Longitudinal tracking simulations

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

Links

- Introductory CAS website (http://cas.web.cern.ch/schools/vysoke-tatry-2019)
- Programme (http://cas.web.cern.ch/sites/cas.web.cern.ch/files/programmes/vysoke-tatry-2019-programme_2.pdf)

Python distribution for transverse exercises in Slovakia (from Guido): https://codimd.web.cern.ch/s/HkfQy3YbB (https://codimd.web.cern.ch/s/HkfQy3YbB)

First afternoon

Functions and classes to be imported from 'support_functions.py'

Function	Syntax
Plot phase space	`plotPhaseSpace(distribution, figname=None, xbins=50, ybins=50, xlim=None, ylim=None)`
Calculate oscillation spectrum	`oscillation_spectrum(phase_track, fft_zero_padding=0)`
Calculate synchrontron tune	`synchrotron_tune(phase_track, fft_zero_padding=0)`
Generation of a bunch distribution	`generateBunch(bunch_position, bunch_length, bunch_energy, energy_spread, n_macroparticles)`
Calculate separatrix	`separatrix(phase_array, eta, beta, energy, charge, voltage, harmonic, energy_gain)`
Run animation	`run_animation(particles, trackingFunction, figname, iterations, framerate, xbins=50, ybins=50, xlim=None, ylim=None, phase_sep=None, separatrix_array=None)`
Bucket area reduction ratio	`reduction_ratio(deltaE, voltage)`

For discussion

- Define β and η or start from β and γ_{tr}?
- Keep separatrix?

Import modules

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

Exercise 1: Analytically calculate synchrotron frequency bucket height and area

- Calculate the height of the stationary bucket analytically.
- Determine the synchrotron frequency at the centre of the stationary bucket.
- The analytical calculations will serve to benchmark the tracking simulations.

In [2]:
```
energy=10     #typically [MeV], [GeV], [TeV]
beta=0.9      #relativistic velocity factor
charge=1      #in units of the electron charge, just unity for protons
voltage=0.1   #RF voltage, typically in units of [V], [kV], [MV]
harmonic=1    #Harmonic number of the RF system
eta=0.01      #1/gamma**2-1/gamma_transition**2
```

In [3]:
```
stationaryBucketHeight = np.sqrt(2*energy*beta**2*charge*voltage/(np.pi*harmoni
c*eta))
stationaryBucketCentralSynchrotronTune = np.sqrt(harmonic*eta*charge*voltage/(2*
np.pi*energy*beta**2))
print("Synchtron frequency period: "+str(1/stationaryBucketCentralSynchrotronTun
e)+" turns")
print("Bucket height:              "+str(stationaryBucketHeight)+" arbitrary uni
ts")
```

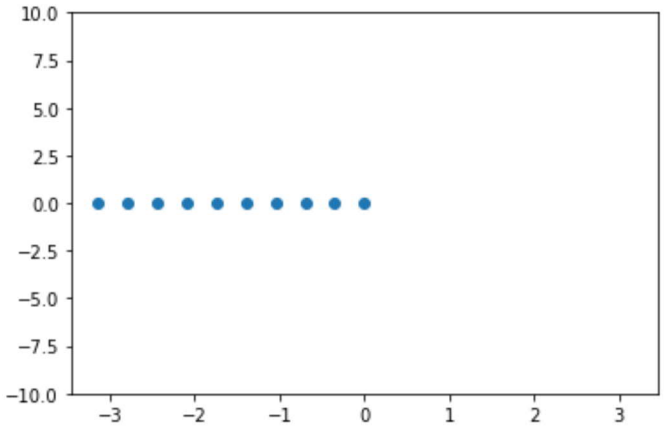
```
Synchtron frequency period: 225.59654471679005 turns
Bucket height:              7.1809610472257885 arbitrary units
```

Exercise 2: Generate an initial distribution

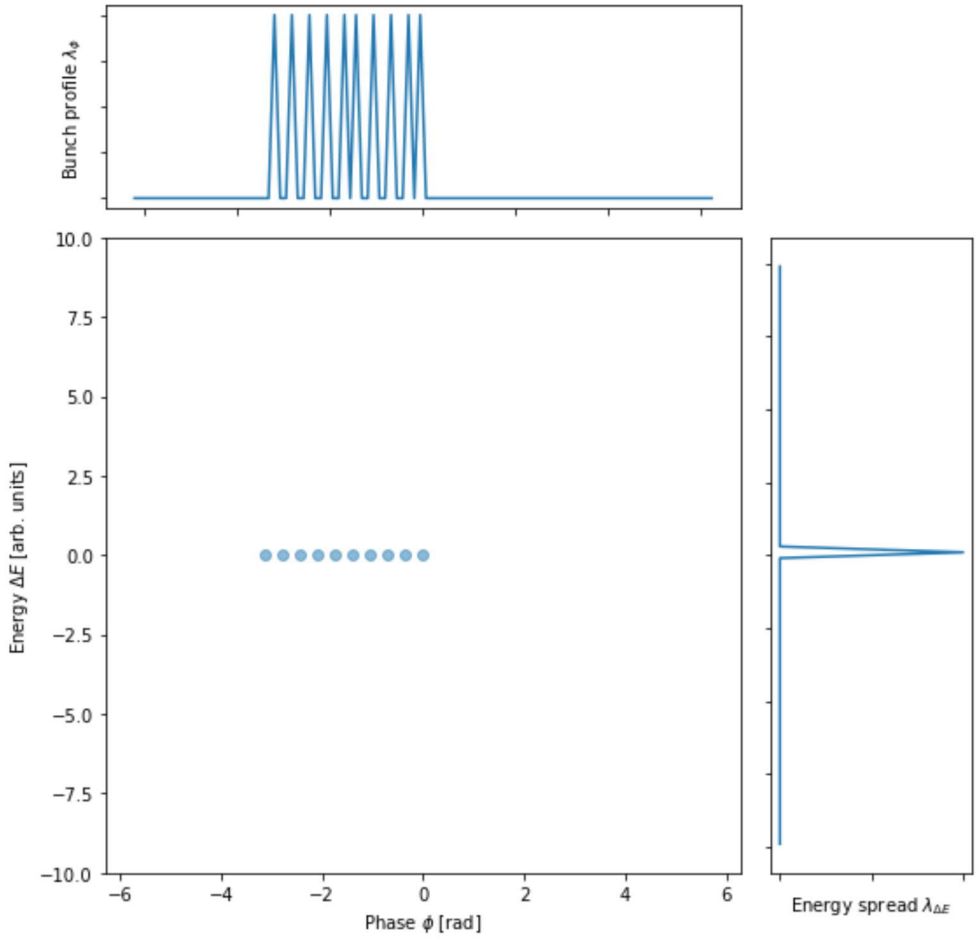
- Generate an ensemble of few test particles, e.g. with phase but no energy offset.
- Plot the longitudinal phase space using `plotPhaseSpace` from the support functions.

In [4]:
```
particles = np.linspace(-np.pi,0,10)
particles = np.array([particles, np.zeros(len(particles))])
```

In [5]:
```
%matplotlib inline
plt.scatter(*particles)
plt.xlim(-1.1*np.pi,1.1*np.pi)
plt.ylim(-10,10)
plt.show()
```



In [6]:
```
from support_functions import plotPhaseSpace
plotPhaseSpace(particles,
               xbins=100, ybins=100,
               xlim=(-2*np.pi, 2*np.pi), ylim=(-10, 10))
```



Exercise 3: Set-up tracking equations

- Set-up the tracking equations for the case of constant energy (no synchtrotron radiation).
- The function should take an ensemble of particles and apply the tracking equations for one turn.

Choice of variables: ϕ and ΔE, consistent with presentations

$$\phi_{n+1} = \phi_n - 2\pi h\eta\frac{\Delta E_n}{\beta^2 E_0},\ \eta = \frac{1}{\gamma^2} - \frac{1}{\gamma_{\mathrm{tr}}^2}$$
$$\Delta E_{n+1} = \Delta E_n + qVg(\phi_{n+1}) - U_0$$

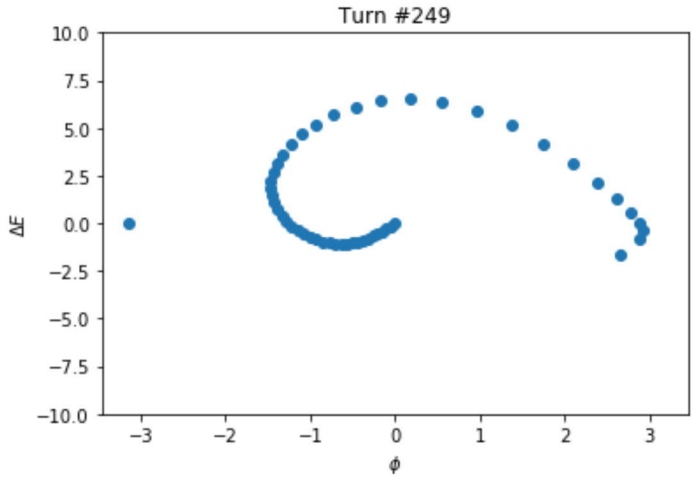
In [7]:
```
def trackOneTurn(particles):
    particles[0] = particles[0] - 2*np.pi*eta/beta**2*particles[1]/energy
    particles[1] = particles [1] + charge*voltage*np.sin(harmonic*particles[0])
```

In [8]:
```
def plotDistribution(particlePhase, particleEnergy, turnIndex=None):
    plt.close()
    plt.scatter(particlePhase, particleEnergy)
    if turnIndex!=None:
        plt.title("Turn #"+str(turnIndex))
    plt.xlim(-1.1*np.pi,1.1*np.pi)
    plt.ylim(-10, 10)
    plt.xlabel(r"$\phi$")
    plt.ylabel(r"$\Delta E$")
```

Exercise 4: Perform tracking of particles, check bucket parameters

- Track the initial ensemble of particles (from exercise 2) for about one period of the synchrotron frequency.
- Compare the bucket height and synchrotron frequency with the analytical calculation.

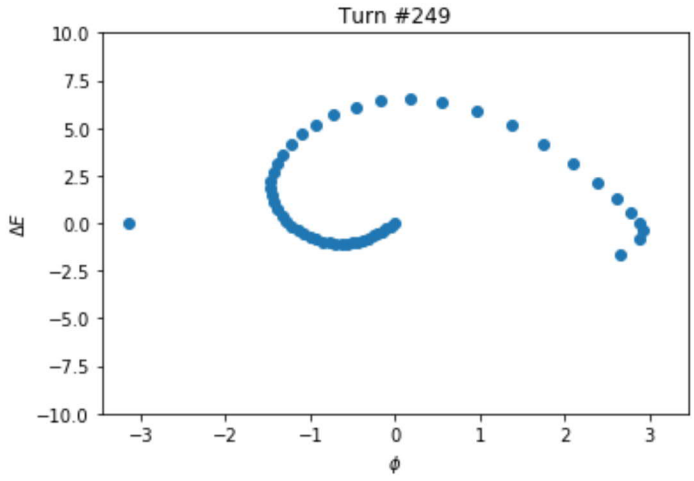
In [9]:
```
particles = particles = np.linspace(-np.pi,0,50)
particles = np.array([particles, np.zeros(len(particles))])
for turnIndex in range(250):
    trackOneTurn(particles)
plotDistribution(*particles, turnIndex=turnIndex)
```



Exercise 5: Accumulative tracking: follow trajectories and show phase space

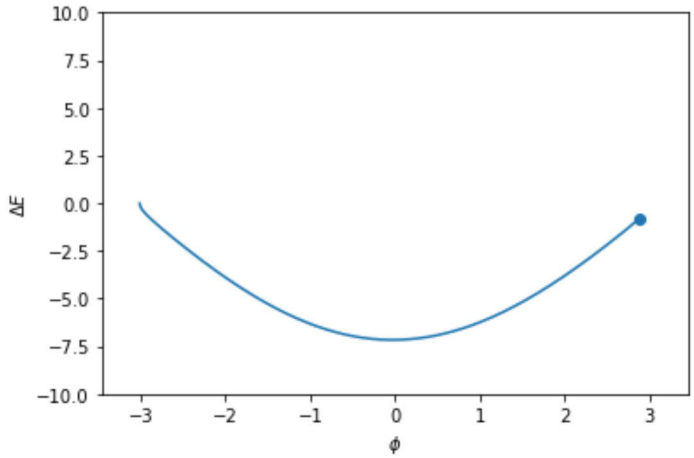
- Write simple functions to record and plot the trajectory of invidual particles.
- Track an ensemble of particle trajectories to visualise the bucket.

In [10]:
```
nTurns = 250
nParticles = 50
particles = np.linspace(-np.pi,0,nParticles)
particles = np.array([particles, np.zeros(len(particles))])
particlesAccumulated = np.zeros([nTurns, 2, nParticles])
for turnIndex in range(nTurns):
    trackOneTurn(particles)
    particlesAccumulated[turnIndex] = np.copy(particles)
plotDistribution(*particlesAccumulated[-1], turnIndex=turnIndex)
```

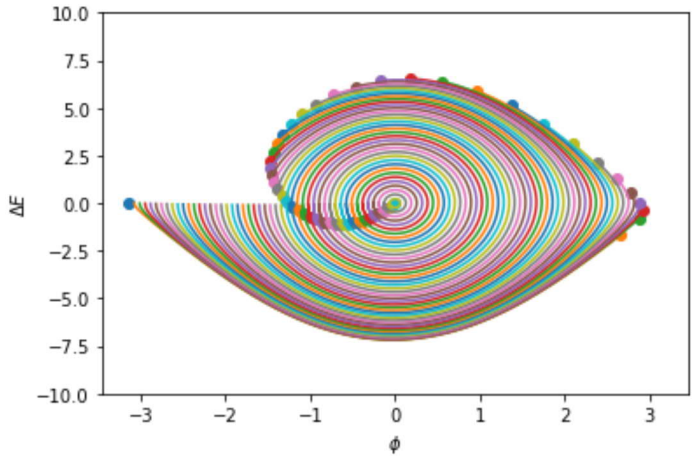


In [11]:
```
def plotTrack(particlePhase, particleEnergy):
    plt.plot(particlePhase, particleEnergy)
    plt.scatter(particlePhase[-1:],particleEnergy[-1:])
    plt.xlim(-1.1*np.pi,1.1*np.pi)
    plt.ylim(-10, 10)
    plt.xlabel(r"$\phi$")
    plt.ylabel(r"$\Delta E$")
```

In [12]:
```
plotTrack(*particlesAccumulated[:,:,2].T)
```



In [13]:
```
[plotTrack(*particleTrack) for particleTrack in particlesAccumulated.T]
plt.show()
```

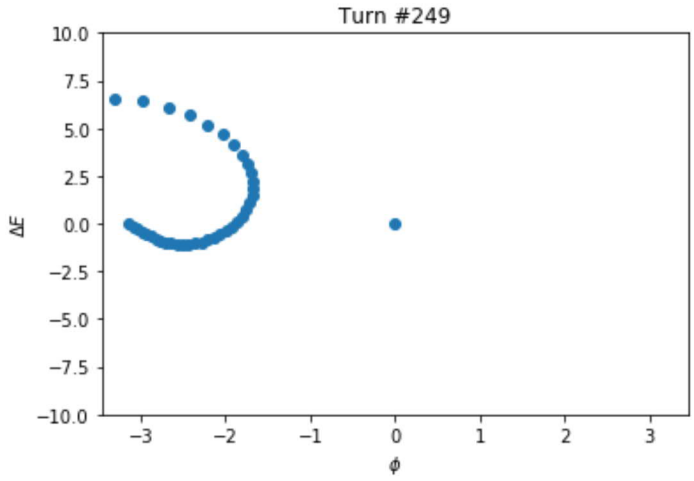


Exercise 6: On the other side of transition energy

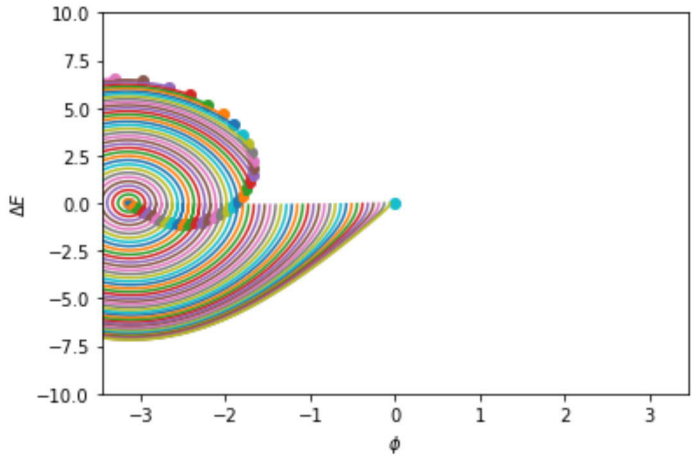
- Change the sign of the phase slip factor, η.
- What does that change for the bucket and why?

In [14]:
```
eta=-0.01
```

In [15]:
```
nTurns = 250
nParticles = 50
particles = np.linspace(-np.pi,0,nParticles)
particles = np.array([particles, np.zeros(len(particles))])
particlesAccumulated = np.zeros([nTurns, 2, nParticles])
for turnIndex in range(nTurns):
    trackOneTurn(particles)
    particlesAccumulated[turnIndex] = np.copy(particles)
plotDistribution(*particlesAccumulated[-1], turnIndex=turnIndex)
```



In [16]:
```
[plotTrack(*particleTrack) for particleTrack in particlesAccumulated.T]
plt.show()
```



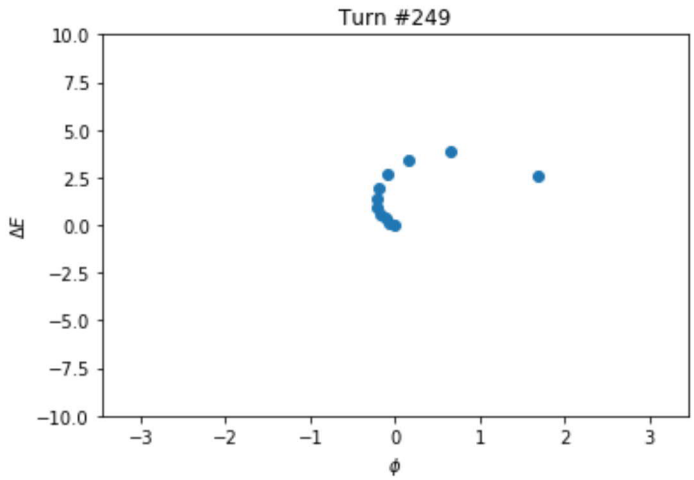
Exercise 7: Non-zero stable phase, acceleration and compensation of energy loss

- Add an energy gain or loss per turn to the tracking equations.
- Track some particles and observe the effect on the bucket.

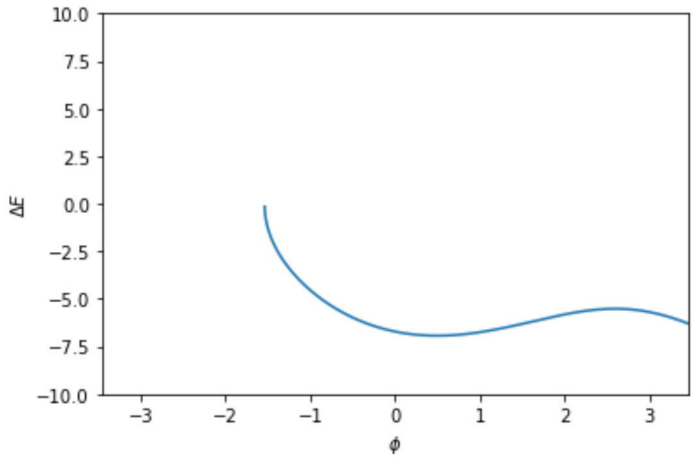
In [17]:
```
eta=0.01
stablePhase=30
```

In [18]:
```
def trackOneTurnEnergyChange(particles):
    particles[0] = particles[0] - 2*np.pi*eta/beta**2*particles[1]/energy
    particles[1] = particles [1] + charge*voltage*(np.sin(harmonic*particles[0])
- np.sin(stablePhase/180*np.pi))
```

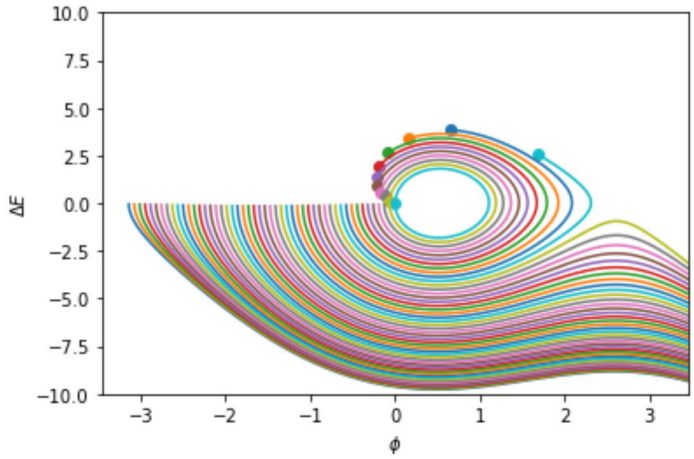
In [19]:
```
nTurns = 250
nParticles = 50
particles = np.linspace(-np.pi,0,nParticles)
particles = np.array([particles, np.zeros(len(particles))])
particlesAccumulated = np.zeros([nTurns, 2, nParticles])
for turnIndex in range(nTurns):
    trackOneTurnEnergyChange(particles)
    particlesAccumulated[turnIndex] = np.copy(particles)
plotDistribution(*particlesAccumulated[-1], turnIndex=turnIndex)
```



In [20]:
```
plotTrack(*particlesAccumulated[:,:,25].T)
```



In [21]:
```
[plotTrack(*particleTrack) for particleTrack in particlesAccumulated.T]
plt.show()
```



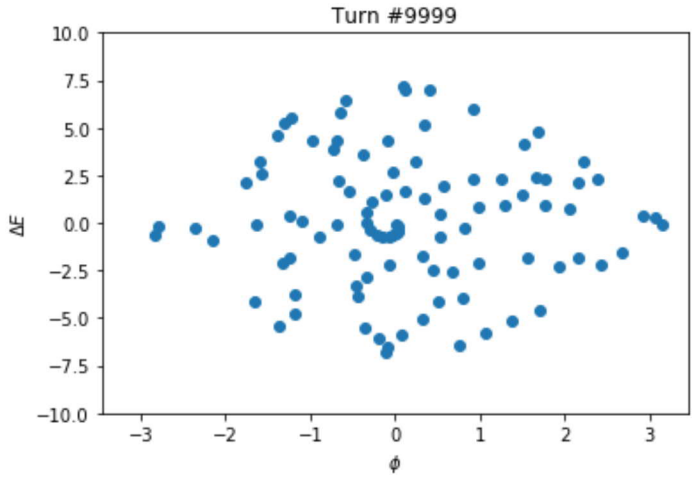
Exercise 8: Synchrotron frequency distribution, comparison with analytical formula

- Go back again the tracking equations for the stationary bucket (accumulative tracking).
- Track particles for a few synchrotron frequency periods to analyze the frequency of synchrotron oscillation.
- The function `oscillation_spectrum` returns the spectrum of phase or energy offset for a given particle.
- The function `synchrotron_tune` returns the frequency of the peak, the synchrotron tune, $Q_\mathrm{S} = f_\mathrm{S}/f_\mathrm{rev}$.
- Optionally plot the synchrotron frequency versus phase or energy offset. This illustrates the synchrotron frequency distribution.

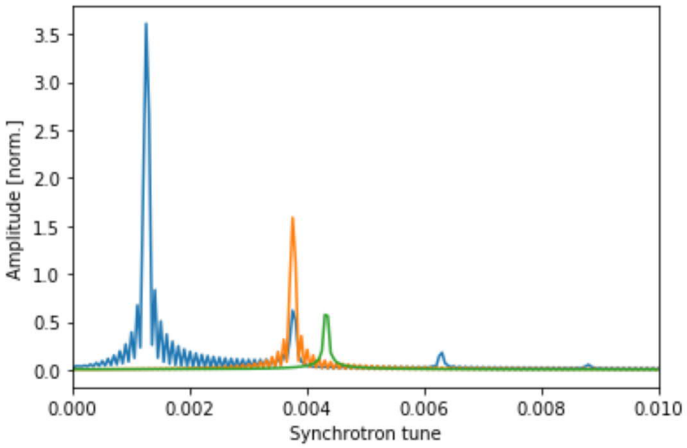
- Analytical approximation (stationary bucket): $\dfrac{\omega(\Delta\phi_\mathrm{u})}{\omega_\mathrm{S}} = \dfrac{\pi}{2K[\sin(\phi_\mathrm{u}/2)]} \simeq 1 - \dfrac{\phi_\mathrm{u}^2}{16}$

In [22]:
```
def trackOneTurn(particles):
    particles[0] = particles[0] - 2*np.pi*eta/beta**2*particles[1]/energy
    particles[1] = particles [1] + charge*voltage*np.sin(harmonic*particles[0])
```

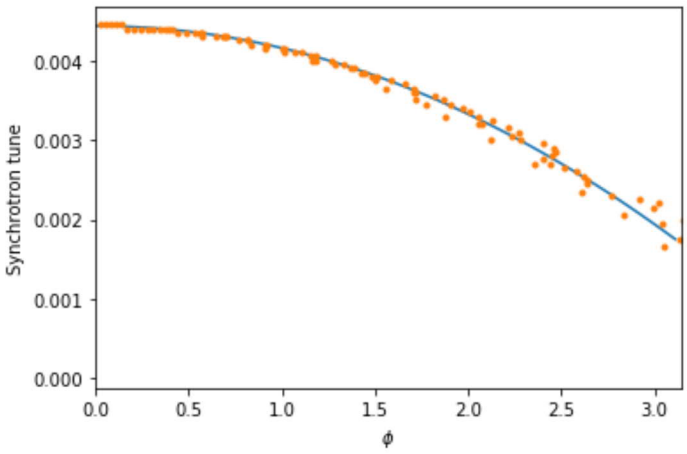
In [23]:
```
plt.show()
nTurns = 10000
nParticles = 100
particles = np.linspace(-np.pi,0,nParticles, endpoint=False)
particles = np.array([particles, np.zeros(len(particles))])
#particles = np.array([distribution[0], distribution[1]])
particlesAccumulated = np.zeros([nTurns, 2, nParticles])
for turnIndex in range(nTurns):
    trackOneTurn(particles)
    particlesAccumulated[turnIndex] = np.copy(particles)
plotDistribution(*particlesAccumulated[-1], turnIndex=turnIndex)
```



In [24]:
```
from support_functions import oscillation_spectrum

plt.clf
plt.plot(*oscillation_spectrum(particlesAccumulated[:,0,1], fft_zero_padding=100
00))
plt.plot(*oscillation_spectrum(particlesAccumulated[:,0,50], fft_zero_padding=10
000))
plt.plot(*oscillation_spectrum(particlesAccumulated[:,0,80], fft_zero_padding=10
000))
plt.xlim(0,0.01)
plt.xlabel("Synchrotron tune")
plt.ylabel("Amplitude [norm.]")
plt.show()
```



In [25]:
```
from support_functions import synchrotron_tune

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

synchrotronTuneDistribution = np.zeros([2, nParticles])
for particleIndex, particleTrack in enumerate(particlesAccumulated.T):
    synchrotronTuneDistribution[0, particleIndex], synchrotronTuneDistribution
[1, particleIndex] = \
        synchrotron_tune(particleTrack[0], fft_zero_padding=10000)

synchrotronTuneDistributionApproximation = np.linspace(0, np.pi, 100, endpoint=F
alse)
synchrotronTuneDistributionApproximation = np.array([synchrotronTuneDistribution
Approximation, \
                                                     stationaryBucketCentralSync
hrotronTune * \
                                                     (1-synchrotronTuneDistribut
ionApproximation**2/16)])

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



Exercise 9: Mismatched distribution, bunch rotation

- Generate bunch distribution using the `generateBunch` function.
- Try to match the bunch to the bucket. Track it for many periods of the synchrontron frequency.
- Introduce a phase or energy error between bunch and bucket.
- Introduce a voltage mismatch. What do you observe?

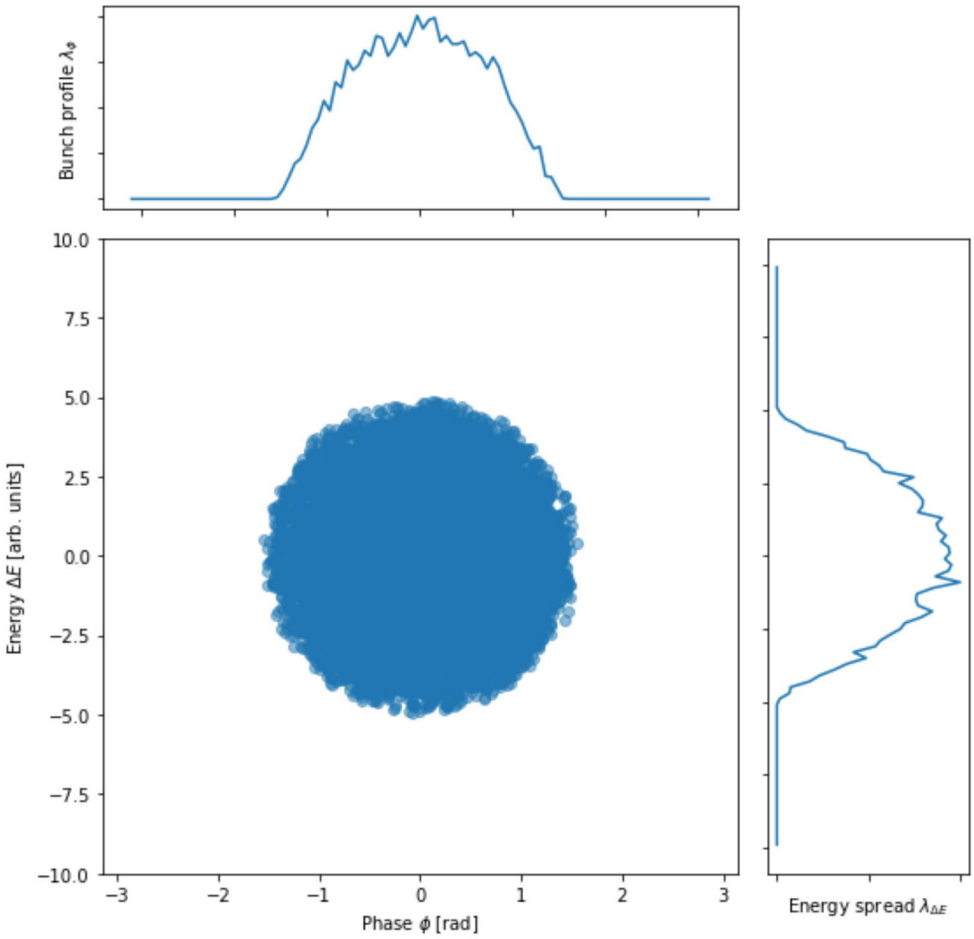
In [26]:
```
def trackOneTurn(particles):
    particles[0] = particles[0] - 2*np.pi*eta/beta**2*particles[1]/energy
    particles[1] = particles [1] + charge*voltage*np.sin(harmonic*particles[0])
```

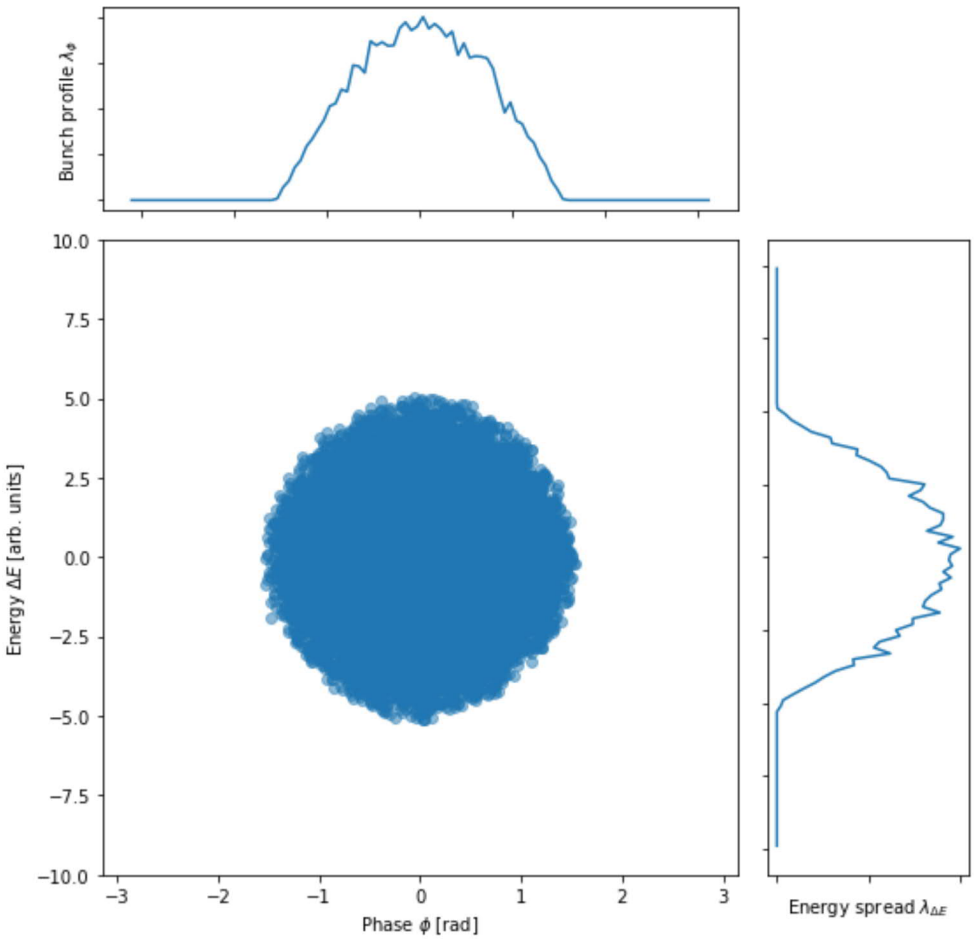
In [27]:
```
from support_functions import generateBunch

bunch_position = 0
bunch_length = np.pi

energy_position = 0
energy_spread = 10

n_macroparticles = 10000

distribution = generateBunch(
    bunch_position, bunch_length,
    energy_position, energy_spread,
    n_macroparticles)
```

Matched (almost)

In [28]:
```
particles = np.array([distribution[0], distribution[1]])
particlesAccumulated = [np.copy(particles)]
for turnIndex in range(10000):
    trackOneTurn(particles)
    particlesAccumulated.append(np.copy(particles))
```

In [29]:
```
plotPhaseSpace(particlesAccumulated[0],
               xbins=100, ybins=100,
               xlim=(-np.pi, np.pi), ylim=(-10, 10))
plotPhaseSpace(particlesAccumulated[-1],
               xbins=100, ybins=100,
               xlim=(-np.pi, np.pi), ylim=(-10, 10))
```

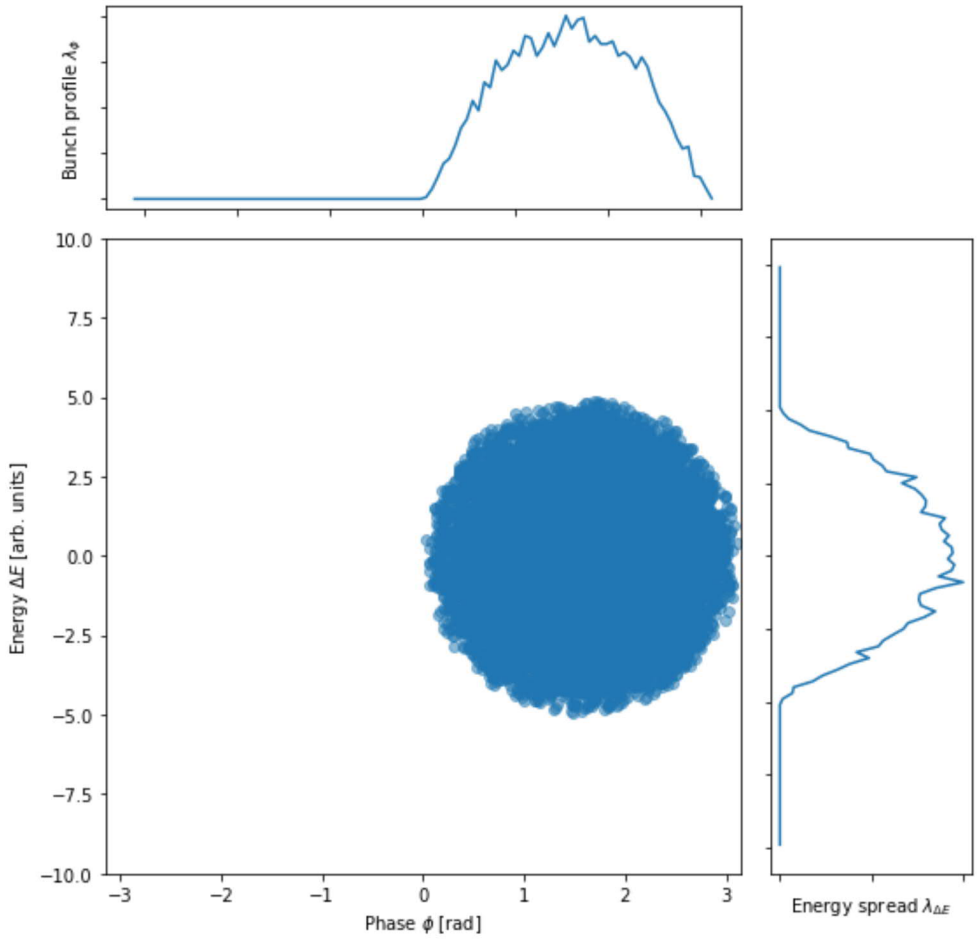

Bunch profile λ_ϕ
Energy ΔE [arb. units]
Phase ϕ [rad]
Energy spread $\lambda_{\Delta E}$

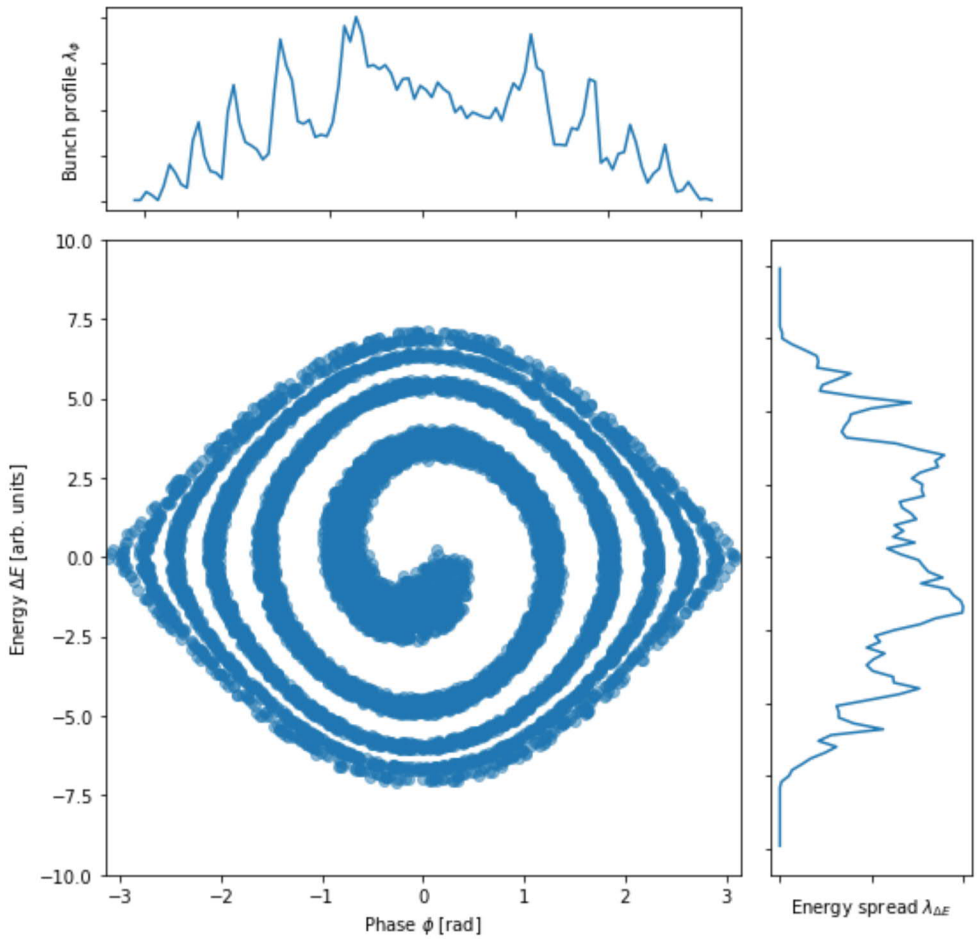

Bunch profile λ_ϕ
Energy ΔE [arb. units]
Phase ϕ [rad]
Energy spread $\lambda_{\Delta E}$

Phase error

In [30]:
```
phaseError = np.pi/2
particles = np.array([distribution[0] + phaseError, distribution[1]])
particlesAccumulated = [np.copy(particles)]
for turnIndex in range(2000):
    trackOneTurn(particles)
    particlesAccumulated.append(np.copy(particles))
```

In [31]:
```
plotPhaseSpace(particlesAccumulated[0],
               xbins=100, ybins=100,
               xlim=(-np.pi, np.pi), ylim=(-10, 10))
plotPhaseSpace(particlesAccumulated[-1],
               xbins=100, ybins=100,
               xlim=(-np.pi, np.pi), ylim=(-10, 10))
```

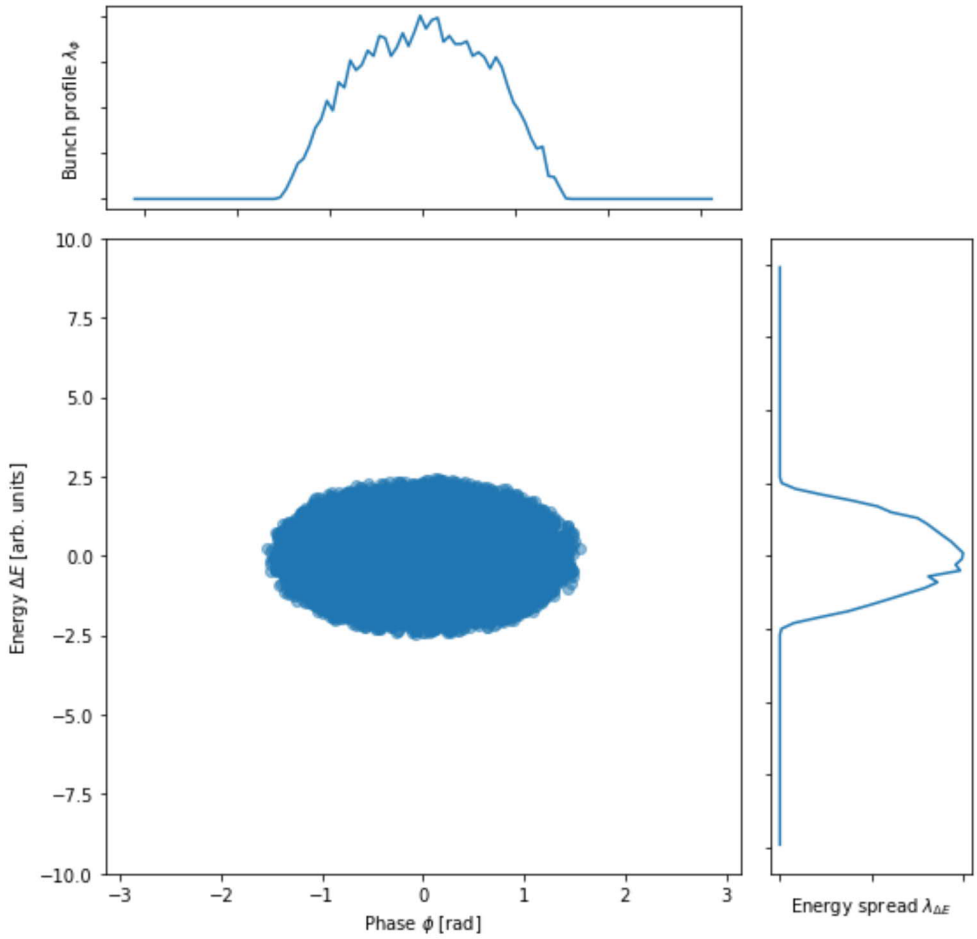

Bunch profile λ_ϕ
Energy ΔE [arb. units]
Phase ϕ [rad]
Energy spread $\lambda_{\Delta E}$

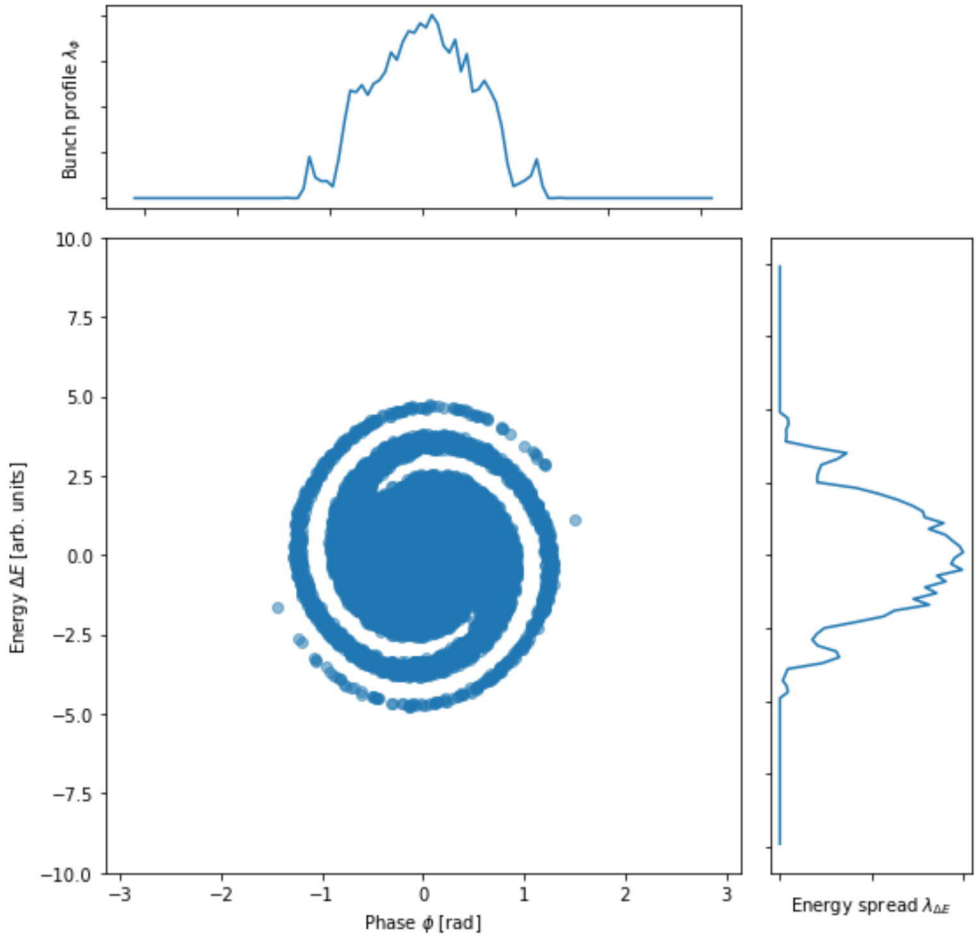

Bunch profile λ_ϕ
Energy ΔE [arb. units]
Phase ϕ [rad]
Energy spread $\lambda_{\Delta E}$

Mismatch

In [32]:
```
aspectRatioFactor = 0.5
particles = np.array([distribution[0], aspectRatioFactor*distribution[1]])
particlesAccumulated = [np.copy(particles)]
for turnIndex in range(2000):
    trackOneTurn(particles)
    particlesAccumulated.append(np.copy(particles))
```

In [33]:
```
plotPhaseSpace(particlesAccumulated[0],
               xbins=100, ybins=100,
               xlim=(-np.pi, np.pi), ylim=(-10, 10))
plotPhaseSpace(particlesAccumulated[-1],
               xbins=100, ybins=100,
               xlim=(-np.pi, np.pi), ylim=(-10, 10))
```


Bunch profile λ_ϕ
Energy ΔE [arb. units]
Phase ϕ [rad]
Energy spread $\lambda_{\Delta E}$

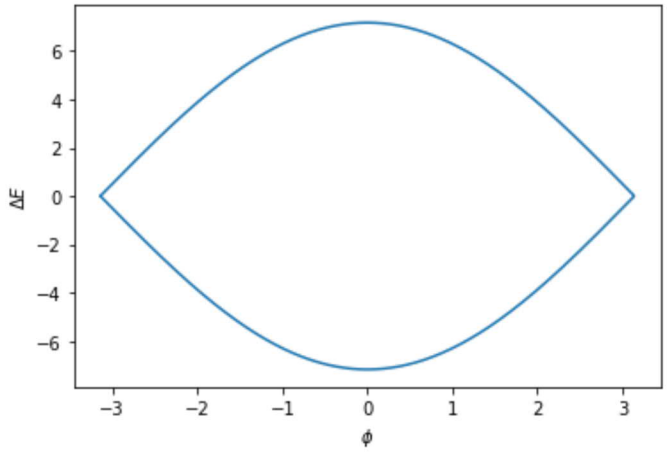

Bunch profile λ_ϕ
Energy ΔE [arb. units]
Phase ϕ [rad]
Energy spread $\lambda_{\Delta E}$

Exercise 10: Ideas for optional tracking simulation studies

- Add second harmonic RF system in the tracking
- What happens when operating both RF systems in phase (bunch-shortening) or in counterphase (bunch-lengthening)?
- Check the effect on the synchrotron frequency.

- Animate the evolution of the bunch distribution for the test cases of exercise 9.

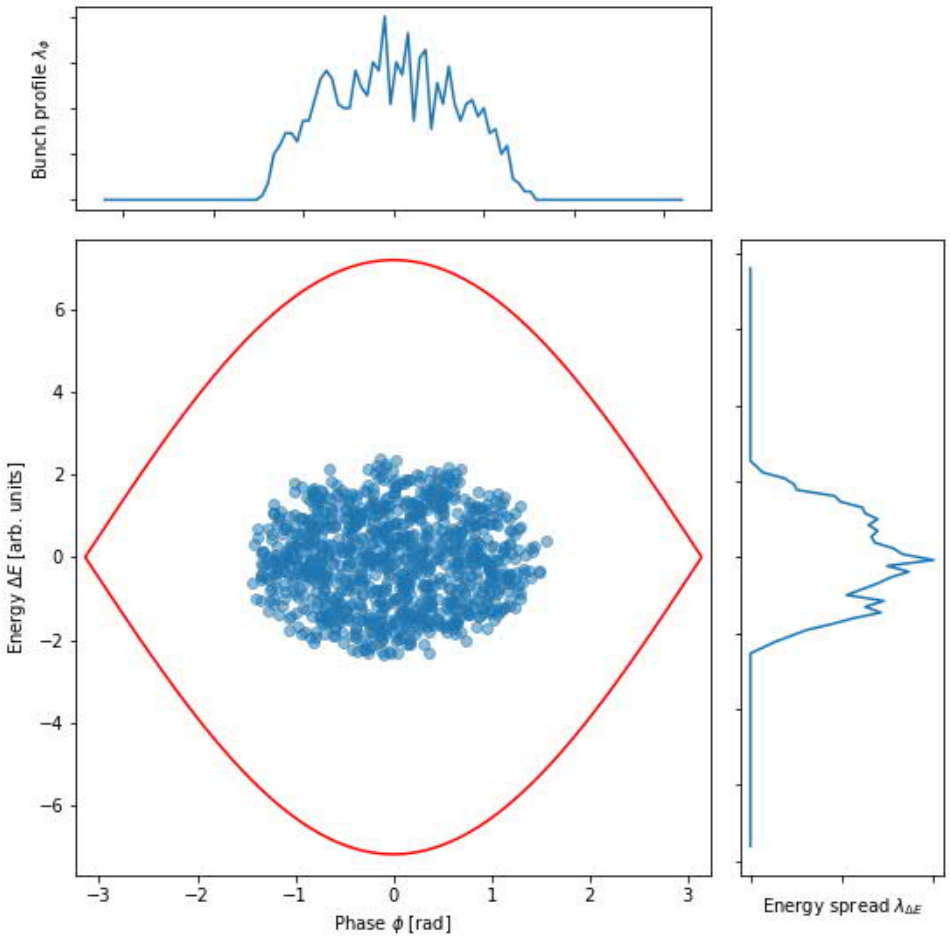
In [34]:
```
from support_functions import separatrix

stablePhase = 0
phase_array = np.linspace(-np.pi,np.pi,1000)
energy_gain = charge*voltage*np.sin(stablePhase/180*np.pi)
phase_sep, separatrix_array = separatrix(phase_array, eta, beta, energy, charge,
voltage, harmonic, energy_gain)
plt.plot(phase_sep, separatrix_array)
plt.xlabel(r"$\phi$")
plt.ylabel(r"$\Delta E$")
plt.show()
```

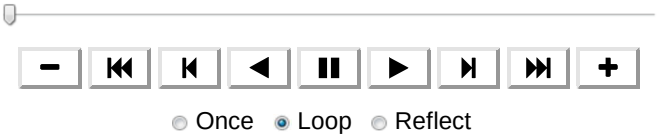


In [35]:
```
from support_functions import generateBunch

bunch_position = 0
bunch_length = np.pi

energy_position = 0
energy_spread = 0.5*10

n_macroparticles = 1000

distribution = generateBunch(
    bunch_position, bunch_length,
    energy_position, energy_spread,
    n_macroparticles)
```

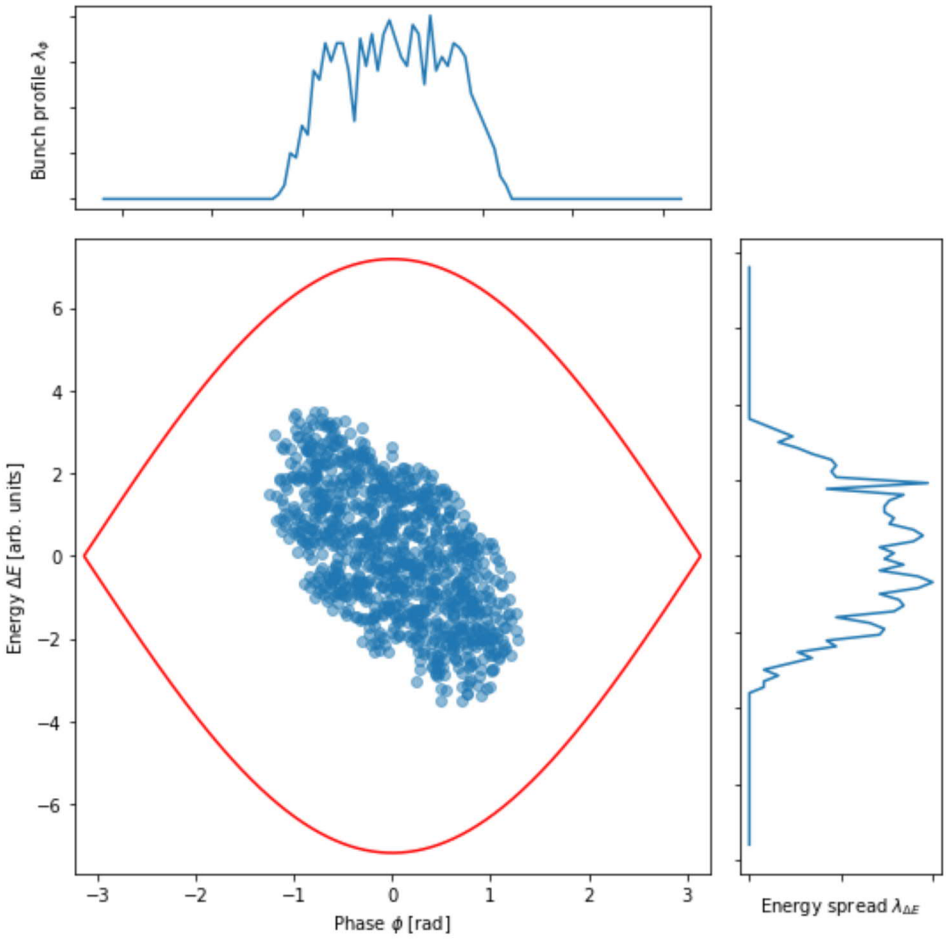
In [36]:
```
from support_functions import run_animation

particles = np.array([distribution[0], distribution[1]])

trackingFunction = trackOneTurnEnergyChange
figname = 'Animate'
iterations = 100
framerate = 30

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

run_animation(particles, trackingFunction, figname, iterations, framerate,
              phase_sep=phase_sep, separatrix_array=separatrix_array,
              xbins=100, ybins=100,
              xlim=(np.min(phase_sep)-0.1, np.max(phase_sep)+0.1),
              ylim=(np.min(separatrix_array)-0.5, np.max(separatrix_array)+0.5))
```

Out[36]:





Once Loop Reflect



Transverse course

- Guido's notebook for the CAS (https://cernbox.cern.ch/index.php/s/j7JCfPEonD5VLNG)
- Guido's guidelines for Denmark CAS (https://codimd.web.cern.ch/evH2Es22Qsag2PbZJjgQ_A)

In [ ]: