Basic design of RF systems

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 trasverse exercises in Slovakia (from Guido): https://codimd.web.cern.ch/s/HkfQy3YbB (https://codimd.web.cern.ch/s/HkfQy3YbB)

Second 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)`

Bucket area reduction ratio depdening on stable phase

- Bucket area reduction ratio depdening on stable phase
- Use approximation (S. Y. Lee book, p. 242): $\alpha(\phi_S) \simeq \dfrac{1-\sin\phi_S}{1+\sin\phi_S}$

Import modules

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

Optimizing the CERN-PS for LHC-type beams: acceleration at high frequency

Design an RF system optimized for injection into the PS from a $2.5\,\mathrm{GeV}$ Linac

(see, e.g. Fig. 3 of, [C. Carli, Chamonix 2010 (https://indico.cern.ch/event/67839/contributions/1231573/attachments/1022547/1455755/OtherScenarios.pdf)](https://indico.cern.ch/event/67839/contributions/1231573/attachments/1022547/1455755/OtherScenarios.pdf))

Guidelines:

- Higher injection energy: $E_{\mathrm{kin}} = 2.5\,\mathrm{GeV}$.
- Bunch spacing of $25\,\mathrm{ns}$.
- RF system optimized under these conditions.

Basic parameters of the Proton Synchrotron (PS) at CERN

Parameter	
Energy range	$E_{\mathrm{kin}} = 2.5\,\mathrm{GeV} \ldots 26\,\mathrm{GeV}$
Circumference	$2\pi R = 628\,\mathrm{m}$
Bending radius	$\rho = 70.079\,\mathrm{m}$
Transition gamma	$\gamma_{\mathrm{tr}} = 6.1$
Acceleration time	$1\,\mathrm{s}$
Longitudinal emittance per bunch	$\varepsilon_{\mathrm{l}} = 0.35\,\mathrm{eVs}$
Maximum bucket filling factor	$\varepsilon_{\mathrm{l}}/A_{\mathrm{bucket}} = 0.8$
Total beam intensity	$N = 2 \cdot 10^{13}\,\mathrm{protons}$

Exercise 1: Average energy gain and stable phase

- How much energy does the particle gain during each turn assuming a constant ramp rate?
- What would be the stable phase for an RF voltage of 200 kV?

In [2]:
```
injectionEnergy = 2.5E9 #eV
extractionEnergy = 26E9 #eV

c0 = sciCont.c
e0 = sciCont.e
protonMass = sciCont.physical_constants['proton mass energy equivalent in MeV
'][0]*1E6
protonCharge = 1 #e

momentumInjection = np.sqrt((protonMass + injectionEnergy)**2 - protonMass**2)
momentumExtraction = np.sqrt((protonMass + extractionEnergy)**2 - protonMass**2)

print("Momentum at injection:  "+str(momentumInjection/1E9) +" GeV/c")
print("Momentum at extraction: "+str(momentumExtraction/1E9) +" GeV/c")
```

```
Momentum at injection:  3.3077727259441514 GeV/c
Momentum at extraction: 26.921926904060935 GeV/c
```

In [3]:
```
circumference = 2*np.pi*100 #m
bendingRadius = 70.079      #m
accelerationDuration = 1    #s

magneticFieldInjection = momentumInjection / (c0*bendingRadius*protonCharge)
magneticFieldExtraction = momentumExtraction / (c0*bendingRadius*protonCharge)
averageBDot = (magneticFieldExtraction - magneticFieldInjection)/accelerationDur
ation

print("Average B-Dot: "+str(averageBDot)+" T/s")
```

```
Average B-Dot: 1.1239934891041807 T/s
```

In [4]:
```
averageEnergyGainPerTurn = 2*np.pi*protonCharge*circumference/(2*np.pi)*bendingR
adius*averageBDot
print("Average per turn energy gain: "+str(averageEnergyGainPerTurn)+" eV")
```

```
Average per turn energy gain: 49491.60748180557 eV
```

In [5]:
```
rfVoltage = 200E3 #V
stablePhase = np.arcsin(averageEnergyGainPerTurn/rfVoltage)
print("Stable phase angle: "+str(180/np.pi*stablePhase)+" deg")
```

```
Stable phase angle: 14.327142764919783 deg
```

Exercise 2: Power transfer to the beam

- How much power is transferred from the RF system to the beam?

In [6]:
```
nProtons = 1E13
energyGainJoules = (extractionEnergy - injectionEnergy)*e0*nProtons
averagePowerToBeam = energyGainJoules / accelerationDuration
print("Average power to beam: "+str(averagePowerToBeam/1E3)+" kW")
```

```
Average power to beam: 37.6511505888 kW
```

Exercise 3: RF frequency and harmonic

- Choose RF frequency and harmonic of the RF system.
- Note a few arguments for your choice.
- What is the frequency range of the RF system?

In [7]:
```
betaInjection = np.sqrt(1 - (protonMass/(protonMass + injectionEnergy))**2)
betaExtraction = np.sqrt(1 - (protonMass/(protonMass + extractionEnergy))**2)

print(f"beta injection: {betaInjection}, beta extraction: {betaExtraction}")
```

```
beta injection: 0.9620450760527051, beta extraction: 0.9993932358694079
```

In [8]:
```
revolutionFrequencyInjection = c0*betaInjection/circumference
revolutionFrequencyExtraction = c0*betaExtraction/circumference

print("Revolution frequency at injection:  "+str(revolutionFrequencyInjection/1E3)+ "kHz")
print("Revolution frequency at extraction: "+str(revolutionFrequencyExtraction/1E3)+ "kHz")
```

```
Revolution frequency at injection:  459.02491165918104kHz
Revolution frequency at extraction: 476.84500781396434kHz
```

Arguments for choice of RF frequency

- 25 ns bunch spacing suggests 40 MHz

Choice of harmonic number

In [9]:
```
flatTopRFFrequency = 1/25E-9
harmonicNumber = flatTopRFFrequency/revolutionFrequencyExtraction
print("40 MHz/revolution frequency at extraction: "+str(harmonicNumber))
harmonicNumber = int(round(harmonicNumber))
print("Nearest integer harmonic number, h = "+str(harmonicNumber))
```

```
40 MHz/revolution frequency at extraction: 83.88469910458944
Nearest integer harmonic number, h = 84
```

RF frequency swing

In [10]:
```
injectionRFFrequency = harmonicNumber * revolutionFrequencyInjection
extractionRFFrequency = harmonicNumber * revolutionFrequencyExtraction

frequencySwingFRev = revolutionFrequencyExtraction - revolutionFrequencyInjection
frequencySwingRF = extractionRFFrequency - injectionRFFrequency

print("Revolution frequency swing: "+str(frequencySwingFRev/1E3)+" kHz ("+str(100*frequencySwingFRev/revolutionFrequencyInjection)+" %)")
print("RF frequency swing: "+str(frequencySwingRF/1E6)+" MHz")
```

```
Revolution frequency swing: 17.820096154783272 kHz (3.8821631903094658 %)
RF frequency swing: 1.4968880770017952 MHz
```

Exercise 4: Calculate bucket area during the cycle, determine RF voltage along the cycle

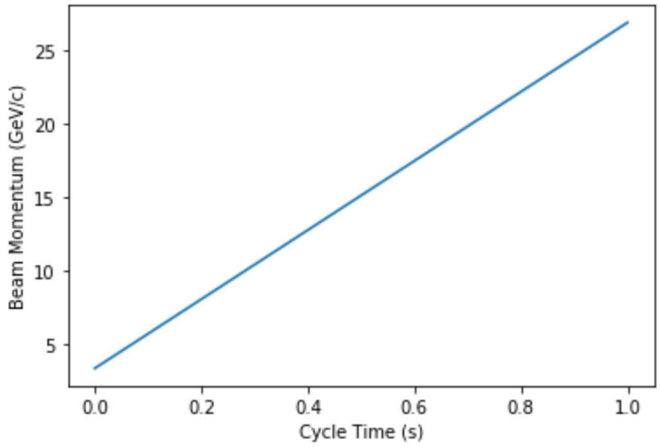
- Plot momentum, kinetic energy and revolution frequency (and/or further parameters) during the cycle.
- Calculate the bucket area along the cycle and chose an RF voltage such that a bunch with $0.35\,\mathrm{eV}$ longitudinal emittance can be comfortably accelerated, e.g $\varepsilon_\mathrm{l}/A_\mathrm{bucket} \simeq 0.8$.

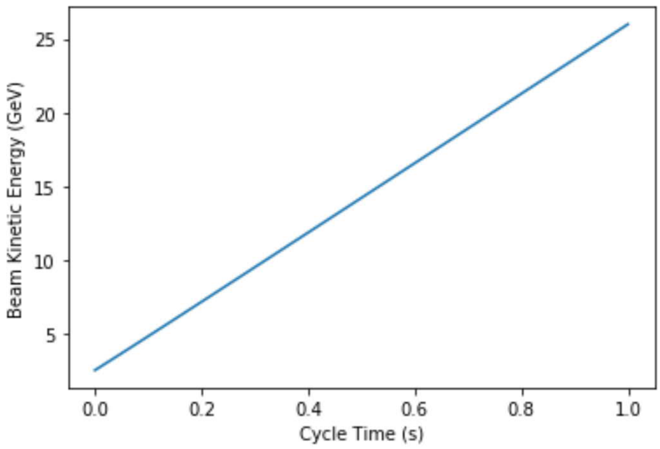
In [11]:
```
longitudinalEmittance = 0.35 #eVs
targetFillingFactor = 0.8
targetBucketArea = longitudinalEmittance / targetFillingFactor

print("Target bucket area: "+str(targetBucketArea)+" eVs")
```

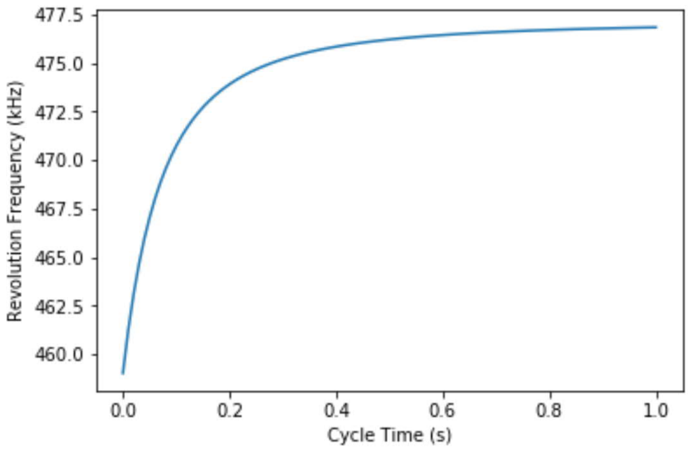
```
Target bucket area: 0.43749999999999994 eVs
```

Momentum, energy and revolution frequency during cycle

In [12]:
```
import matplotlib.pyplot as plt

timeRange = np.linspace(0, 1, 100)
BFieldRange = np.linspace(magneticFieldInjection, magneticFieldExtraction, len(t
imeRange))
momentumRange = protonCharge*bendingRadius*BFieldRange*c0

plt.plot(timeRange, momentumRange/1e9)
plt.xlabel("Cycle Time (s)")
plt.ylabel("Beam Momentum (GeV/c)")
plt.show()

energyRange = np.sqrt(momentumRange**2 + protonMass**2)

plt.plot(timeRange, (energyRange - protonMass)/1E9)
plt.xlabel("Cycle Time (s)")
plt.ylabel("Beam Kinetic Energy (GeV)")
plt.show()

betaRange = momentumRange/energyRange
gammaRange = 1/np.sqrt(1-betaRange**2)

fRevRange = c0*betaRange/circumference

plt.plot(timeRange, fRevRange/1e3)
plt.xlabel("Cycle Time (s)")
plt.ylabel("Revolution Frequency (kHz)")
plt.show()
```

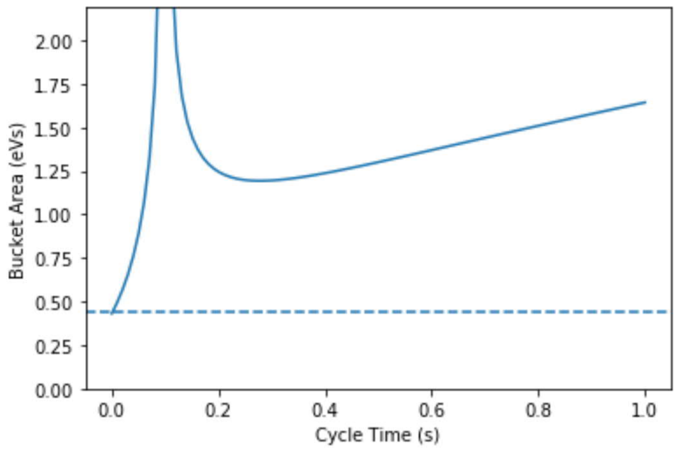

Beam Momentum (GeV/c)
Cycle Time (s)


Beam Kinetic Energy (GeV)
Cycle Time (s)


Revolution Frequency (kHz)
Cycle Time (s)

Bucket area

In [13]:
```
from support_functions import reduction_ratio

rfHarmonic = 84
rfVoltage = 500E3 #V
gamma_T = 6.1

timeRange = np.linspace(0, 1, 100)
BFieldRange = np.linspace(magneticFieldInjection, magneticFieldExtraction, len(t
imeRange))
momentumRange = protonCharge*bendingRadius*BFieldRange*c0
energyRange = np.sqrt(momentumRange**2 + protonMass**2)
gammaRange = 1/np.sqrt(1-betaRange**2)
phaseSlipFactor = 1/gamma_T**2 - 1/gammaRange**2

reductionFactor = reduction_ratio(averageEnergyGainPerTurn, rfVoltage)

bucketAreaRange = 4/c0*circumference*np.sqrt((2*energyRange*rfVoltage)/ \
                                             (np.pi**3*rfHarmonic**3*np.abs(phas
eSlipFactor))) \
                      *reductionFactor

plt.plot(timeRange, bucketAreaRange)
plt.axhline(targetBucketArea, linestyle = '--')
plt.ylim(0,5*targetBucketArea)
plt.xlabel("Cycle Time (s)")
plt.ylabel("Bucket Area (eVs)")
plt.show()
```



Exercise 5: RF cavity, number of cavities, RF amplifier power

- Choose an approriate type of RF cavity.
- How many cavities would you install?
- Please note some arguments for the discussion.

- Frequeny range of about 4 % would require a fixed frequency cavities with a quality factor of only 26 $\rightarrow$ too low
- Need tuned RF cavities $\rightarrow$ partially filled with ferrite material
- Bias ferrite material to tilt hysteresis curve $\rightarrow$ change effective μ
- Cavity could be similar to the one for PS2 or Fermilab booster, but with reduced frequency range
- Single cavity sufficient for 100 kV at 40 MHz (zero beam current)
- Large RF system: would require about 5 cavities (or more)

Exercise 6: Requirements for beam loading

- What is the beam induced voltage and power?
- Under which circumstances do you really need that power?

In [14]:
```
RUponQ = 100
VInduced = nProtons*e0*RUponQ*harmonicNumber*revolutionFrequencyExtraction*2*np.
pi

print("Beam loading induced voltage (flat top): "+str(VInduced/1E3)+" kV")
```

```
Beam loading induced voltage (flat top): 40.32243818979762 kV
```

In [15]:
```
beamCurrent = e0*nProtons*revolutionFrequencyExtraction
beamLoadingPower = VInduced*beamCurrent

print(f"Beam loading power (flat top): "+str(beamLoadingPower/1E3)+" kW")
```

```
Beam loading power (flat top): 30.805936458470182 kW
```

- This additional power would be needed to fully compensate beam loading and operate the cavity at any phase

Exercise 7: Comparison with RF systems at Fermilab Booster

- Compare the parameters of your RF system with the one of the Booster synchrotron at Fermilab.

Parameter	Unit	PS (with 40 MHz RF)	Fermilab Booster
Beam energy, E_{kin}	GeV	2 to 26	0.4 to 8
Circumference, $2\pi R$	m	628.3	467.9
Bending radius, ρ	m	70.08	43.7
Acceleration time	s	1	0.033
RF harmonic, h		84	84
RF frequency, f_{RF}	MHz	38.56 to 40.05	37.86 to 52.81
RF voltage, V_{RF}	kV	500	860
Number of cavities		~5	17
Longiduinal emittance, ε_{l}	eVs	0.35	0.25

Design of an RF system upgrade for an electron storage ring

Design an RF system to run the Soleil electron storage ringe at higher energy and beam current

Guidelines:

- Higher energy: $2.75\,\mathrm{GeV}$ instead of ($3.5\,\mathrm{GeV}$).
- Higher beam current: $800\,\mathrm{mA}$ instead of $500\,\mathrm{mA}$.
- Bunch spacing of $25\,\mathrm{ns}$.
- Design the new RF system which can work in combination with the existing one.

Basic parameters of the Soleil electron storage ring ([parameter table (https://www.synchrotron-soleil.fr/en/research/sources-and-accelerators/parameters-accelerators-storage-ring)](https://www.synchrotron-soleil.fr/en/research/sources-and-accelerators/parameters-accelerators-storage-ring))

Parameter	
Beam energy	$E = 2.75\,\mathrm{GeV} \rightarrow 3.5\,\mathrm{GeV}$
Beam current	$I_\mathrm{b} = 500\,\mathrm{mA} \rightarrow 800\,\mathrm{mA}$
Circumference	$2\pi R = 354.097\,\mathrm{m}$
Bending radius	$\rho = 5.36\,\mathrm{m}$
Momentum compaction factor	$\alpha = 1/\gamma_\mathrm{tr}^2 - 1/\gamma^2 = 4.16 \cdot 10^{-4}$
Harmonic of RF system	$h = 416$
RF frequency	$f_\mathrm{RF} = 352.2\,\mathrm{MHz}$

Exercise 1: Average energy loss per turn

- Calculate the average energy loss per turn to be restituted before and after the upgrade.
- Plot the energy loss versus beam energy.

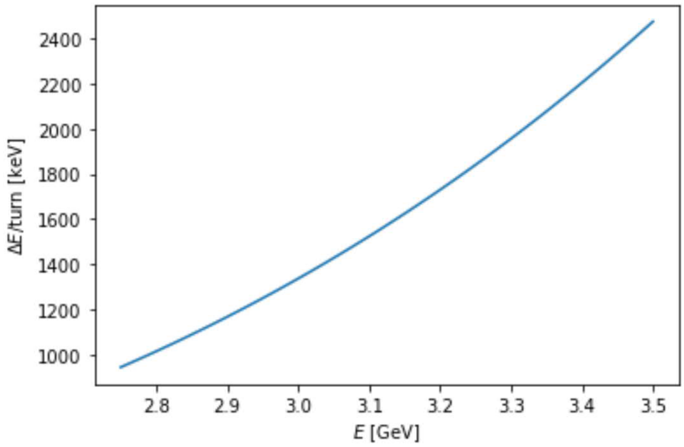
In [16]:
```
electronMass = sciCont.physical_constants['electron mass energy equivalent in MeV'][0]*1E6 #eV
electronCharge = 1 #e
bendingRadius = 5.36 #m
epsilon0 = sciCont.epsilon_0 #F/m
c0 = sciCont.c #m/s
e0 = sciCont.e #C
beamEnergyBefore = 2.75E9 #eV

energyLossPerTurnBefore = e0**2*(beamEnergyBefore/electronMass)**4/(3*epsilon0*bendingRadius)/e0

print("Energy loss per turn before upgrade: "+str(energyLossPerTurnBefore/1E3)+" keV")
```

```
Energy loss per turn before upgrade: 943.900841372113 keV
```

In [17]:
```
beamEnergyAfter = 3.5E9 #eV

energyLossPerTurnAfter = e0**2*(beamEnergyAfter/electronMass)**4/(3*epsilon0*bendingRadius)/e0

print("Energy loss per turn after upgrade: "+str(energyLossPerTurnAfter/1E3)+" keV")
```

```
Energy loss per turn after upgrade: 2476.667899880548 keV
```

In [18]:
```
beamEnergyRange = np.linspace(beamEnergyBefore, beamEnergyAfter, 100)
energyLossPerTurn = e0**2*(beamEnergyRange/electronMass)**4/(3*epsilon0*bendingR
adius)/e0

plt.plot(beamEnergyRange/1E9, energyLossPerTurn/1E3)
plt.xlabel("$E$ [GeV]")
plt.ylabel("$\Delta E/$turn [keV]")
plt.show()
```



Exercise 2: Average RF power

- What is the average power to the beam before and after the upgrade?
- Plot the required RF power versus beam energy.
- Why should the installed RF power pratically be higher?

In [19]:
```
#Electrons therefore beta = 1
circumference = 354.097
revolutionFrequency = c0/circumference

print("Revolution frequency: "+str(revolutionFrequency/1E3)+" kHz")
```

```
Revolution frequency: 846.63936153088 kHz
```

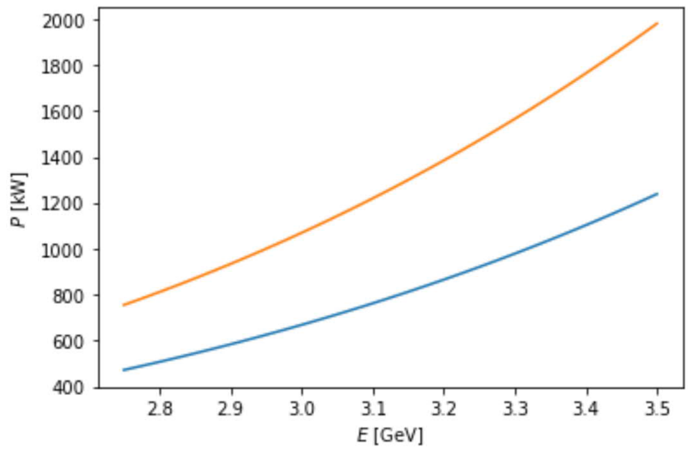
In [20]:
```
beamCurrent = 500E-3 #A
nElectrons = (beamCurrent/e0)/revolutionFrequency
radiationPowerBefore = energyLossPerTurnBefore*revolutionFrequency*nElectrons*e0

print(f"Average power to beam before upgrade: "+str(radiationPowerBefore/1E3)+"
kW")
```

```
Average power to beam before upgrade: 471.9504206860564 kW
```

In [21]:
```
beamCurrent = 800E-3 #A
nElectrons = (beamCurrent/e0)/revolutionFrequency
radiationPowerAfter = energyLossPerTurnAfter*revolutionFrequency*nElectrons*e0

print(f"Average power to beam before upgrade: "+str(radiationPowerAfter/1E3)+" k
W")
```

```
Average power to beam before upgrade: 1981.3343199044384 kW
```

In [22]:
```
beamEnergyRange = np.linspace(beamEnergyBefore, beamEnergyAfter, 100)
energyLossPerTurn = e0**2*(beamEnergyRange/electronMass)**4/(3*epsilon0*bendingRadius)/e0

beamCurrent = 500E-3
radiationPowerBeforeCurrent = energyLossPerTurn*revolutionFrequency*beamCurrent/revolutionFrequency

beamCurrent = 800E-3
radiationPowerAfterCurrent = energyLossPerTurn*revolutionFrequency*beamCurrent/revolutionFrequency

plt.plot(beamEnergyRange/1E9, radiationPowerBeforeCurrent/1E3)
plt.plot(beamEnergyRange/1E9, radiationPowerAfterCurrent/1E3)
plt.xlabel("$E$ [GeV]")
plt.ylabel("$P$ [kW]")
plt.show()
```



Exercise 3: Chose RF frequency. Arguments?

- Chose the RF frequency and harmonic of the additional RF system.
- Note a few arguments supporting your choice.

In [23]:
```
minimumFrequency = 1/25E-9
minimumHarmonic = minimumFrequency/revolutionFrequency
print("Minimum frequency: "+str(minimumFrequency/1E6)+" MHz")
print("40 MHz/revolution frequency at extraction: "+str(minimumHarmonic))
```

```
Minimum frequency: 40.0 MHz
40 MHz/revolution frequency at extraction: 47.24561816695202
```

In [24]:
```
chosenHarmonic = 2*416
chosenFrequency = chosenHarmonic*revolutionFrequency
bucketSpacing = 1/chosenFrequency

print("Bucket spacing: "+str(bucketSpacing*1E9)+" ns")
```

```
Bucket spacing: 1.419639968958895 ns
```

- Must be integer harmonic of existing RF system ($h = 416$)
- Chose twice that frequency ($h = 832$, $f_{\mathrm{RF}} = 704$ MHz) to generate additional voltage more easily and with compact cavities

Exercise 4: RF cavity, number of cavities, RF amplifier power

- Choose an approriate type of RF cavity.
- How many cavities would you install?
- Please note some arguments for the discussion.

- Chose frequency above 40 MHz to generate 1.5 MV in addition
- 40 MHz would be too low to efficiently obtain high voltage
  - RF frequency unnecessarily low
  - Cavities would be too large
- Chose multiple of the existing RF system at $h = 416$

- Vacuum resonator at fixed frequency
- Bell-shape: avoid multipactor and higher order modes
- Moderate R/Q

Exercise 5: Requirements for beam loading: beam induced voltage and power

- Calculate the beam induced voltage and power in the additional cavity.
- How does the power compare to the power lost by synchrotron radiation?
- Under which circumstances do you really need that power?

In [25]:
```
RUponQ = 44
VInduced = nElectrons*e0*RUponQ*chosenHarmonic*revolutionFrequency*2*np.pi

print("Beam loading induced voltage: "+str(VInduced/1E3)+" kV")
```

```
Beam loading induced voltage: 184.01187818018423 kV
```

In [26]:
```
beamCurrent = 800E-3 #A
beamLoadingPower = VInduced*beamCurrent

print(f"Beam loading power (flat top): "+str(beamLoadingPower/1E3)+" kW")
```

```
Beam loading power (flat top): 147.2095025441474 kW
```

- This would only be needed to fully compensate beam loading
- Detune RF cavities such that the system of power generator and beam is resonant at $832 f_{\mathrm{rev}}$

Exercise 6: Beam life time with no RF

- How many turns would the beam survive without RF? For a first estimate one can assume a constant energy loss. The momentum acceptance is on the order of 0.5%.
- Optionally: take into account the energy loss per turn changes with beam energy.
- Plot the beam energy versus number of turns.

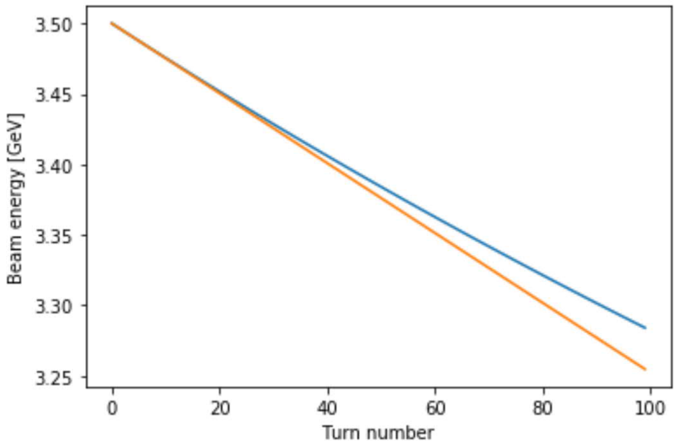
In [27]:
```
momentumRatioAcceptance = 0.5E-2
beamEnergy = 3.5E9 #eV
designMomentum = np.sqrt(beamEnergy**2 - electronMass**2)
lossMomentum = designMomentum*(1 - momentumRatioAcceptance)
lossEnergy = np.sqrt(lossMomentum**2 + electronMass**2)

print("Momentum at which particles are lost: "+str(lossMomentum/1E9)+" GeV/c")
print("Energy at which particles are lost:   "+str(lossEnergy/1E9)+" GeV")
```

```
Momentum at which particles are lost: 3.482499962883668 GeV/c
Energy at which particles are lost:   3.4825000003739657 GeV
```

In [28]:
```
nTurnsUntilLost = (beamEnergy - lossEnergy)/energyLossPerTurnBefore
lifeTime = nTurnsUntilLost/revolutionFrequency

print("Life time (before upgrade): "+str(lifeTime*1E6)+" us ("+str(nTurnsUntilLo
st)+" turns)")

nTurnsUntilLost = (beamEnergy - lossEnergy)/energyLossPerTurnAfter
lifeTime = nTurnsUntilLost/revolutionFrequency

print("Life time (after upgrade):  "+str(lifeTime*1E6)+" us ("+str(nTurnsUntilLo
st)+" turns)")
```

```
Life time (before upgrade): 21.898441658647435 us (18.54008266439849 turns)
Life time (after upgrade):  8.345873706899651 us (7.065945186626879 turns)
```

In [29]:
```
beamEnergyInitial = 3.5E9
nTurns = 100
turnRange = np.array(range(nTurns))
energyTurnByTurn = np.zeros(nTurns)
energyLossPerTurnInitial = e0**2*(beamEnergyInitial/electronMass)**4/(3*epsilon
0*bendingRadius)/e0

beamEnergy = beamEnergyInitial
for turn in turnRange:
    energyTurnByTurn[turn] = beamEnergy
    energyLossPerTurn = e0**2*(beamEnergy/electronMass)**4/(3*epsilon0*bendingRa
dius)/e0
    beamEnergy = beamEnergy - energyLossPerTurn

plt.plot(turnRange, energyTurnByTurn/1E9)
plt.plot(turnRange, (beamEnergyInitial - turnRange*energyLossPerTurnInitial)/1E
9)
plt.xlabel("Turn number")
plt.ylabel("Beam energy [GeV]")
plt.show()
```



Exercise 7: Radiation damping time

- Caluclate the damping times of the synchrotron oscillations before and after the upgrade.

In [30]:
```
revolutionTime = 1/revolutionFrequency
momentumCompationFactor = 4.16E-4
dampingIntegralD = momentumCompationFactor * circumference/(2*np.pi*bendingRadiu
s)

print("D = "+str(dampingIntegralD)+" (well below one, 2+D can be approximate to
2)")

dampingTimeBefore = beamEnergyBefore*revolutionTime/energyLossPerTurnBefore  #no
t radiationPowerBefore (which is for the whole beam)
dampingTimeAfter  = beamEnergyBefore*revolutionTime/energyLossPerTurnAfter   #no
t radiationPowerAfter

print("SR Damping time before: "+str(dampingTimeBefore*1E3)+" ms")
print("SR Damping time after:  "+str(dampingTimeAfter*1E3)+" ms")
```

```
D = 0.004373920850699349 (well below one, 2+D can be approximate to 2)
SR Damping time before: 3.4411837627523014 ms
SR Damping time after:  1.311494467681602 ms
```

Exercise 8: Comparison with RF system at ESRF

- Compare the parameters of the (additional) RF system with the one of the storage ring at ESRF.

Parameter	Unit	ESRF	Soleil	Soleil (CAS upgrade)
Beam energy, E	GeV	6	2.75	3.5
Beam current, I_{beam}	mA	200	500	800
Circumference, $2\pi R$	m	844	354	354
Bending radius, ρ	m	23.37	5.36	5.36
RF harmonic, h		992	416	416 amd 832
RF frequency, f_{RF}	MHz	352	352	352 and 704
RF voltage, V_{RF}	MV	6.5	3	3 + >1.5
Number of cavities		14	4	4 + ~4

Transverse course

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