Exercise 14

Using the 5 FODO cells of Exercise 9, trasport the beam of Exercise 13 and plot its rms size and divergence along the line.

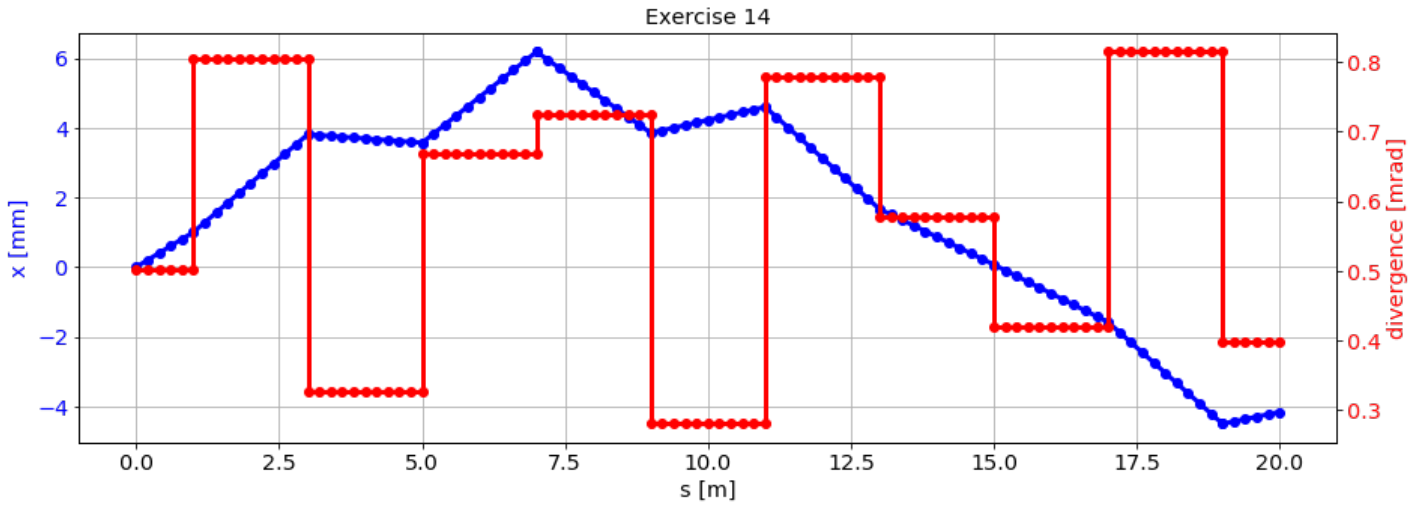
SOLUTION

In [1]:

```
# Import the modules
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sympy as sy
from matplotlib import pyplot as plt
%matplotlib inline

D = lambda L: [(np.array([[1, L],[0, 1]]), L)]
Q = lambda f: [(np.array([[1, 0],[-1/f, 1]]), 0)]
```

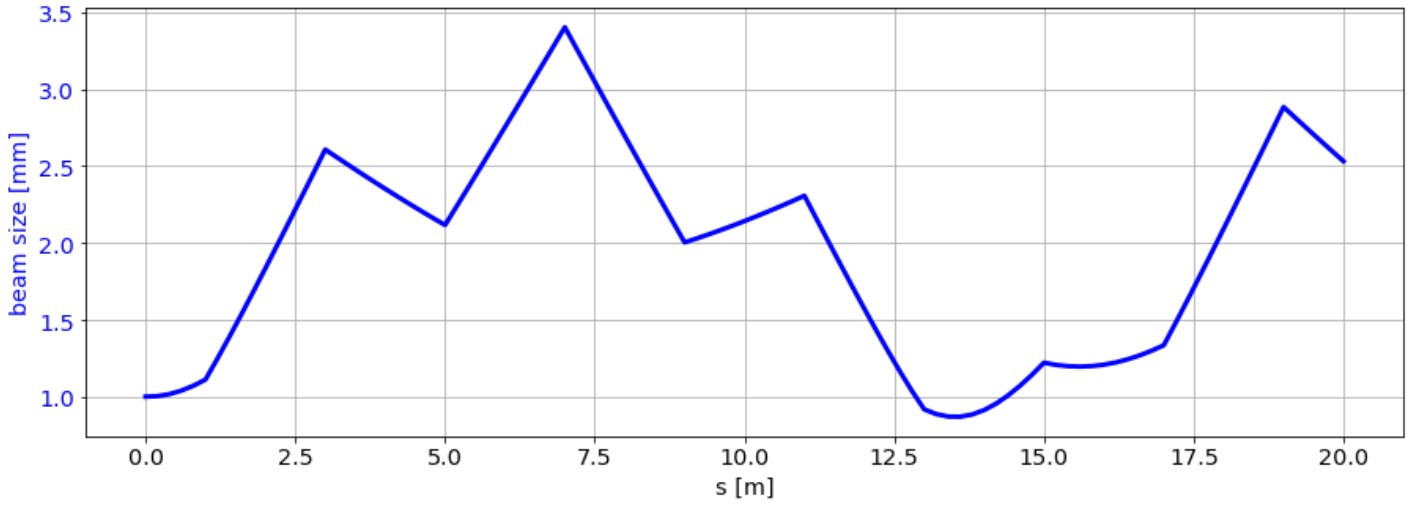
In [2]:

```
params = {'legend.fontsize': 'x-large',
          'figure.figsize': (15, 5),
         'axes.labelsize': 'x-large',
         'axes.titlesize':'x-large',
         'xtick.labelsize':'x-large',
         'ytick.labelsize':'x-large'}
plt.rcParams.update(params)

#prepare the optics
f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(-f)+10*D(L_2/(10))+Q(f)+5*D(L_2/2/5)
beamline_5FODO=5*fodo_lattice

#prepare the beam
Npart=10000
beam=np.random.randn(2, Npart)
x0=0;
xp0=1
sigx=1;
sigxp=0.5;
beam[0,:]=sigx*beam[0,:]+x0
beam[1,:]=sigxp*beam[1,:]+xp0

beamlist=[(beam,0)]
for element in beamline_5FODO:
    beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))

plt.plot([x[1] for x in beamlist],[np.mean(x[0][0,:])for x in beamlist],'o-b',lw=3)
plt.grid(True)
plt.xlabel('s [m]')
plt.gca().set_ylabel('x [mm]', color='b')
plt.gca().tick_params(axis='y', labelcolor='b')

ax2 = plt.gca().twinx()  # instantiate a second axes that shares the same x-axis
ax2.set_ylabel("divergence [mrad]", color='r')
ax2.tick_params(axis='y', labelcolor='r')
ax2.plot([x[1] for x in beamlist],[np.std(x[0][1,:])for x in beamlist],'o-r',lw=3)
plt.title('Exercise 14');
```



In [3]:

```
plt.plot([x[1] for x in beamlist],[np.std(x[0][0,:])for x in beamlist],'-b',lw=3)
plt.grid(True)
plt.xlabel('s [m]')
plt.gca().set_ylabel('beam size [mm]', color='b')
plt.gca().tick_params(axis='y', labelcolor='b')
```



Exercise 15

Starting from Exercise 14, what happens if you (a) increase or (b) reduce by a factor 2 the initial beam divercence (sigxp)?

SOLUTIONS

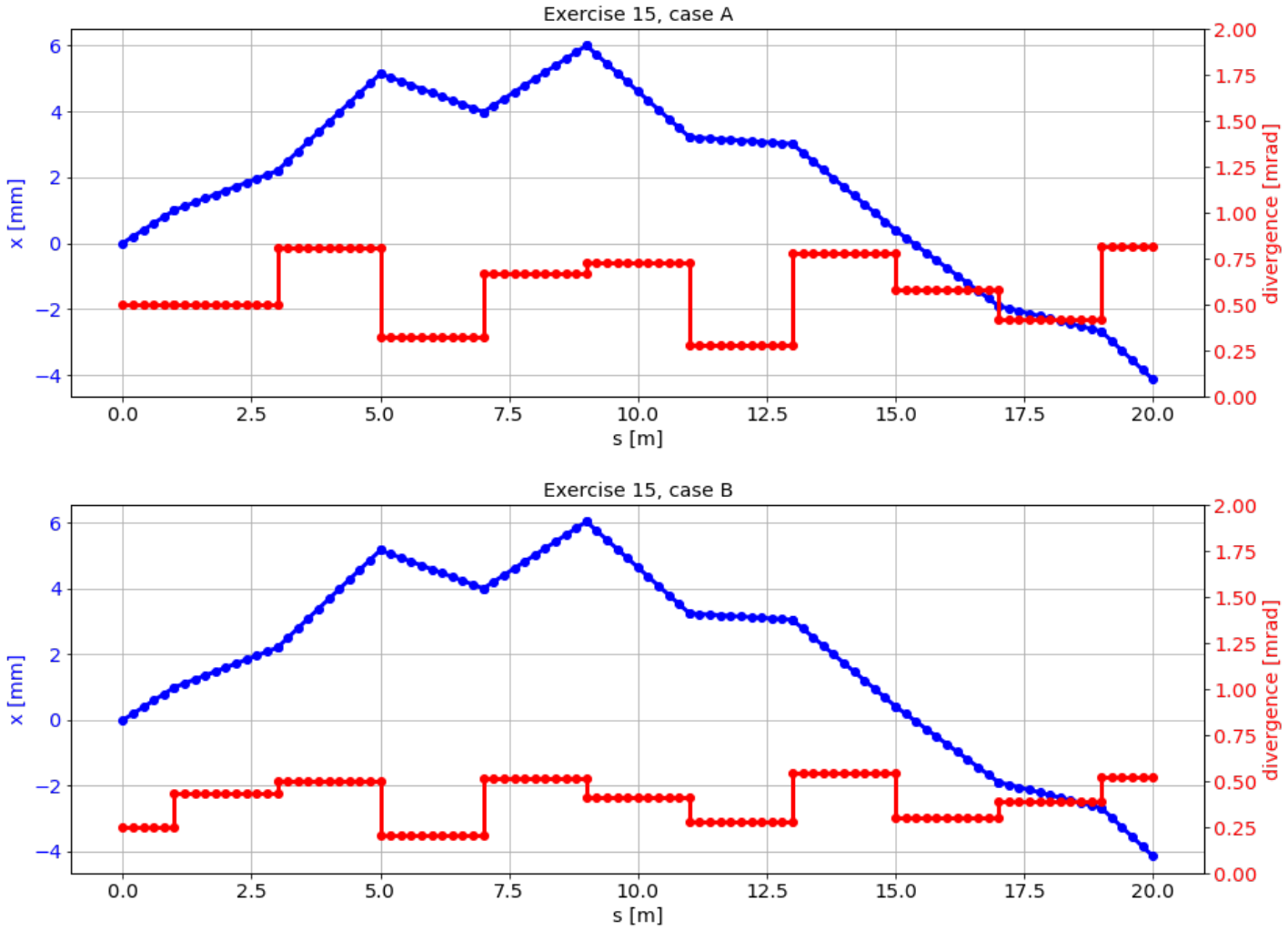
In [4]:

```
# CASE a
f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(f)+10*D(L_2/(10))+Q(-f)+5*D(L_2/2/5)
beamline_5FODO=5*fodo_lattice

#prepare the beam
Npart=10000
beam=np.random.randn(2, Npart)
x0=0;
xp0=1
sigx=1;
sigxp=0.5;
beam[0,:]=sigx*beam[0,:]+x0
beam[1,:]=sigxp*beam[1,:]+xp0

beamlist=[(beam,0)]
for element in beamline_5FODO:
    beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))

plt.plot([x[1] for x in beamlist],[np.mean(x[0][0,:])for x in beamlist],'o-b',lw=3)
plt.grid(True)
plt.xlabel('s [m]')
plt.gca().set_ylabel('x [mm]', color='b')
plt.gca().tick_params(axis='y', labelcolor='b')

ax2 = plt.gca().twinx()  # instantiate a second axes that shares the same x-axis
ax2.set_ylabel("divergence [mrad]", color='r')
ax2.tick_params(axis='y', labelcolor='r')
ax2.plot([x[1] for x in beamlist],[np.std(x[0][1,:])for x in beamlist],'o-r',lw=3)
plt.ylim([0,2])

plt.title('Exercise 15, case A')
```

```
# CASE b, just a copy of the first one with a small change...
f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(f)+10*D(L_2/(10))+Q(-f)+5*D(L_2/2/5)
beamline_5FODO=5*fodo_lattice

#prepare the beam
Npart=10000
beam=np.random.randn(2, Npart)
x0=0;
xp0=1
sigx=1;
sigxp=0.5/2;
beam[0,:]=sigx*beam[0,:]+x0
beam[1,:]=sigxp*beam[1,:]+xp0

beamlist=[(beam,0)]
for element in beamline_5FODO:
    beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))

plt.figure()
plt.plot([x[1] for x in beamlist],[np.mean(x[0][0,:])for x in beamlist],'o-b',lw=3)
plt.grid(True)
plt.xlabel('s [m]')

plt.gca().set_ylabel('x [mm]', color='b')
plt.gca().tick_params(axis='y', labelcolor='b')

ax2 = plt.gca().twinx()  # instantiate a second axes that shares the same x-axis
ax2.set_ylabel("divergence [mrad]", color='r')
ax2.tick_params(axis='y', labelcolor='r')
ax2.plot([x[1] for x in beamlist],[np.std(x[0][1,:])for x in beamlist],'o-r',lw=3)
plt.ylim([0,2])

plt.title('Exercise 15, case B')
```

Out[4]:

```
Text(0.5, 1.0, 'Exercise 15, case B')
```



It very important to note that the divergence of CASE A and CASE B are not proportional.

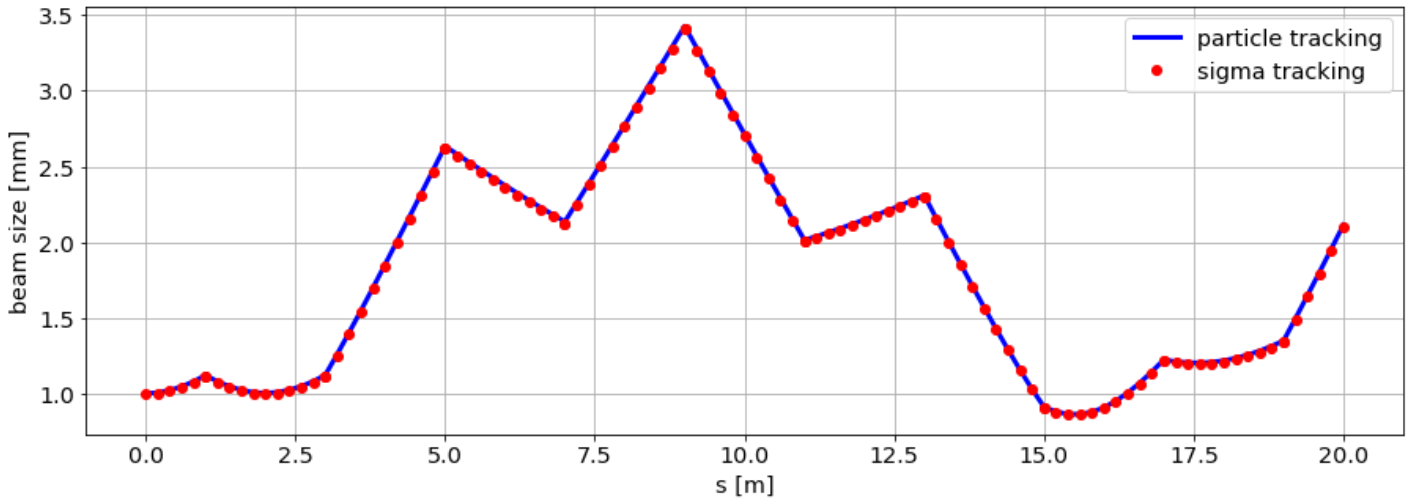
Exercise 16

Using Equation from the Primer, display (a) the average position of the particles along the beam line. Likewise, (b) display the angular divergence. Compare with the result you found in Exercise 14.

SOLUTIONS

In [5]:

```
#lattice
f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(f)+10*D(L_2/(10))+Q(-f)+5*D(L_2/2/5)
beamline_5FODO=5*fodo_lattice

#prepare the beam
Npart=10000
beam0=np.random.randn(2, Npart)
x0=0;
xp0=1
sigx=1;
sigxp=0.5;
beam0[0,:]=sigx*beam0[0,:]+x0
beam0[1,:]=sigxp*beam0[1,:]+xp0
beamlist=[(beam0,0)]

#prepare the sigma matrix
sigma0=np.array([[sigx**2, 0],[0, sigxp**2]])
sigmalist=[(sigma0,0)]

for element in beamline_5FODO:
    beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))

for element in beamline_5FODO:
    sigmalist.append((element[0] @ sigmalist[-1][0] @ element[0].transpose() ,sigmalist[-1][1]+ element[1]))

plt.plot([x[1] for x in beamlist],[np.std(x[0][0,:])for x in beamlist],'-b',lw=3, label='particle tracking')
plt.plot([x[1] for x in sigmalist],[np.sqrt(x[0][0,0]) for x in sigmalist],'or', lw=3,label='sigma tracking')
plt.grid(True)
plt.legend(loc='best')
plt.xlabel('s [m]')
plt.ylabel('beam size [mm]');
```



Exercise 17

Can you find an initial beam matrix sigma0 that reproduces itself at the end of the beam line?

SOLUTION

This problem is not so simple. One could proceed with try and error. But there are three parameters to fix so it can be quite cumbersome.

In [43]:

```
from ipywidgets import interactive

def plotIt(sigma11, sigma22, sigma12):
    #lattice
    f=2.5
    L_2=2
    fodo_lattice= 5*D(L_2/2/5)+Q(f)+10*D(L_2/(10))+Q(-f)+5*D(L_2/2/5)
    beamline_5FODO=5*fodo_lattice

    #prepare the sigma matrix
    sigma0=np.array([[sigma11,sigma12 ],[sigma12, sigma22]])
    sigmalist=[(sigma0,0)]

    for element in beamline_5FODO:
        sigmalist.append((element[0] @ sigmalist[-1][0] @ element[0].transpose() ,sigmalist[-1][1]+ element[1]
))

    plt.plot([x[1] for x in sigmalist],[np.sqrt(x[0][0,0]) for x in sigmalist],'-b',lw=3)
    plt.grid(True)

interactive_plot = interactive(plotIt,sigma11=(0,10,.1),sigma22=(0,10,.01),sigma12=(-10,10,.01),continuous_upd
ate=False)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

In [44]:

```
def plotIt(sigma11):
    #lattice
    f=2.5
    L_2=2
    sigma22=1/sigma11
    sigma12=0
    # Very convenient to chose a point with no x-xp correlation
    fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
    beamline_5FODO=5*fodo_lattice

    #prepare the sigma matrix
    sigma0=np.array([[sigma11,sigma12 ],[sigma12, sigma22]])
    sigmalist=[(sigma0,0)]

    for element in beamline_5FODO:
        sigmalist.append((element[0] @ sigmalist[-1][0] @ element[0].transpose() ,sigmalist[-1][1]+ element[1]
))

    #plt.plot([x[1] for x in beamlist],[np.std(x[0][0,:])for x in beamlist],'-b',lw=3)
    plt.plot([x[1] for x in sigmalist],[np.sqrt(x[0][0,0]) for x in sigmalist],'b-',lw=3)
    plt.grid(True)

interactive_plot = interactive(plotIt,sigma11=(0,10,.1),continuous_update=False)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

If the lattice is stable (and this is the case of our simple FODO, $f > L_{FODO}/4$), then only a specific sigma matrix (after having fixed the beam emittance, in our example we choose unitary emittance) shows a periodic behaviour. The beam associated to that sigma matrix is the **matched beam**.

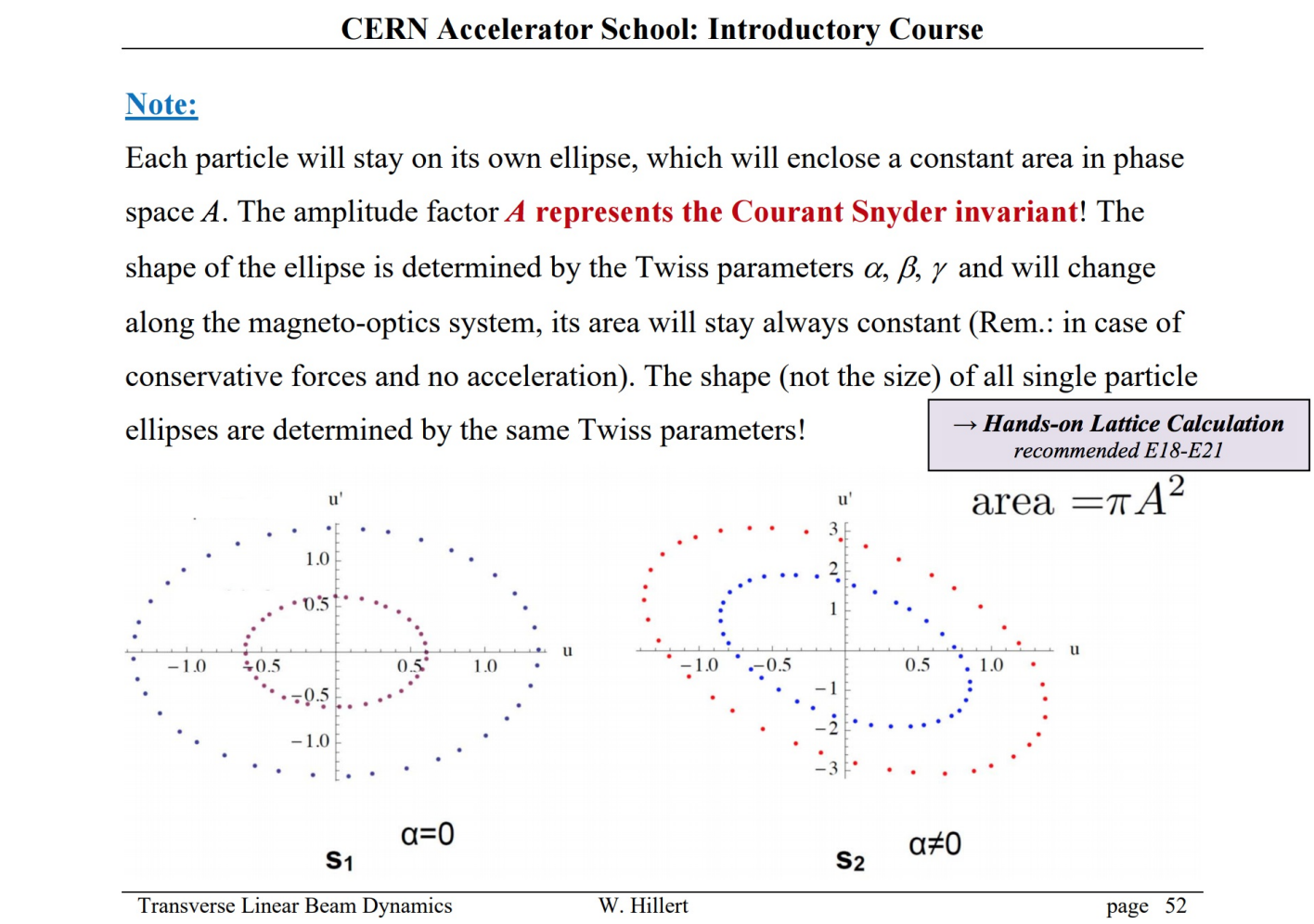
Periodical systems

Now we are going to explore the behaviour of the single particle (and later of the beam), not along the generic position *s* but turn-after-turn, i.e., s_0, $s_0 + L_{period}$, $s_0 + 2L_{period}$, ... where L_{period} is the period of our periodic lattice.

In [8]:

```
# Refer to
from IPython.display import Image
fig = Image(filename=('/Users/sterbini/CERNBox/2019/CAS/Vysoke_Tatry/Python/Friday/pag52.png'))
fig
```

Out[8]:



Exercise 18

Explore different initial coordinate of a single particle in a periodical FODO lattice. Compare the phase-space plots you obtain by considering the particle position in the trace space for the different turns.

SOLUTIONS

For plotting the trace space evolution of a particle we need to observe it turn-after-turn. The natural way to do that is to compress the beam line (in general composed by more than one element) in a single transformation: the **one-turn-matrix**. Do to so we introduce a special method.

In [ ]:

```

```

In [45]:

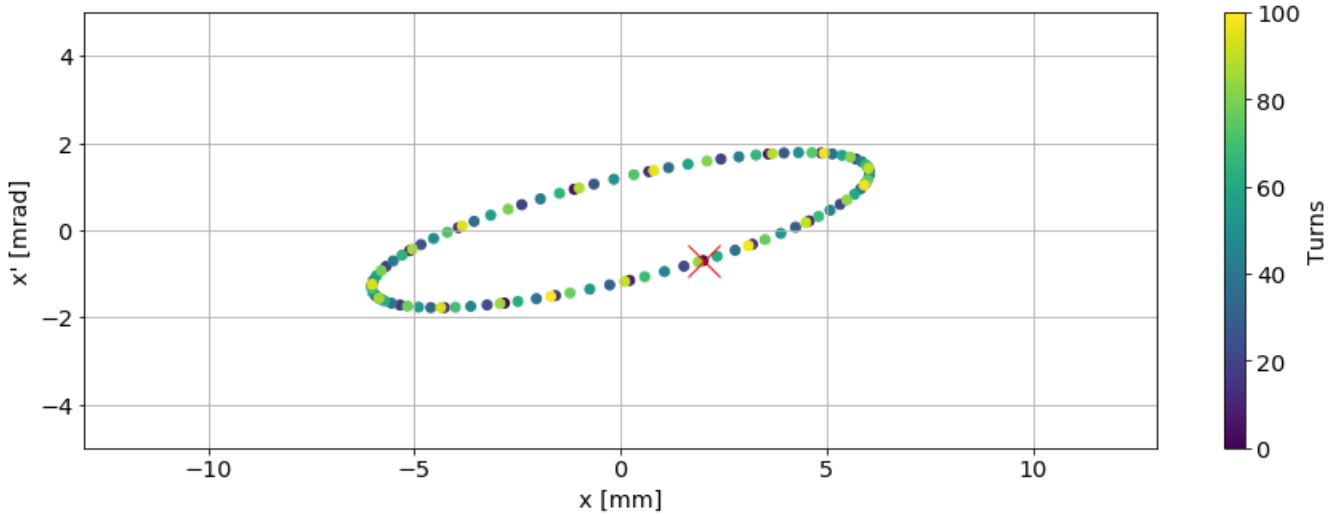
```
def compress_beamline(my_beamline, dimension=2):
    M=np.eye(dimension)
    s=0
    for i in my_beamline:
        M=i[0] @ M
        s=s+i[1]
    return [(M,s)]

f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(f)+10*D(L_2/(10))+Q(-f)+5*D(L_2/2/5)
OTM=compress_beamline(fodo_lattice)
```

In [46]:

```
def plotIt(x, xp):
    beamlist=[(np.array([[x],[xp]]),0)]
    myLattice=100*OTM
    for element in myLattice:
        beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))
    plt.scatter(np.array([x[0][0][0] for x in beamlist]),np.array([x[0][1][0] for x in beamlist]),c=[x[1]/(L_2
*2) for x in beamlist])
    plt.plot(beamlist[0][0][0],beamlist[0][0][1],'xr',ms=20)
    cb=plt.colorbar()
    cb.set_label('Turns')
    plt.xlabel('x [mm]')
    plt.ylabel("x' [mrad]")
    plt.xlim(-13,13)
    plt.ylim(-5,5)
    plt.grid(True)


interactive_plot = interactive(plotIt,x=(-2,2,.1),xp=(-2,2,.1),continuous_update=True)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

In [11]:

```
plotIt(2, -.7)
```



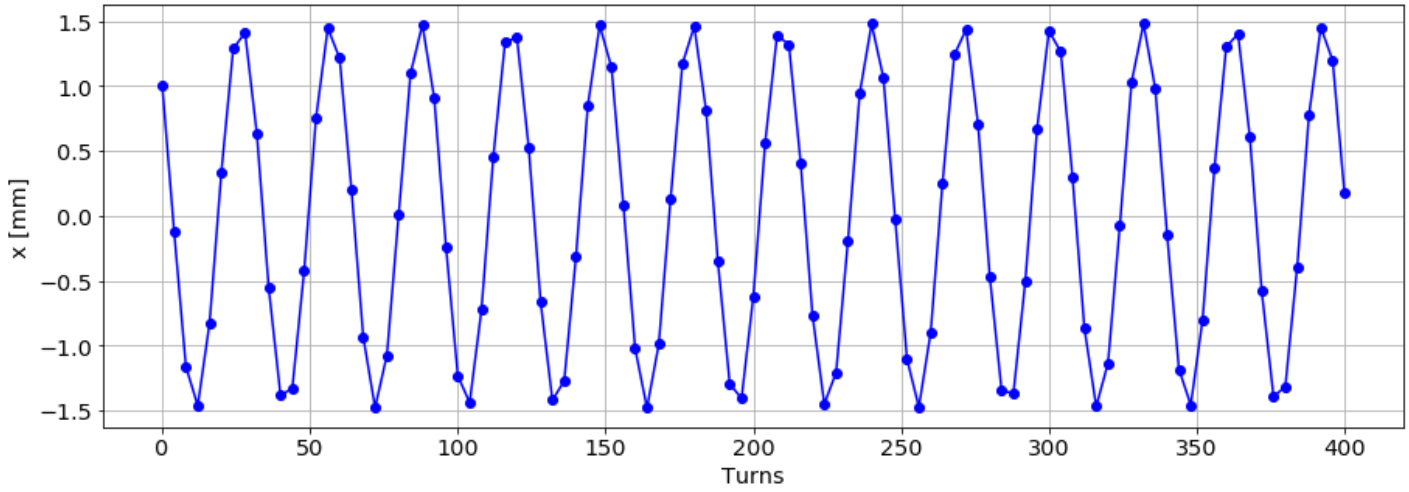
Exercise 19

Plot the position of the particle vs the number of turns. What do you obtain?

SOLUTION

In [12]:

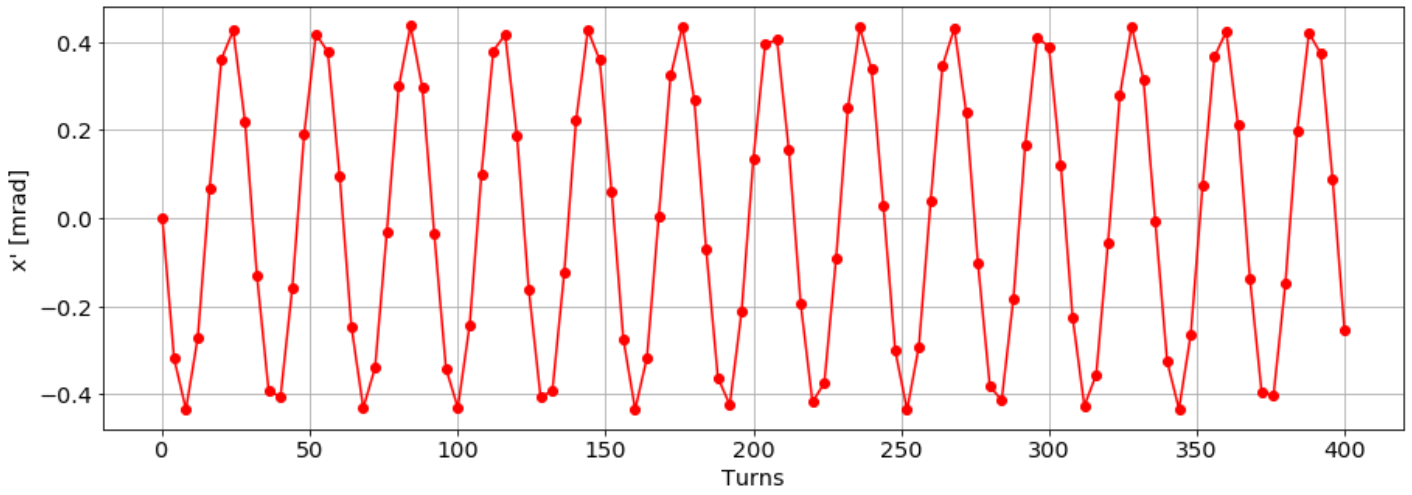
```
f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(f)+10*D(L_2/(10))+Q(-f)+5*D(L_2/2/5)
OTM=compress_beamline(fodo_lattice)
beamlist=[(np.array([[1],[0]]),0)]
myLattice=100*OTM
for element in myLattice:
    beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))

plt.plot([x[1] for x in beamlist],[x[0][0] for x in beamlist],'o-b')
plt.xlabel('Turns')
plt.ylabel('x [mm]')
plt.grid(True)
```



We clearly see a betatronic oscillation. And this is true also for x' (se below, and not the different amplitude and phase).

In [13]:

```
plt.plot([x[1] for x in beamlist],[x[0][1] for x in beamlist],'o-r')
plt.xlabel('Turns')
plt.ylabel("x' [mrad]")
plt.grid(True)
```



What is the fractional tune of the machine?

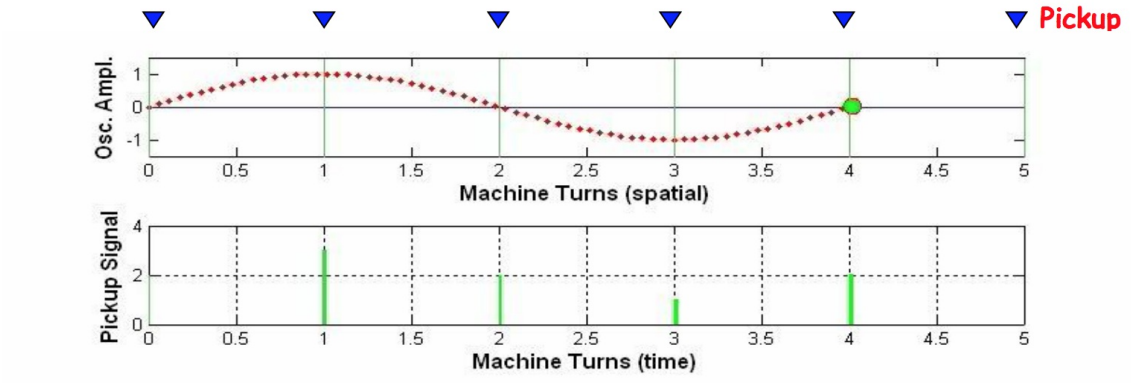
In [14]:

```
# Refer to
from IPython.display import Image
fig = Image(filename=('/Users/sterbini/CERNBox/2019/CAS/Vysoke_Tatry/Python/Friday/hermann.png'))
fig
```

Out[14]:

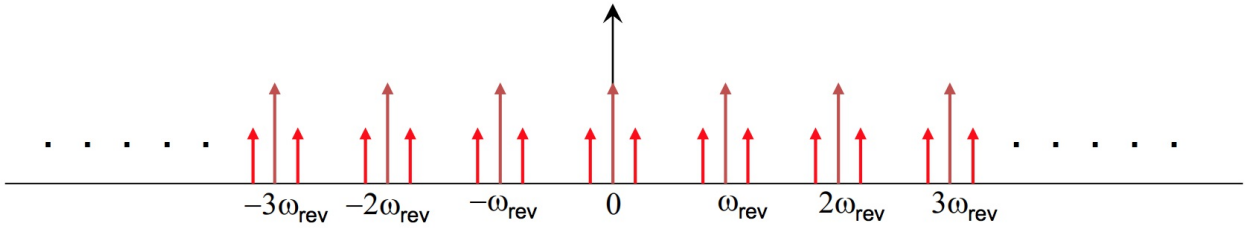
Multi-bunch modes: single oscillating bunch

One single unstable bunch oscillating at the tune frequency $\nu\omega_0$: for simplicity we consider a vertical tune $\nu < 1$, ex. $\nu = 0.25$. $M = 1 \rightarrow$ only mode #0 exists



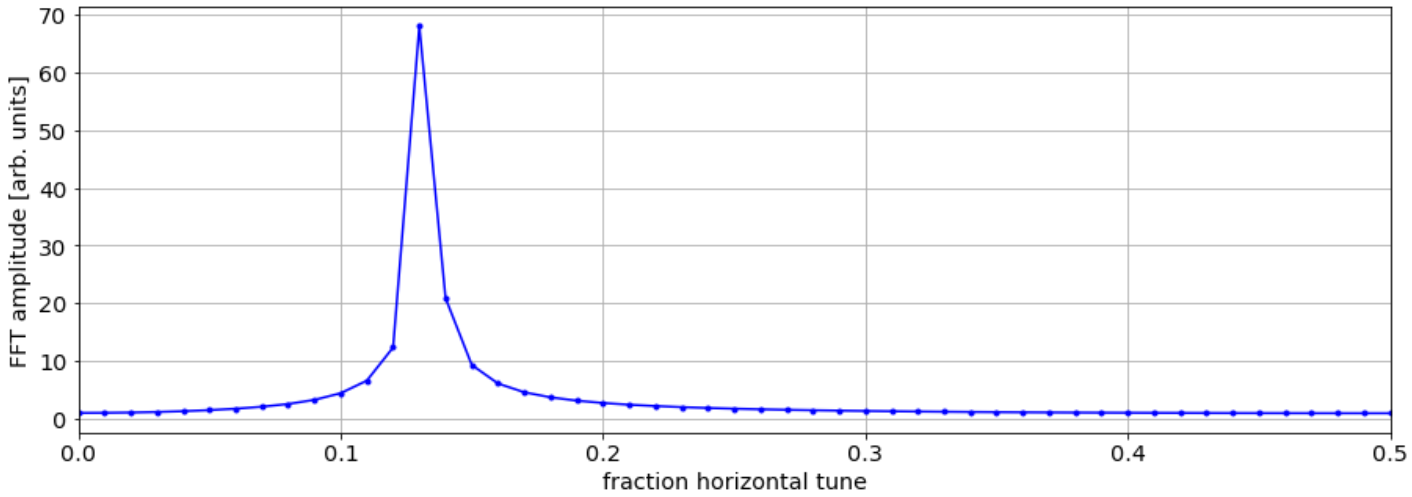
The pickup signal is a sequence of pulses modulated in amplitude with frequency $\nu\omega_0$

Two sidebands at $\pm\nu\omega_0$ appear at each of the revolution harmonics



In [15]:

```
x_vector=[x[0][0][0] for x in beamlist]
# fft
fft_amplitude=np.abs(np.fft.fft(x_vector))
plt.plot(np.linspace(0,1,len(fft_amplitude)),fft_amplitude,'.-b')
plt.xlim([0,0.5])
plt.xlabel('fraction horizontal tune')
plt.ylabel('FFT amplitude [arb. units]')
plt.grid()
```



Exercise 20

In the definition of FODO at the previous exercise reverse the polarity of both quadrupoles and prepare a phase-space plot. How does it differ from the one in Exercise 18?

SOLUTION

In [16]:

```
f=2.5
L_2=2
fodo_lattice= 5*D(L_2/2/5)+Q(-f)+10*D(L_2/(10))+Q(f)+5*D(L_2/2/5)
OTM=compress_beamline(fodo_lattice)

def plotIt(x, xp):
    beamlist=[(np.array([[x],[xp]]),0)]
    myLattice=100*OTM
    for element in myLattice:
        beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))
    plt.scatter(np.array([x[0][0][0] for x in beamlist]),np.array([x[0][1][0] for x in beamlist]),c=[x[1]/(L_2
*2) for x in beamlist])
    plt.plot(beamlist[0][0][0],beamlist[0][0][1],'xr',ms=20)
    cb=plt.colorbar()
    cb.set_label('Turns')
    plt.xlabel('x [mm]')
    plt.ylabel("x' [mrad]")
    plt.xlim(-13,13)
    plt.ylim(-5,5)
    plt.grid(True)


interactive_plot = interactive(plotIt,x=(-2,2,.1),xp=(-2,2,.1),continuous_update=True)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

As you can see now the ellipse tilt is inverted.

Exercise 21

From Exercise 20, change the starting point of the lattice at the center of the focusing quadrupole (hint: split the focusing quadrupole in two focusing quadrupole with double focal length and place them at the start and at the end of the lattice).

SOLUTION

In [17]:

```
f=2.5
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
OTM=compress_beamline(fodo_lattice)

def plotIt(x, xp):
    beamlist=[(np.array([[x],[xp]]),0)]
    myLattice=100*OTM
    for element in myLattice:
        beamlist.append((element[0] @ beamlist[-1][0],beamlist[-1][1]+ element[1]))
    plt.scatter(np.array([x[0][0][0] for x in beamlist]),np.array([x[0][1][0] for x in beamlist]),c=[x[1]/(L_2
*2) for x in beamlist])
    plt.plot(beamlist[0][0][0],beamlist[0][0][1],'xr',ms=20)
    cb=plt.colorbar()
    cb.set_label('Turns')
    plt.xlabel('x [mm]')
    plt.ylabel("x' [mrad]")
    plt.xlim(-18,18)
    plt.ylim(-2.3,2.3)
    plt.grid(True)


interactive_plot = interactive(plotIt,x=(-2,2,.1),xp=(-2,2,.1),continuous_update=True)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

The ellipse is not tilted anymore.

Exercise 22

Find the range of focal lengths f for which the FODO cells permit stable oscillations.

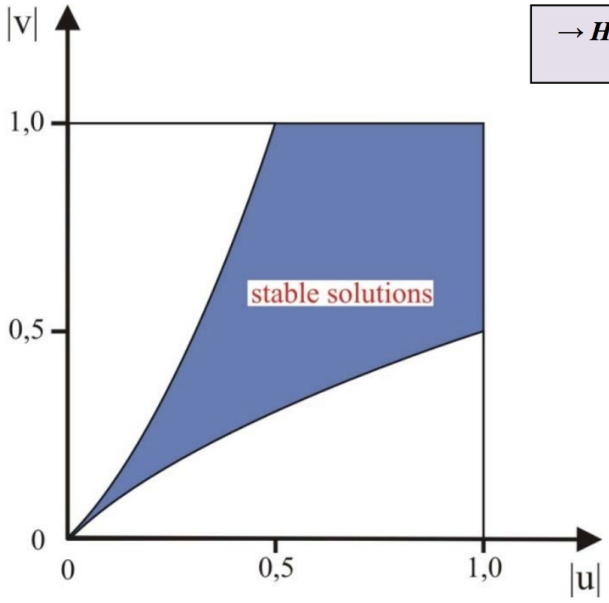
In [18]:

```
# Refer to
from IPython.display import Image
fig = Image(filename=('/Users/sterbini/CERNBox/2019/CAS/Vysoke_Tatry/Python/Friday/pag71.png'))
fig
```

Out[18]:

CERN Accelerator School: Introductory Course

which gives the famous necktie-diagram for thin lens approximation:



→ ***Hands-on Lattice Calculation***
recommended E22

In the simple case of equal focusing strengths, we arrive at

$$|f_1| = |f_2| = \frac{|f_D|}{2} = \frac{|f_F|}{2} = \frac{|f|}{2} \quad \rightarrow \quad \left|\frac{2L}{f}\right| = \left|\frac{L_{\mathrm{FODO}}}{f}\right| < 1$$

Transverse Linear Beam Dynamics W. Hillert page 71

SOLUTION

To solve this exercise we need to compute the phase advance and initial optics functions associated to a OTM rotation. To do that we define the following function.

In [47]:

```
def R2beta(R):
    mu=np.arccos(0.5*(R[0,0]+R[1,1]))
    if (R[0,1]<0):
        mu=2*np.pi-mu;
    Q=mu/(2*np.pi)
    beta=R[0,1]/np.sin(mu)
    alpha=(0.5*(R[0,0]-R[1,1]))/np.sin(mu)
    gamma=(1+alpha**2)/beta
    return (Q, beta, alpha, gamma)
```

In [48]:

```
f=1
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
OTM=compress_beamline(fodo_lattice)
R2beta(OTM[0][0])
```

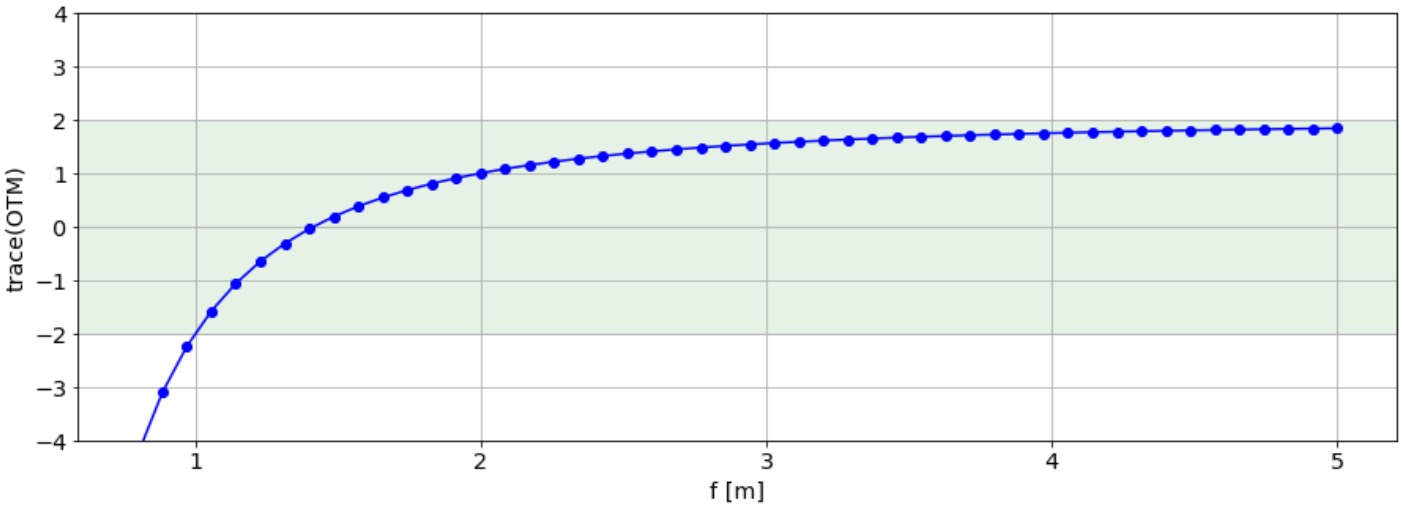
Out[48]:

```
(0.49999999419080615,
 219176631.31217423,
 -7.604216954454841e-09,
 4.562530202299239e-09)
```

The stability condition is satisfied if (only 1D case), the trace of the matrix is smaller than 2 (for the H adn V case). For a FODO this can be done analytically. But in this exercise we will follow a numerical approach.

In [49]:

```
def myOTM_trace(f):
    L_2=2
    fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
    OTM=compress_beamline(fodo_lattice)
    return np.trace(OTM[0][0])

def setShadedRegion(ax,color='g' ,xLimit=[0,1], yLimit='FullRange',alpha=.1):
    """
    setShadedRegion(ax,color='g' ,xLimit=[0,1],alpha=.1)
    ax: plot axis to use
    color: color of the shaded region
    xLimit: vector with two scalars, the start and the end point
    alpha: transparency settings
    yLimit: if set to "FullRange" shaded the entire plot in the y direction
    If you want to specify an intervall, please enter a two scalar vector as xLimit
    """
    if yLimit=='FullRange':
        aux=ax.get_ylim()
        plt.gca().fill_between(xLimit, [aux[0],aux[0]],  [aux[1],aux[1]],color=color, alpha=alpha)
        ax.set_ylim(aux)
    else:
        plt.gca().fill_between(xLimit,
                    [yLimit[0],yLimit[0]],  [yLimit[1],yLimit[1]],color=color, alpha=alpha)

f_range=np.linspace(.8,5)
plt.plot(f_range, [myOTM_trace(f) for f in f_range],'-bo')
my_xlim=plt.xlim()
setShadedRegion(plt.gca(),xLimit=my_xlim,yLimit=[-2,2])
plt.ylim(-4,4)
plt.xlim(my_xlim);
plt.xlabel('f [m]')
plt.ylabel('trace(OTM)')
plt.grid(True)
```

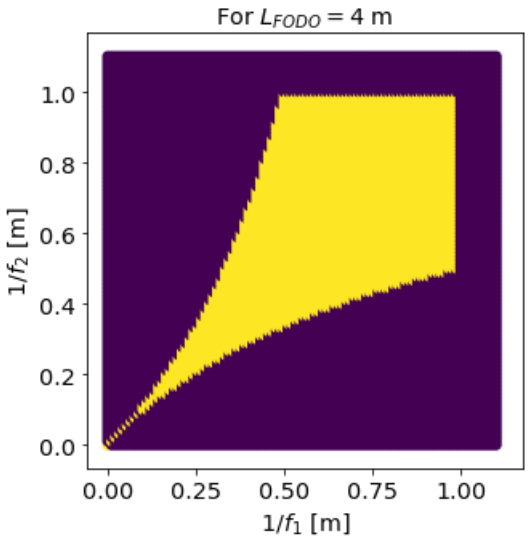


In [50]:

```
# if we want to do it for quadrupoles with different focal lenght you have to take
#into account explicity the vertical plane.
import itertools
L_2=2
kl1_list=[]
kl2_list=[]
stable_list=[]
for kl1, kl2 in itertools.product(np.linspace(.001,1.1,100),np.linspace(.001,1.1,100)):
    kl1_list.append(kl1)
    kl2_list.append(kl2)
    f1=1/kl1
    f2=1/kl2
    fodo_lattice_h= Q(f1)+10*D(L_2/10)+Q(-f2)+10*D(L_2/(10))
    fodo_lattice_compressed_h=compress_beamline(fodo_lattice_h)
    fodo_lattice_v= Q(-f1)+10*D(L_2/10)+Q(f2)+10*D(L_2/(10))
    fodo_lattice_compressed_v=compress_beamline(fodo_lattice_v)
    if np.abs(np.trace(fodo_lattice_compressed_h[0][0]))<2 and np.abs(np.trace(fodo_lattice_compressed_v[0][0]
))<2:
        stable_list.append(1)
    else:
        stable_list.append(0)

plt.scatter(kl1_list,kl2_list,c=stable_list)
plt.axis('square');
plt.xlabel('$1/f_1$ [m]')
plt.ylabel('$1/f_2$ [m]')
plt.title('For $L_{FODO}=4$ m')
# the kite's area is the stable one as shown in the lecture
```

Out[50]:

```
Text(0.5, 1.0, 'For $L_{FODO}=4$ m')
```



Computing the optical functions

Now we will use the matrix approach to compute the optical functions as suggested in the lecture.

In [23]:

```
# Refer to
from IPython.display import Image
fig = Image(filename=('/Users/sterbini/CERNBox/2019/CAS/Vysoke_Tatry/Python/Friday/pag73.png'))
fig
```

Out[23]:

4.3.2. Periodic beta functions

Periodic solutions of a periodic lattice of period-length L will be

$$\beta(s_0 + L) = \beta(s_0) = \beta_0$$
$$\alpha(s_0 + L) = \alpha(s_0) = \alpha_0$$

Comparing the transfer matrix for one period with its Twiss parameter representation

$$\mathbf{M} = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix} = \begin{pmatrix} \cos\mu + \alpha_0 \sin\mu & \beta_0 \sin\mu \\ -\gamma_0 \sin\mu & \cos\mu - \alpha_0 \sin\mu \end{pmatrix}$$

we can determine the Twiss parameters at the symmetry points (where $\alpha = 0$!)

$$\alpha_0 = 0, \qquad \beta_0 = \frac{r_{12}}{\sqrt{1 - r_{11}^2}}, \qquad \gamma_0 = \frac{-r_{21}}{\sqrt{1 - r_{11}^2}}, \qquad \cos\mu = r_{11}$$

and transform them to any position s using e.g. the beta matrix formalism

$$\begin{pmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{pmatrix} = \mathbf{M}(s, s_0) \cdot \begin{pmatrix} \beta_0 & 0 \\ 0 & \gamma_0 \end{pmatrix} \cdot {}^T\mathbf{M}(s, s_0)$$

→ ***Hands-on Lattice Calculation***
recommended E23-E26

thus revealing the development of β(s), α(s), γ(s). Minimum $\langle\beta\rangle$ for $\mu \approx 90°$!

Exercise 23

Using the function **R2beta** (see the Primer) you can finally find the periodic opitcs solution of the lattice. From that you can find the sigma-matrix of the beam that is periodic. Convince yourself that the σ matrix is indeed equal to the one at the start, σ_0.

SOLUTION

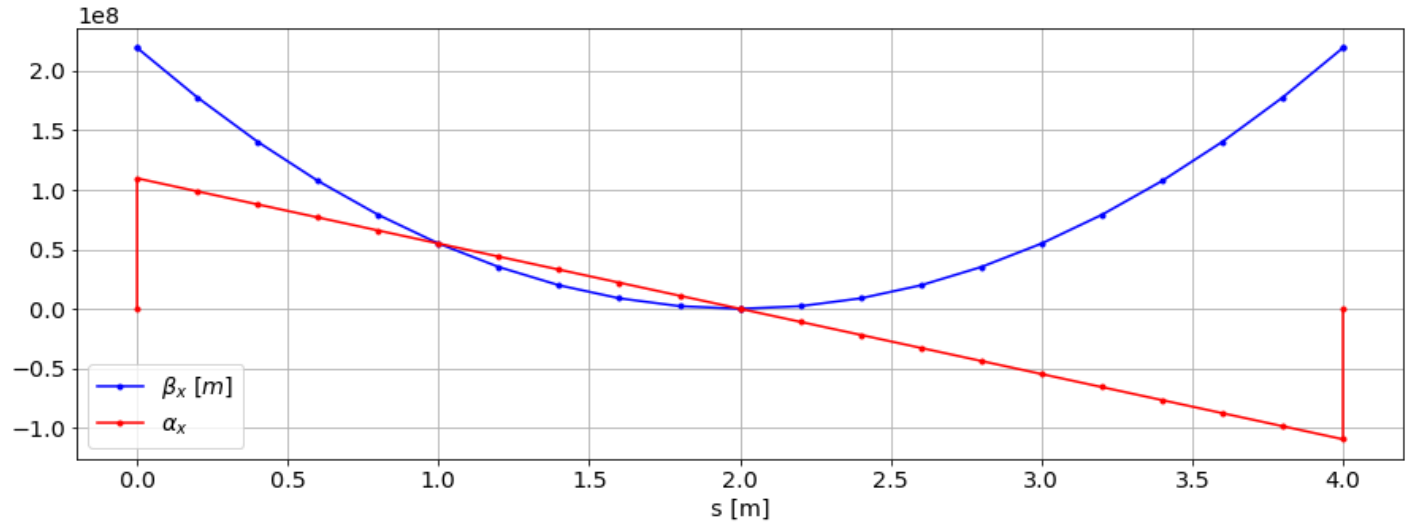
One has to remeber the definition of the σ matrix (for the momement we will consired $\epsilon = 1$) and how to transport it.

In [24]:

```
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_0=np.array([[beta, -alpha],[-alpha, gamma]])

sigma_list=[(sigma_0,0)];
for element in fodo_lattice:
    sigma_list.append((element[0]@sigma_list[-1][0]@(element[0].transpose()),sigma_list[-1][1]+element[1]))

plt.plot([i[1] for i in sigma_list], [i[0][0,0] for i in sigma_list],'.-b',label='$\\beta_x\ [m]$')
plt.plot([i[1] for i in sigma_list], [-i[0][0,1] for i in sigma_list],'.-r',label='$\\alpha_x$')
plt.grid(True)
plt.legend(loc='best')

plt.xlabel('s [m]')
```

Out[24]:

```
Text(0.5, 0, 's [m]')
```



Exercise 24

Write down the numerical values of initial beam matrix σ_0, then build a beam line made of 15 consecutive cells by changing the definition of the lattince and then, using σ_0 with the noted-down numbers, prepare a plot of the beam sizes along the 15 cells. Is it also periodic?

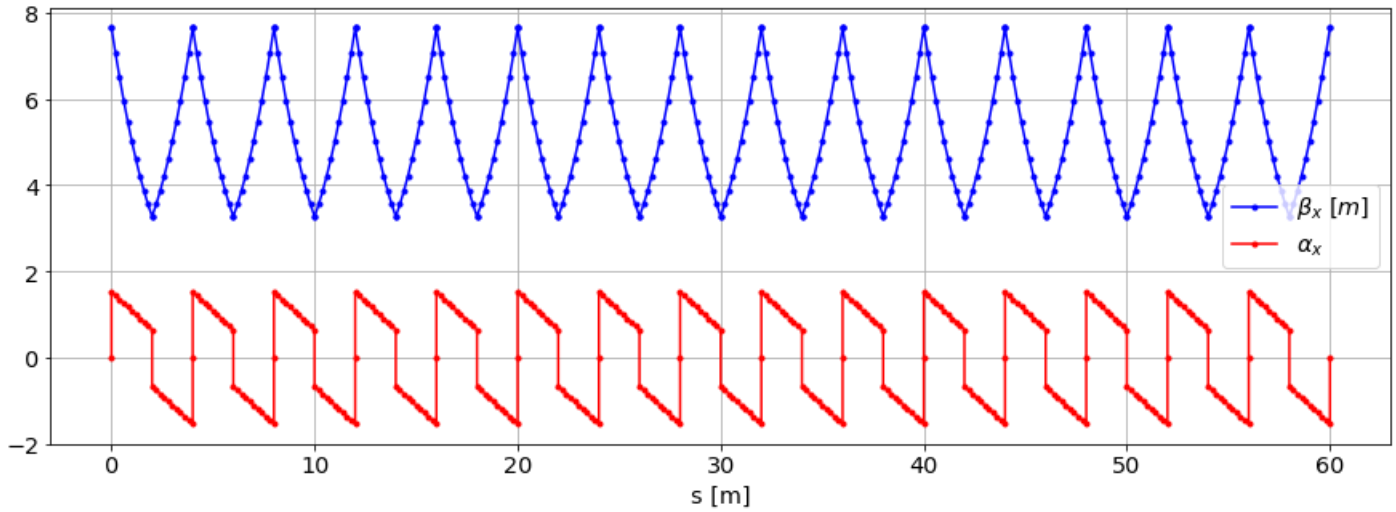
SOLUTION

In [25]:

```
f=2.5
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_0=np.array([[beta, -alpha],[-alpha, gamma]])

sigma_list=[(sigma_0,0)];
for element in 15*fodo_lattice:
    sigma_list.append((element[0]@sigma_list[-1][0]@(element[0].transpose()),sigma_list[-1][1]+element[1]))

plt.plot([i[1] for i in sigma_list], [i[0][0,0] for i in sigma_list],'.-b',label='$\\beta_x\ [m]$')
plt.plot([i[1] for i in sigma_list], [-i[0][0,1] for i in sigma_list],'.-r',label='$\\alpha_x$')
plt.grid(True)
plt.legend(loc='best')
plt.xlabel('s [m]')
```

Out[25]:

```
Text(0.5, 0, 's [m]')
```



Exercise 25

Verify that all the "cumulative" matrices (product of the first n elements of a beam-line) have unit determinant. Is this a sufficient condition for being symplectic matrices?

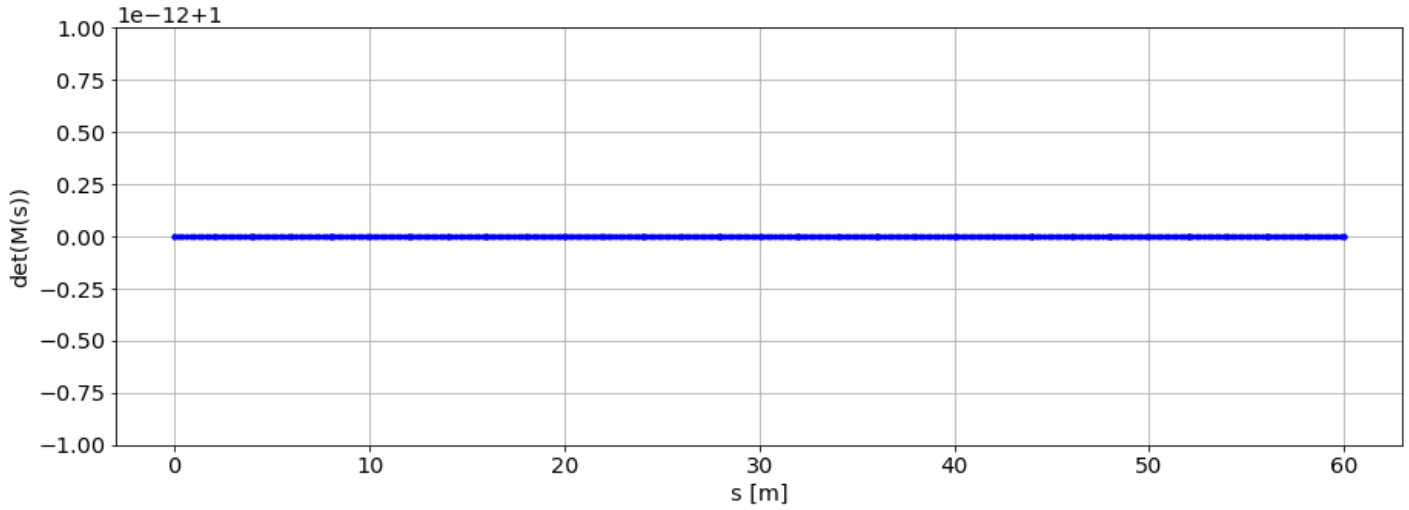
SOLUTION

In [26]:

```
f=2.5
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)

cumulative_matrix_list=[(np.eye(2),0)];

for element in 15*fodo_lattice:
    cumulative_matrix_list.append((element[0]@cumulative_matrix_list[-1][0],cumulative_matrix_list[-1][1]+elem
ent[1]))

plt.plot([i[1] for i in cumulative_matrix_list], [np.linalg.det(i[0]) for i in cumulative_matrix_list],'b.-')
plt.grid(True)
plt.xlabel('s [m]')
plt.ylabel('det(M(s))')
```

Out[26]:

```
Text(0, 0.5, 'det(M(s))')
```

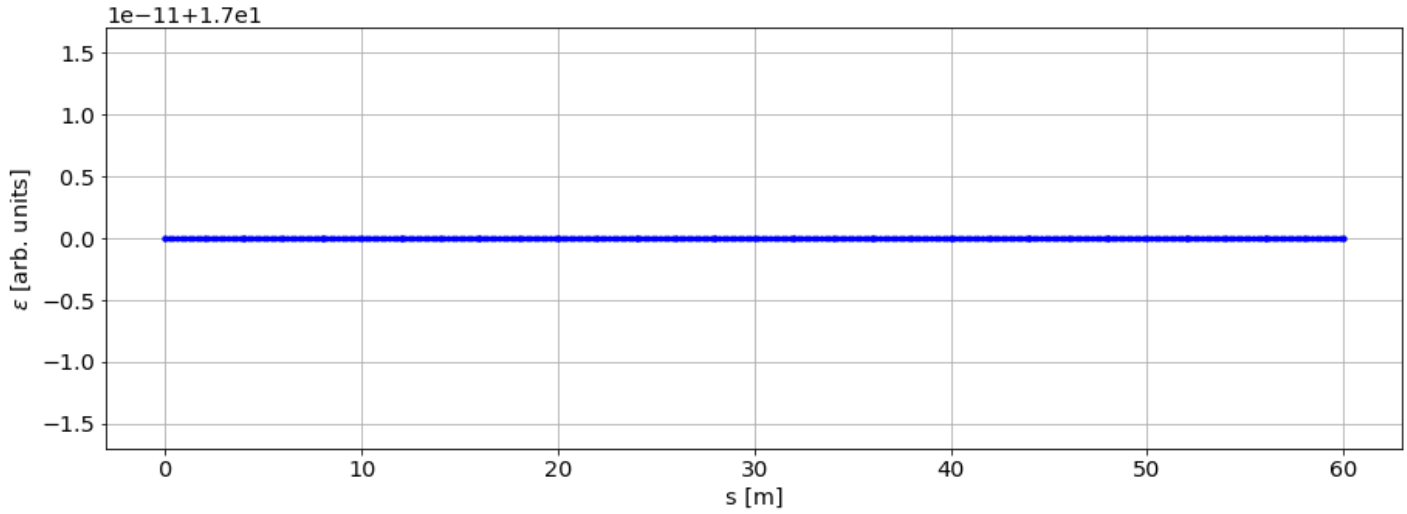


Exercise 26

Multiply σ_0 from Exercise 24 by 17 and calculate the emittance. Then propagate the sigma matrix through the beam line from Exercise 24 and verify that the emittance of the sigma matrix after every element is indeed constant and equal to its initial value.

In [27]:

```
f=2.5
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_0=17*np.array([[beta, -alpha],[-alpha, gamma]])

sigma_list=[(sigma_0,0)];
for element in 15*fodo_lattice:
    sigma_list.append((element[0]@sigma_list[-1][0]@(element[0].transpose()),sigma_list[-1][1]+element[1]))

plt.plot([i[1] for i in sigma_list], [np.sqrt(np.linalg.det(i[0])) for i in sigma_list],'.-b',label='$\\beta_x
\ [m]$')

plt.grid(True)
plt.xlabel('s [m]')
plt.ylabel('$\epsilon$ [arb. units]');
```



Matching

Now we are going to modify the lattice parameters **to match** the lattice.

Exercise 27

Vary f by hand and try to find

- a value that returns Q= 1/6,
- a value that produces a 90 deg phase-advance. What is the corresponding value of Q?

SOLUTION

In [28]:

```
def plotIt(f):
    L_2=2
    fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
    OTM=compress_beamline(fodo_lattice)
    (tune, beta, alpha, gamma)=R2beta(OTM[0][0])
    print(f'For a f={f} m, the tune is {tune}.')

interactive_plot = interactive(plotIt,f=(1.001,5,.001),continuous_update=True)
output = interactive_plot.children[-1]
output.layout.height = '30px'
interactive_plot
```

In [29]:

```
f=2
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
print(f'The tune is {tune}.')
```

```
The tune is 0.16666666666666677.
```

In [30]:

```
f=2/np.sqrt(2)
L_2=2
fodo_lattice= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
print(f'The tune is {tune}.')
```

```
The tune is 0.24999999999999992.
```

Exercise 28

Matching of a FODO lattice with μ=60 deg and μ=90 deg.

SOLUTION

In [31]:

```
# This is the optics solution for a tune of 1/6
f=2
L_2=2
fodo_lattice= Q(f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_60=np.array([[beta, -alpha],[-alpha, gamma]])
print(f'The tune is {tune}')
display(sigma_60)
```

```
The tune is 0.16666666666666663

array([[6.92820323, 1.73205081],
       [1.73205081, 0.57735027]])
```

In [32]:

```
# This is the optics solution for a tune of 1/4
f=2/np.sqrt(2)
L_2=2
fodo_lattice= Q(f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))
OTM=compress_beamline(fodo_lattice)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_90=np.array([[beta, -alpha],[-alpha, gamma]])
print(f'The tune is {tune}')
sigma_90
```

```
The tune is 0.25000000000000017
```

Out[32]:

```
array([[6.82842712, 2.41421356],
       [2.41421356, 1.        ]])
```

In [33]:

```
# Now we need to propate the solution from sigma60 to sigma90 using a very simple matching section
L_2=2
f1=1.66
f2=1.39
matching_section= Q(f1)+10*D(L_2/10)+Q(-f2)+10*D(L_2/(10))
matching_section_compressed=compress_beamline(matching_section)
matching_section_compressed[0][0]@sigma_60@(matching_section_compressed[0][0]).transpose()
```

Out[33]:

```
array([[6.82913655, 2.4064363 ],
       [2.4064363 , 0.99440619]])
```

In [34]:

```
def plotIt(f1, f2):
    L_2=2
    #f1=1.66
    #f2=1.39
    matching_section= Q(f1)+10*D(L_2/10)+Q(-f2)+10*D(L_2/(10))
    matching_section_compressed=compress_beamline(matching_section)
    print('We get this sigma matrix at the end')
    print(matching_section_compressed[0][0]@sigma_60@(matching_section_compressed[0][0]).transpose())
    print('and we should get')
    print(sigma_90)

interactive_plot = interactive(plotIt,f1=(0.1,5,.001),f2=(0.001,5,.001),continuous_update=True)
output = interactive_plot.children[-1]
output.layout.height = '100px'
interactive_plot
```

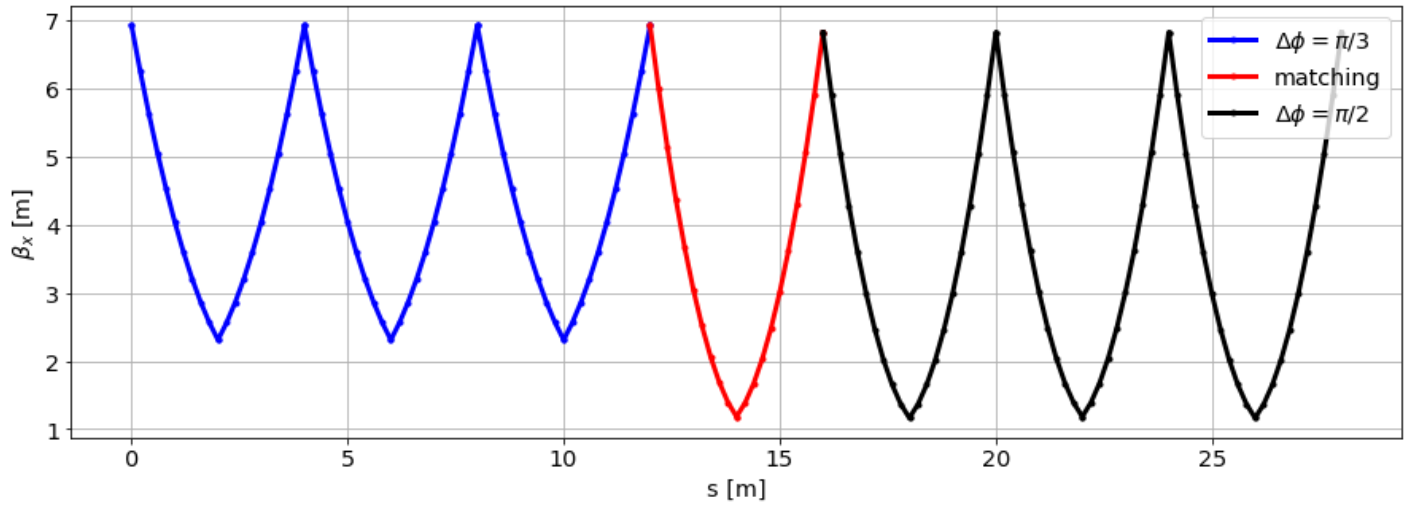
We are going now to plot it.

In [35]:

```
L_2=2
f1=1.66
f2=1.39
matching_section= Q(f1)+10*D(L_2/10)+Q(-f2)+10*D(L_2/(10))
matching_section_compressed=compress_beamline(matching_section)

f_90=2/np.sqrt(2)
L_2=2
fodo_lattice_90= Q(f_90)+10*D(L_2/10)+Q(-f_90)+10*D(L_2/(10))


f=2
fodo_lattice_60= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)

OTM=compress_beamline(fodo_lattice_60)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_60=np.array([[beta, -alpha],[-alpha, gamma]])


sigma_list=[(sigma_60,0)];
for element in 2*fodo_lattice_60+Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+matching_section+fodo_lattice_90*3:
    sigma_list.append((element[0]@sigma_list[-1][0]@(element[0].transpose()),sigma_list[-1][1]+element[1]))

s=[i[1] for i in sigma_list]
x=[i[0][0,0] for i in sigma_list]


import pandas as pd
aux=pd.DataFrame({'s':s,'x':x})
myfilter=aux['s']<12;
plt.plot(aux[myfilter].s,aux[myfilter].x,'.-b',lw=3,label='$\Delta\phi=\pi/3$')
myfilter=(aux['s']>=12-.2) & (aux['s']<16);
plt.plot(aux[myfilter].s,aux[myfilter].x,'.-r',lw=3,label='matching')
myfilter=(aux['s']>=16-.2);
plt.plot(aux[myfilter].s,aux[myfilter].x,'.-k',lw=3,label='$\Delta\phi=\pi/2$')


plt.grid(True)
plt.xlabel('s [m]')
plt.ylabel('$\\beta_x$ [m]')
plt.legend(loc='best')
```

Out[35]:

```
<matplotlib.legend.Legend at 0x1a26ba5f98>
```



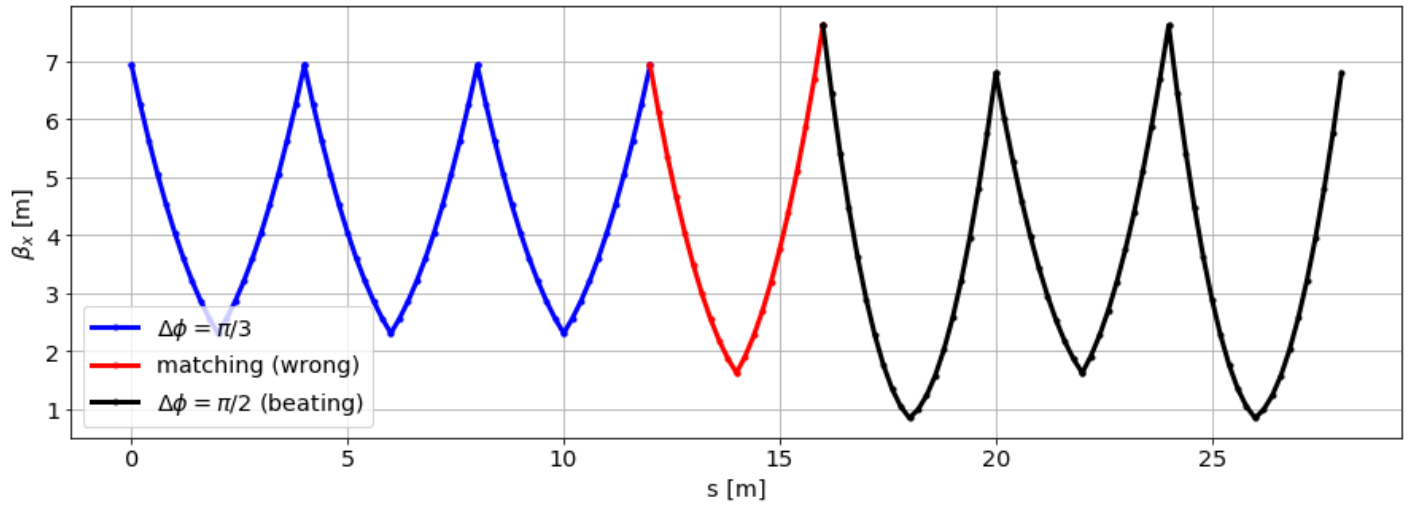
Here you are an example of beta-beating.

In [36]:

```
L_2=2
f1=1.8#1.66
f2=1.39
matching_section= Q(f1)+10*D(L_2/10)+Q(-f2)+10*D(L_2/(10))
matching_section_compressed=compress_beamline(matching_section)

f_90=2/np.sqrt(2)
L_2=2
fodo_lattice_90= Q(f_90)+10*D(L_2/10)+Q(-f_90)+10*D(L_2/(10))


f=2
fodo_lattice_60= Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+Q(2*f)

OTM=compress_beamline(fodo_lattice_60)
(tune, beta, alpha, gamma)=R2beta(OTM[0][0])
sigma_60=np.array([[beta, -alpha],[-alpha, gamma]])


sigma_list=[(sigma_60,0)];
for element in 2*fodo_lattice_60+Q(2*f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))+matching_section+fodo_lattice_90*3:
    sigma_list.append((element[0]@sigma_list[-1][0]@(element[0].transpose()),sigma_list[-1][1]+element[1]))

s=[i[1] for i in sigma_list]
x=[i[0][0,0] for i in sigma_list]


import pandas as pd
aux=pd.DataFrame({'s':s,'x':x})
myfilter=aux['s']<12;
plt.plot(aux[myfilter].s,aux[myfilter].x,'.-b',lw=3,label='$\Delta\phi=\pi/3$')
myfilter=(aux['s']>=12-.2) & (aux['s']<16);
plt.plot(aux[myfilter].s,aux[myfilter].x,'.-r',lw=3,label='matching (wrong)')
myfilter=(aux['s']>=16-.2);
plt.plot(aux[myfilter].s,aux[myfilter].x,'.-k',lw=3,label='$\Delta\phi=\pi/2$ (beating)')


plt.grid(True)
plt.xlabel('s [m]')
plt.ylabel('$\\beta_x$ [m]')
plt.legend(loc='best')
```

Out[36]:

```
<matplotlib.legend.Legend at 0x1a27119860>
```



Even in this simple case, doing the matching by hand is uncovenient. We could build up a simple solver.

In [37]:

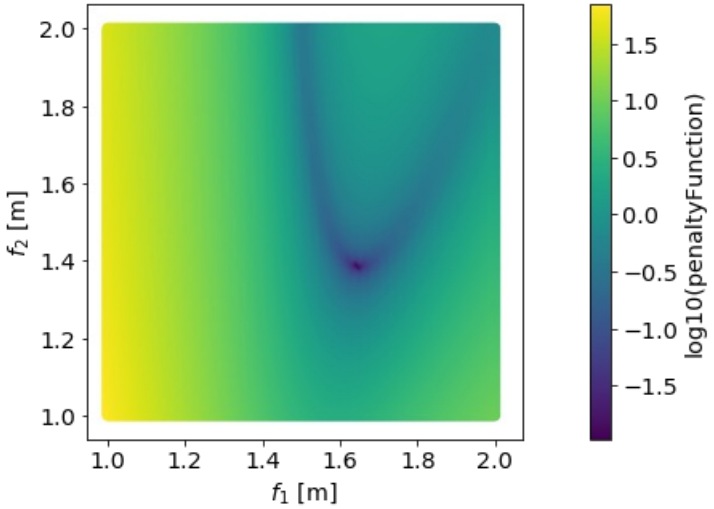
```
# Using a brute force scan
def penaltyFunction(f1, f2):
    f=2
    L_2=2
    fodo_lattice= Q(f)+10*D(L_2/10)+Q(-f)+10*D(L_2/(10))
    OTM=compress_beamline(fodo_lattice)
    (tune, beta, alpha, gamma)=R2beta(OTM[0][0])
    sigma_60=np.array([[beta, -alpha],[-alpha, gamma]])
    #f1=1.66
    #f2=1.39
    matching_section= Q(f1)+10*D(L_2/10)+Q(-f2)+10*D(L_2/(10))
    matching_section_compressed=compress_beamline(matching_section)
    sigma_final=matching_section_compressed[0][0]@sigma_60@(matching_section_compressed[0][0]).transpose()
    return np.sqrt((sigma_final[0,0]-sigma_90[0][0])**2+(sigma_final[0][1]-sigma_90[0][1])**2)
```

In [38]:

```
f1_list=[]
f2_list=[]
penaltyFunction_list=[]
for f1, f2 in itertools.product(np.linspace(1,2,200),np.linspace(1,2,200)):
    f1_list.append(f1)
    f2_list.append(f2)
    penaltyFunction_list.append(penaltyFunction(f1,f2))
```

In [39]:

```
plt.scatter(f1_list,f2_list,c=np.log10(penaltyFunction_list))
plt.axis('square');
plt.xlabel('$f_1$ [m]')
plt.ylabel('$f_2$ [m]')
cb=plt.colorbar()
cb.set_label('log10(penaltyFunction)')
```



It is very important to appreciate the NON-linearities of linear optics, when we refer to relation between focal-length and optics function.

In [40]:

```
# to find the minimum use pandas dataframes
aux=pd.DataFrame({'f1':f1_list,'f2':f2_list, 'penalty function':penaltyFunction_list})
aux[aux['penalty function']==aux['penalty function'].min()]
```

Out[40]:

	f1	f2	penalty function
26278	1.658291	1.39196	0.010528

In [42]:

```
from scipy import optimize
def penaltyFunctionSingleArgument(argument):
    return penaltyFunction(argument[0],argument[1])

optimize.minimize(penaltyFunctionSingleArgument,[1,1])
```

Out[42]:

```
      fun: 2.5248341076039876e-08
 hess_inv: array([[ 7.65291583e-09, -1.35709267e-09],
       [-1.35709267e-09,  2.44228506e-09]])
      jac: array([0.75995995, 1.02077933])
  message: 'Desired error not necessarily achieved due to precision loss.'
     nfev: 639
      nit: 54
     njev: 157
   status: 2
  success: False
        x: array([1.65681256, 1.38987534])
```

In [ ]:

```

```