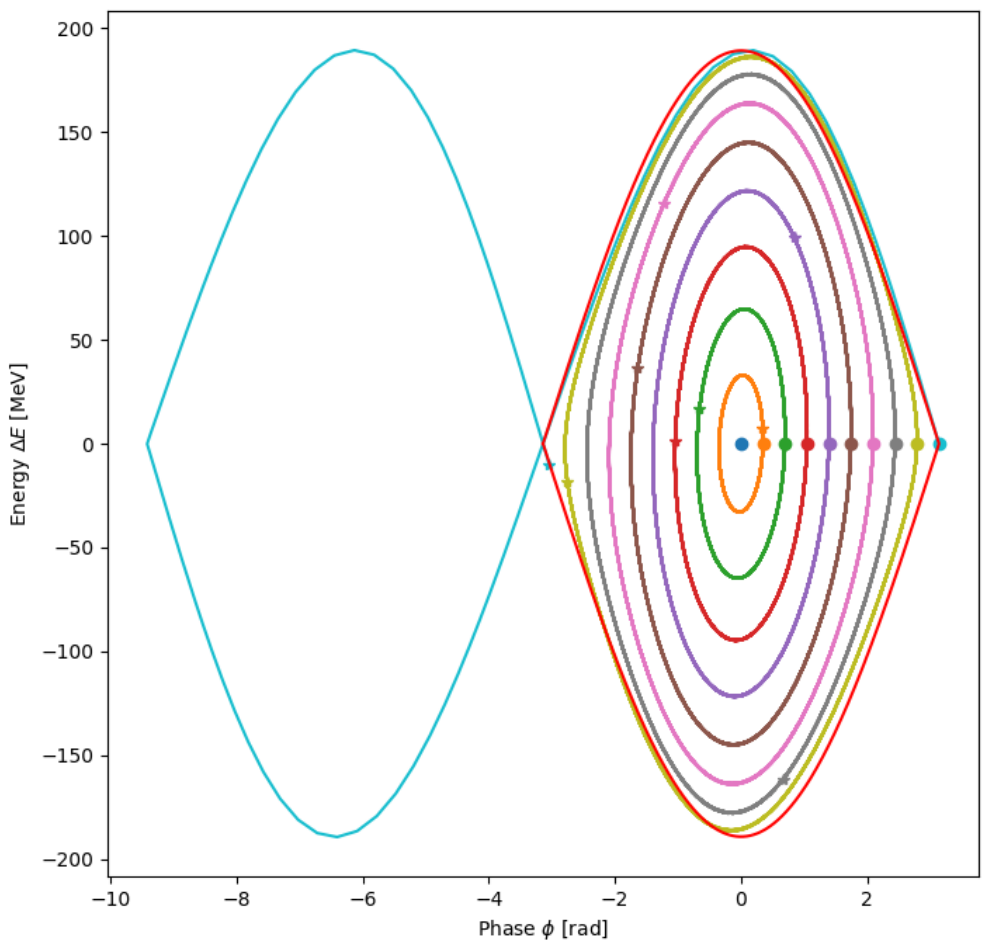
```
gamma = energy/E0


t_rev = circumference/(beta*c)
f_rev = 1/t_rev

harmonic = 4620
voltage = 15e6
f_rf = harmonic*f_rev
t_rf = 1/f_rf

gamma_t = 18
alpha_c = 1/gamma_t**2.
eta = alpha_c - 1/gamma**2.

print("Beta: " +str(beta))
print("Gamma: " +str(gamma))
print("Revolution period: " +str(t_rev*1e6) + " mus")
print("RF frequency: " +str(f_rf/1e6) + " MHz")
print("RF period: " +str(t_rf*1e9) + " ns")
print("Momentum compaction factor: " +str(alpha_c))
print("Phase slippage factor: " +str(eta))
```

```
Beta: 0.9980255059233202
Gamma: 15.921044947044525
Revolution period: 23.099893041602865 mus
RF frequency: 200.00092605101628 MHz
RF period: 4.999976848831789 ns
Momentum compaction factor: 0.0030864197530864196
Phase slippage factor: -0.0008586697734143615
```

[ ]:
```
n_particles = 10
n_turns = 1000

particlePhase = np.linspace(0, np.pi, n_particles)
particleEnergy = np.zeros(n_particles)

particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)

phase_array = np.linspace(-np.pi, np.pi, 1000)
```

```
phase_sep, separatrix_array = separatrix(
    phase_array, f_rev, eta, beta, energy, charge, voltage, harmonic)

plot_phase_space_trajectory(particlePhaseArray, particleEnergyArray,
                            phase_sep=phase_sep,␣
 ↪separatrix_array=separatrix_array)
```

```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```



1.10 Exercise 6: Comparison with analytical evaluations

1. Calculate the height and area of the stationary bucket analytically for lower/higher beam energies.
2. Determine analytically the synchrotron frequency at the centre of the stationary bucket.
3. Compare qualitatively the results you obtained with the tracking
   - *These analytical calculations serve to benchmark the tracking simulations.*
   - *You can also compare with the accelerating cases.*

[ ]:
```
Ekin = 26e9

charge = 1
E0 = m_p * c**2. / e
circumference = 6911.5
energy = Ekin + E0
momentum = np.sqrt(energy**2. - E0**2.)
beta = momentum / energy
gamma = energy / E0

t_rev = circumference / (beta * c)
f_rev = 1 / t_rev
omega_rev = 2 * np.pi * f_rev

harmonic = 4620
voltage = 4.5e6
f_rf = harmonic * f_rev
t_rf = 1 / f_rf
omega_rf = 2 * np.pi * f_rf

gamma_t = 18
alpha_c = 1 / gamma_t**2.
eta = alpha_c - 1 / gamma**2.

phi_s = np.arcsin(U0 / voltage)

stationaryBucketHeight = beta * \
    np.sqrt(2 * charge * voltage * energy / (np.pi * harmonic * np.abs(eta)))
print("Bucket height: " +
      str(stationaryBucketHeight / 1e6) + " MeV")

bucketHeightRed = np.abs(
    np.cos(phi_s) - (np.pi - 2 * phi_s) / 2 * np.sin(phi_s))**(1 / 2.)
print("Accelerating bucket height: " +
      str(stationaryBucketHeight / 1e6 * bucketHeightRed) + " MeV")

stationaryBucketArea = 16 * beta / omega_rf * \
```

```
    np.sqrt(charge * voltage * energy / (2 * np.pi * harmonic * np.abs(eta)))
print("Bucket area: " +
      str(stationaryBucketArea) + " eVs")

bucketAreaRed = (1 - np.sin(phi_s)) / (1 + np.sin(phi_s))
print("Accelerating bucket area: " +
      str(stationaryBucketArea * bucketAreaRed) + " eVs")

stationaryBucketCentralSynchrotronFrequency = np.sqrt(-omega_rev**2 * harmonic *
                                                      eta * charge * voltage *␣
 ↪np.cos(np.pi) / (2 * np.pi * beta**2 * energy)) / (2 * np.pi)
stationaryBucketCentralSynchrotronTune =␣
 ↪stationaryBucketCentralSynchrotronFrequency / f_rev
print("Synchtron frequency: " +
      str(stationaryBucketCentralSynchrotronFrequency) + " Hz")
print("Synchtron period: " +
      str(1 / stationaryBucketCentralSynchrotronFrequency * 1e3) + " ms")
print("Inverse tune: " +
      str(1 / stationaryBucketCentralSynchrotronTune) + " turns")
```

```
Bucket height: 94.37296851790389 MeV
Accelerating bucket height: 62.83826390584678 MeV
Bucket area: 0.5999719724218308 eVs
Accelerating bucket area: 0.25215177121828763 eVs
Synchtron frequency: 657.9604989768133 Hz
Synchtron period: 1.5198480783498225 ms
Inverse tune: 65.88476102073503 turns
```

1.11 Exercise 7: Mismatched distribution, bunch rotation

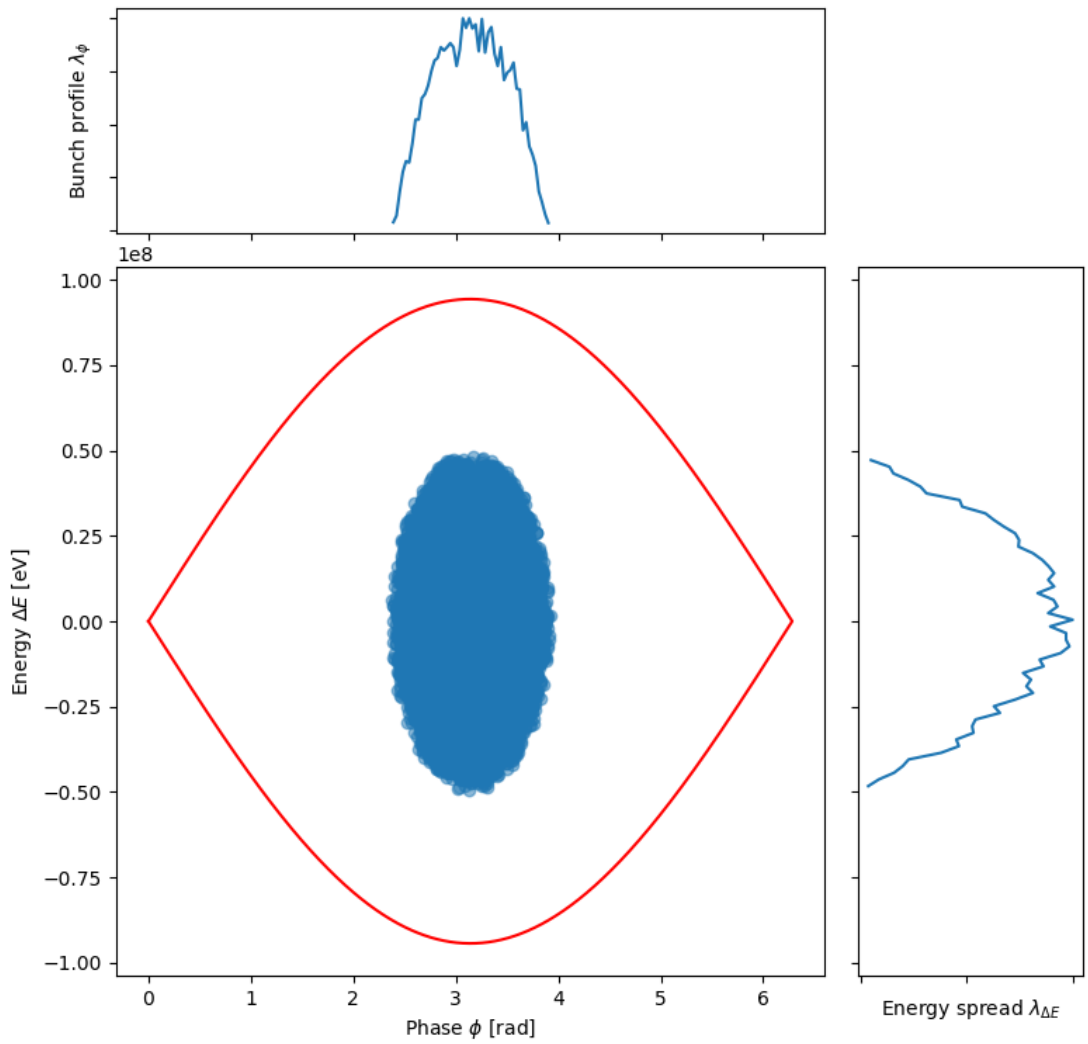
1. Generate a bunch distribution using the `generate_bunch` function.
   - *Use the* ***`plot_phase_space_distribution`*** *to plot the distribution in phase space*
   - *Use your tracking routine to see the bunch distribution before/after tracking*
   - *Use the* ***`run_animation`*** *function to monitor the evolution of the distribution while tracking*
   - *For the* ***`run_animation`*** *please call if you have trouble with the function syntax*
2. Try to match the bunch to the bucket. Track it for many synchrotron periods.
3. Introduce a phase or energy error between bunch and bucket.
4. Introduce a voltage mismatch. What do you observe?

[ ]:
```
from support_functions import generate_bunch

bunch_position = np.pi
bunch_length = np.pi/2

energy_position = 0
energy_spread = 100e6
```

```
n_macroparticles = 10000

phase_coordinates, energy_coordinates = generate_bunch(
    bunch_position, bunch_length,
    energy_position, energy_spread,
    n_macroparticles)
```

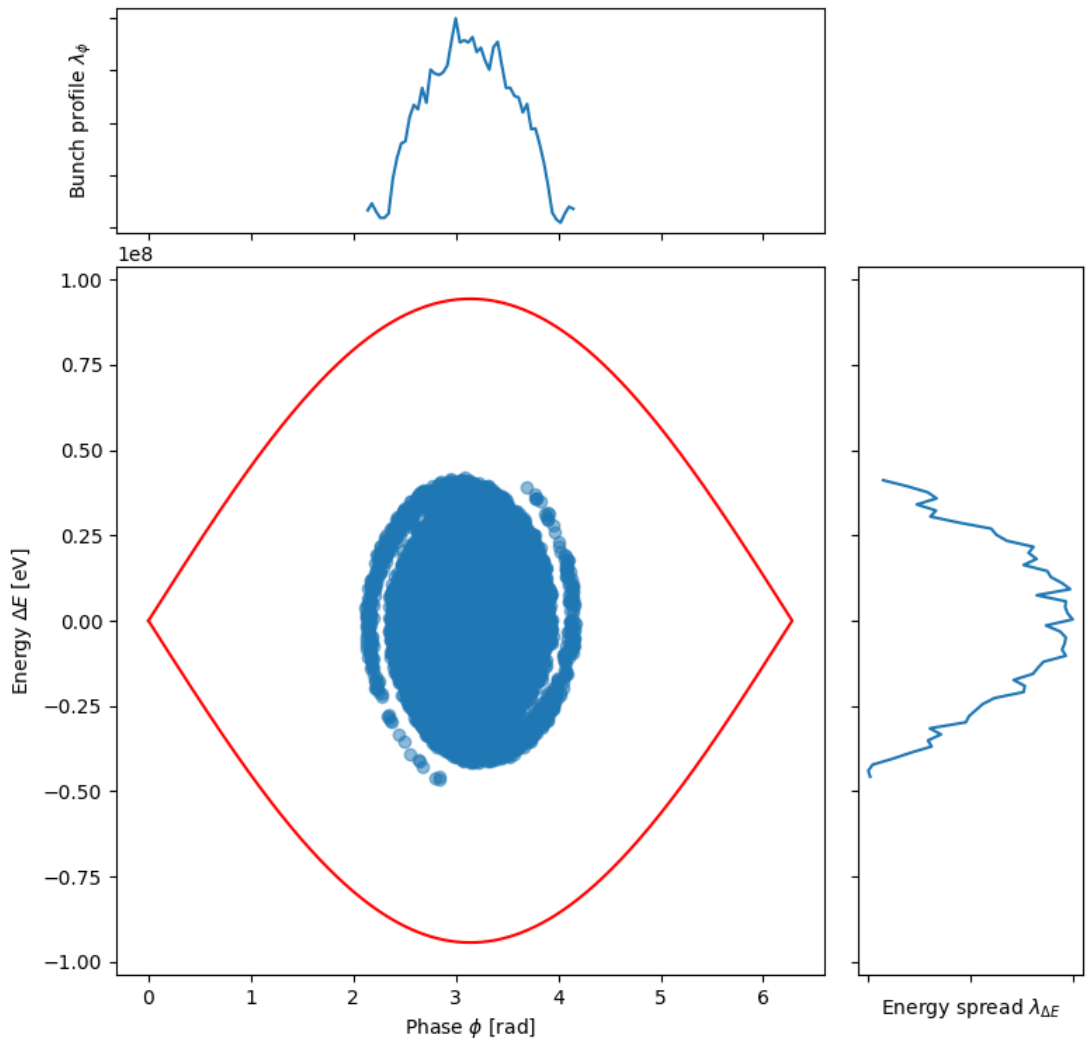
[ ]:
```
from support_functions import plot_phase_space_distribution

phase_array = np.linspace(0, 2 * np.pi, 1000)
phase_sep, separatrix_array = separatrix(
    phase_array, f_rev, eta, beta, energy, charge, voltage, harmonic)

plot_phase_space_distribution(
    phase_coordinates,
    energy_coordinates,
    phase_sep=phase_sep,
    separatrix_array=separatrix_array,)
```

```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```



[ ]:
```
n_turns = 1000

for i in range(n_turns):
    phase_coordinates = drift(phase_coordinates, energy_coordinates, harmonic,␣
 ↪eta, beta, energy)
    energy_coordinates = kick(energy_coordinates, phase_coordinates, charge,␣
 ↪voltage)

plot_phase_space_distribution(
    phase_coordinates,
    energy_coordinates,
    phase_sep=phase_sep,
```

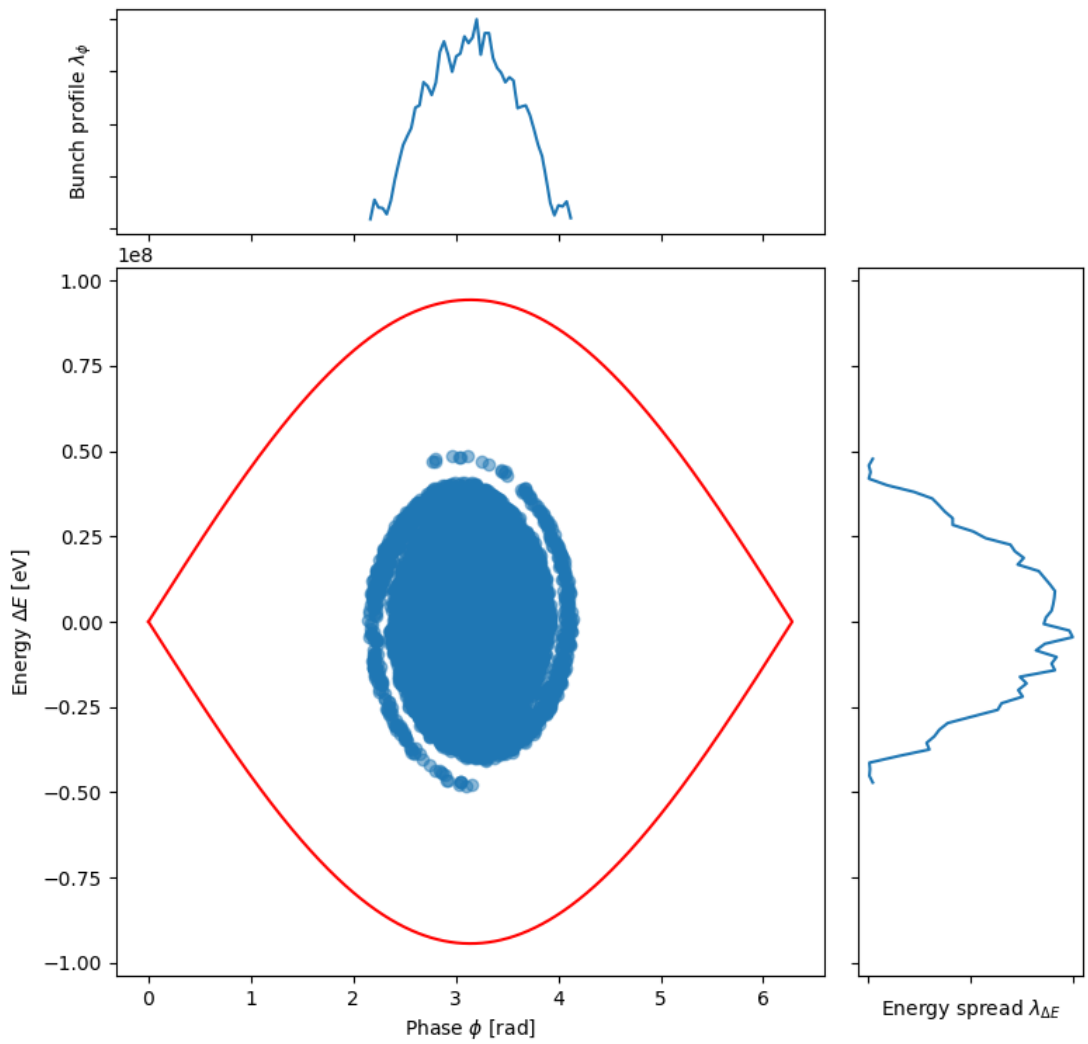
```
    separatrix_array=separatrix_array,)
```



[ ]:
```
from support_functions import run_animation

figname = 'Animate'
iterations = 100
framerate = 30

plt.rcParams['animation.embed_limit'] = 2**32

run_animation(phase_coordinates, energy_coordinates,
              drift, kick,
```

```
                [harmonic, eta, beta, energy],
                [charge, voltage],
                figname, iterations, framerate,
                phase_sep=phase_sep,
                separatrix_array=separatrix_array)
```

[ ]: <IPython.core.display.HTML object>



[ ]:
```
bunch_position = np.pi/2
bunch_length = np.pi/4

energy_position = 0
energy_spread = 100e6
```

```
phase_coordinates, energy_coordinates = generate_bunch(
    bunch_position, bunch_length,
    energy_position, energy_spread,
    n_macroparticles)

figname = 'Dip'

run_animation(phase_coordinates, energy_coordinates,
              drift, kick,
              [harmonic, eta, beta, energy],
              [charge, voltage],
              figname, 4*iterations, framerate,
              phase_sep=phase_sep,
              separatrix_array=separatrix_array)
```

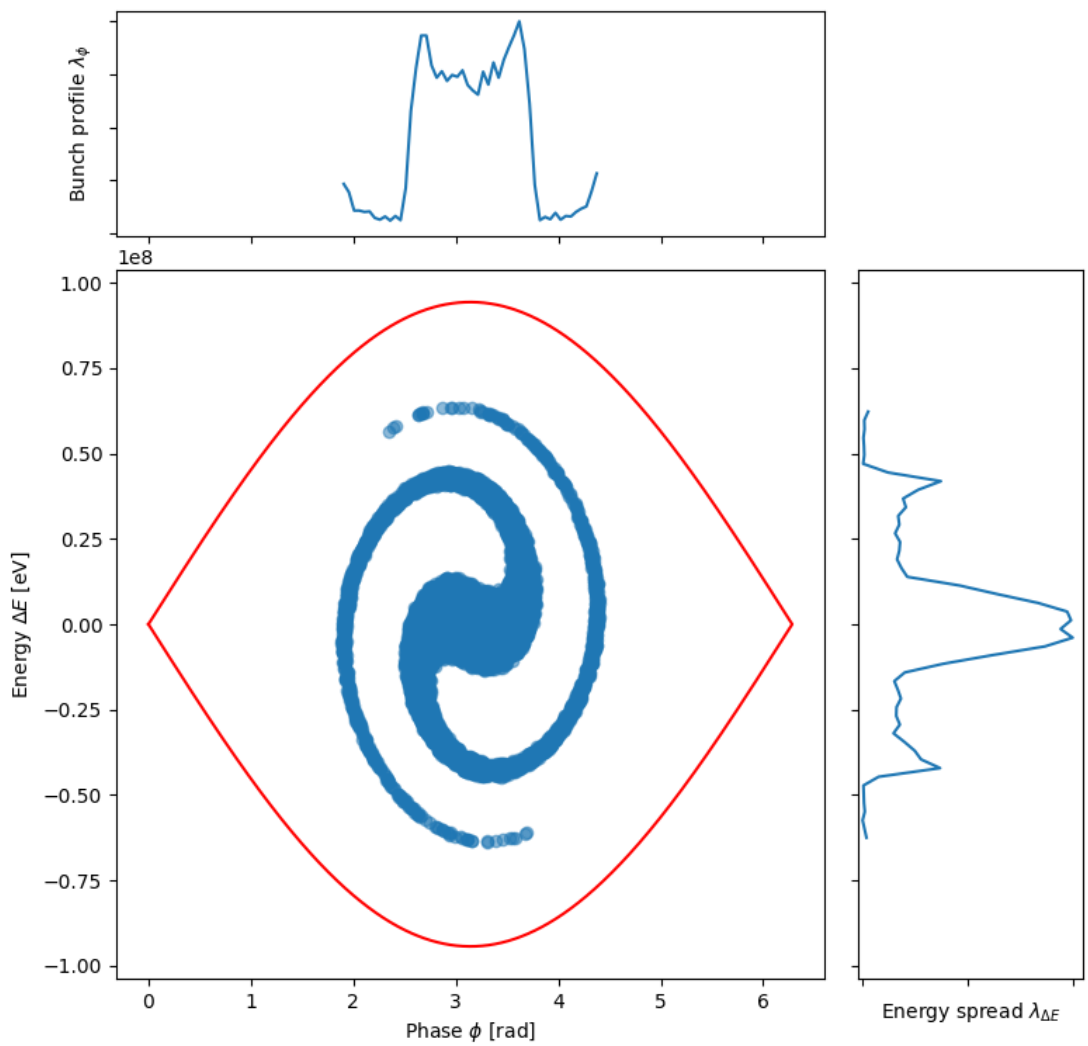
```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```

[ ]:
```
bunch_position = np.pi
bunch_length = np.pi

energy_position = 0
energy_spread = 25e6

phase_coordinates, energy_coordinates = generate_bunch(
    bunch_position, bunch_length,
    energy_position, energy_spread,
    n_macroparticles)

figname = 'Quad'

run_animation(phase_coordinates, energy_coordinates,
              drift, kick,
              [harmonic, eta, beta, energy],
              [charge, voltage],
              figname, 4*iterations, framerate,
              phase_sep=phase_sep,
              separatrix_array=separatrix_array)
```

```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
```

```
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```

[ ]: <IPython.core.display.HTML object>



1.12 Exercise 8: Non-linear synchrotron frequency distribution, comparison with analytical formula

1. Track a few tens of particles for a few synchrotron periods to analyze the frequency of synchrotron oscillation.
   - *You can start with the same script as in Exercise 3 to start with few particles.*

- *The function* ***oscillation_spectrum*** *returns the spectrum of phase or energy oscillations for a given particle.*
- *The function* ***synchrotron_tune*** *returns the tune of a given particle, based on the maximum of the oscillation spectrum obtained with FFT.*

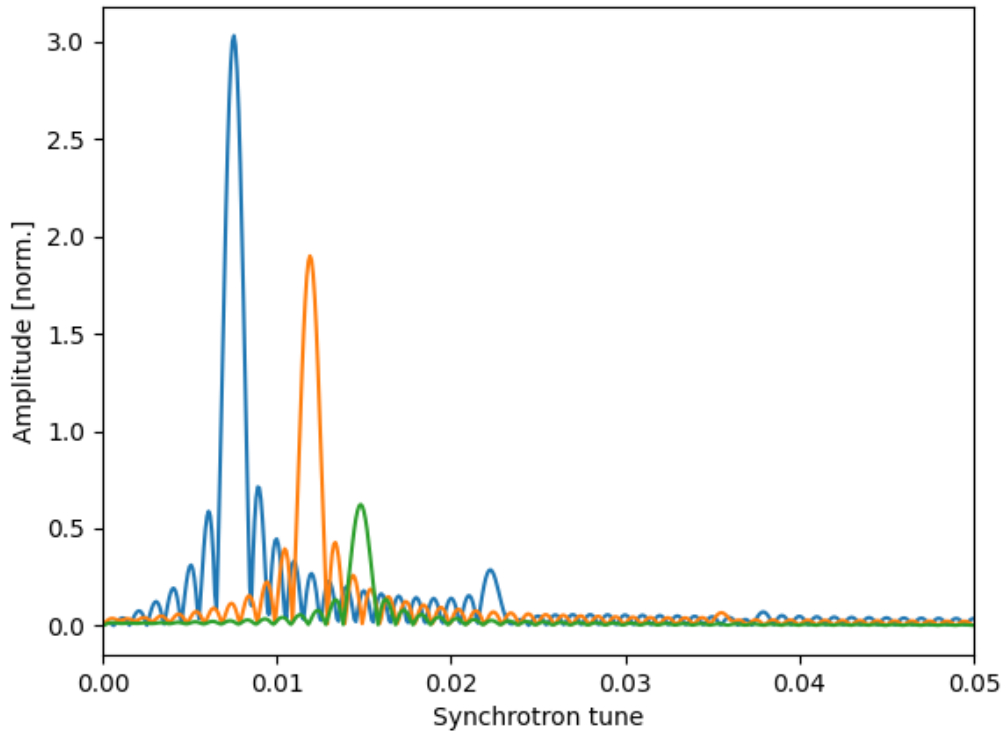
2. Plot the synchrotron frequency versus phase or energy offset. This illustrates the synchrotron frequency distribution.
   - *Beware of particles extremely close to the center of the bucket or exactly on the separatrix*
3. Compare with the expected depedence of the non-linear synchrotron tune from the cheat sheet.

[ ]:
```
n_particles = 100
n_turns = 1000

particlePhase = np.linspace(0.01*np.pi, 0.99*np.pi, n_particles)
particleEnergy = np.zeros(n_particles)

particleEnergyArray = np.zeros((n_turns, n_particles))
particlePhaseArray = np.zeros((n_turns, n_particles))
for i in range(n_turns):
    particlePhaseArray[i] = np.array(particlePhase)
    particleEnergyArray[i] = np.array(particleEnergy)
    particlePhase = drift(particlePhase, particleEnergy, harmonic, eta, beta,␣
 ↪energy)
    particleEnergy = kick(particleEnergy, particlePhase, charge, voltage)
```

[ ]:
```
from support_functions import oscillation_spectrum

plt.figure()
plt.plot(*oscillation_spectrum(particlePhaseArray[:, 10],␣
 ↪fft_zero_padding=10000))
plt.plot(*oscillation_spectrum(particlePhaseArray[:, 40],␣
 ↪fft_zero_padding=10000))
plt.plot(*oscillation_spectrum(particlePhaseArray[:, 80],␣
 ↪fft_zero_padding=10000))
plt.xlim(0, 0.05)
plt.xlabel("Synchrotron tune")
plt.ylabel("Amplitude [norm.]")
```

```
c:\Users\alasheen\cernbox\Software\win\python-3.9.12\lib\site-
packages\ipywidgets\widgets\widget.py:443: DeprecationWarning: Passing
unrecognized arguments to super(Toolbar).__init__().
__init__() missing 1 required positional argument: 'canvas'
This is deprecated in traitlets 4.2.This error will be raised in a future
release of traitlets.
  super().__init__(**kwargs)
```

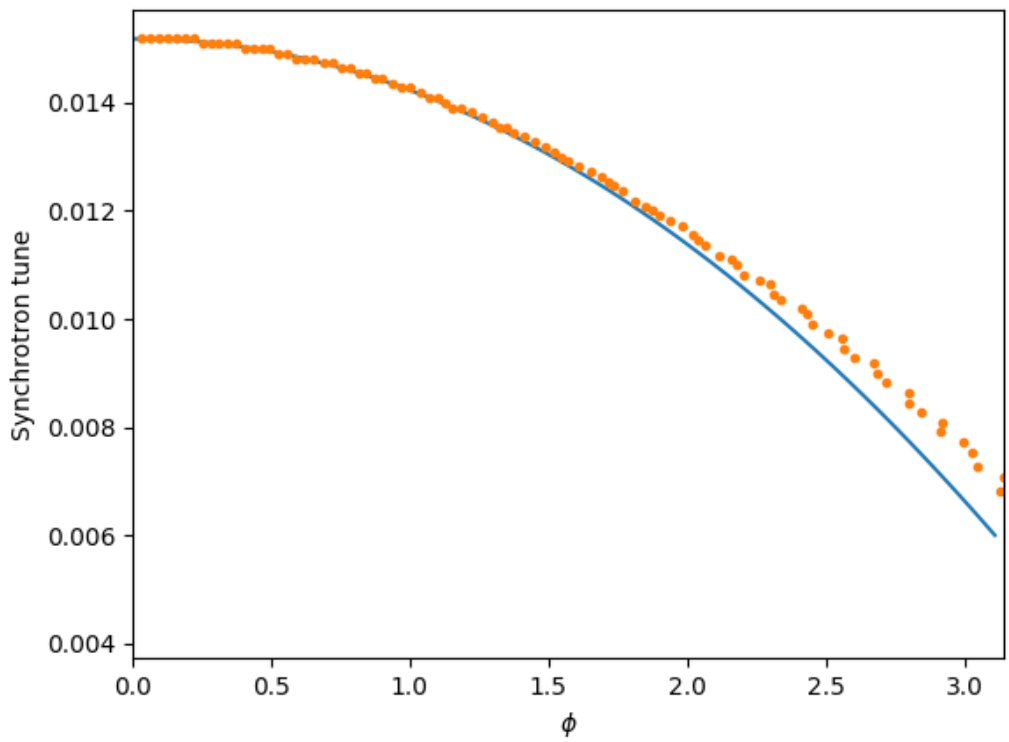
[ ]:
```
Text(0, 0.5, 'Amplitude [norm.]')
```



[ ]:
```
from support_functions import synchrotron_tune

f_s0 = np.sqrt(
    harmonic * eta * charge * voltage / (2 * np.pi * energy * beta**2))

synchrotronTuneDistribution = np.zeros([2, n_particles])
for idx_part in range(n_particles):
    synchrotronTuneDistribution[0, idx_part], synchrotronTuneDistribution[1,␣
 ↪idx_part] = \
        synchrotron_tune(particlePhaseArray[:,idx_part], fft_zero_padding=10000)

synch_tune_amplitude = np.linspace(0, np.pi, 100, endpoint=False)
sync_tune_approx = f_s0 * (1 - synch_tune_amplitude**2 / 16)

plt.figure()
plt.plot(synch_tune_amplitude, sync_tune_approx)
plt.plot(*synchrotronTuneDistribution, ".")
plt.xlim(0, np.pi)
plt.xlabel(r"$\phi$")
plt.ylabel("Synchrotron tune")
```

```
C:\Users\alasheen\AppData\Local\Temp\ipykernel_10512\1408290109.py:14:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  plt.figure()
```

```
[ ]: Text(0, 0.5, 'Synchrotron tune')
```



1.13 Exercises, to infinity and beyond...

- Include synchrotron radiation at 1.3 TeV
- Add second harmonic RF system in the tracking
- What happens when operating both RF systems in phase (bunch-shortening) or in counter-phase (bunch-lengthening)?
- Check the effect on the synchrotron frequency.
- Animate the evolution of the bunch distribution for the test cases of exercise 9.