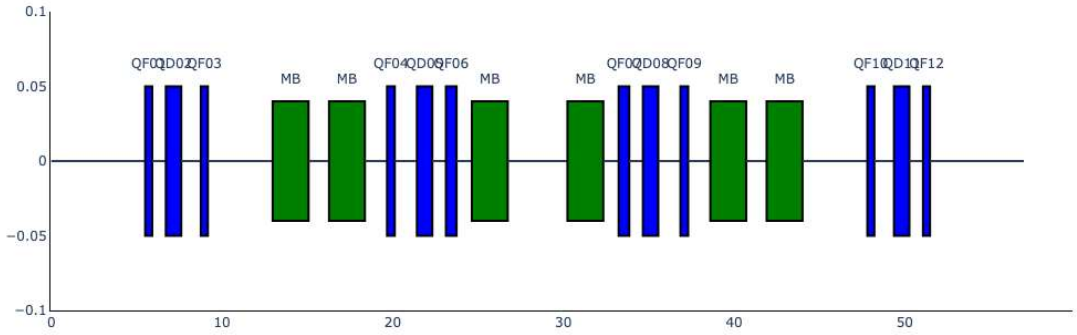
大作业编程基础参考

刘星光，2023年11月26日

RCS-R1布局示意图



In [ ]:

```
#0. 定义基本传输矩阵

import numpy as np

# 漂移段，3*3
def D(l):
    return np.array([[1,l,0],
                     [0,1,0],
                     [0,0,1]])

# 四极磁铁：聚焦平面
def QF(l,k):
    sk=np.sqrt(k)
    skl=sk*l
    return np.array([[np.cos(skl),np.sin(skl)/sk,0],
                     [-sk*np.sin(skl),np.cos(skl),0],
                     [0,0,1]])

# 四极磁铁：散焦平面
def QD(l,k):
    sk=np.sqrt(np.abs(k))
    skl=sk*l
    return np.array([[np.cosh(skl),np.sinh(skl)/sk,0],
                     [sk*np.sinh(skl),np.cosh(skl),0],
                     [0,0,1]])

# 偏转磁铁（sector magnet）：推荐以偏转半径rho 和 偏转角度 theta 定义
def SBend(r, a):
    return np.array([[np.cos(a),    r*np.sin(a),    r*(1-np.cos(a))],
                     [-np.sin(a)/r, np.cos(a),      np.sin(a)],
                     [0,            0,              1]])
# 注：无偏转的平面（比如y）按漂移段处理

# 计算一系列矩阵的点积（dot product），注意点积的顺序
def Mdot(ml):
    mt = ml[0] # 储存结果的临时矩阵，先存上第一个
    if len(ml) == 1: # 如果只有一个元件则直接返回
```

```
        return mt
    else:
        # 依次对元件进行点积（注意顺序）
        for i in range(1,len(ml)):
            mt = np.dot(ml[i],mt)
    return mt

# 通过比对传输矩阵获得twiss参数
# cos φ + α sin φ       β sin φ
# -γ sin φ              cos φ - α sin φ
def twiss_from_m(m):
    mtr=m[0,0]+m[1,1] # 注：虽然定义为三阶用于计算含色散的函数，计算twiss参数比对时仅用2x
    if (mtr>2):
        raise ValueError("二阶矩阵的迹应小于2才是稳定解",mtr)
    c=mtr*0.5 # cos项，等于迹的一半
    # 计算其他项
    s=np.sqrt(1-c**2)
    beta=m[0,1]/s
    alpha=(m[0,0]-c)/s
    gamma=-(m[1,0])/s
    return [beta,alpha,gamma]
```

In [ ]:

```
#1. 来验证函数的正确性，先计算个简单的FODO试试：

# QF: QUAD, L = 0.4, K1 = 4.0;
# D01: DRIFT, L = 0.5;
# QD: QUAD, L= 0.4, K1 = -3.8; # 如果选4.0的话，x和y方向将完全对称，可以试试，这里略取不
# D02: DRIFT, L =0.5;

# 以最直接的方式分别列出两方向的矩阵列表
latx=[QF(0.4,4),D(0.5),QD(0.4,-3.8),D(0.5)]
laty=[QD(0.4,-4),D(0.5),QF(0.4,3.8),D(0.5)]
# 本别获取FODO两个方向的总矩阵
Mx=Mdot(latx)
My=Mdot(laty)

# 输出两个方向的总矩阵确认和后续对比
print("Mx:\n",Mx)
print("My:\n",My)

# det(M) 任意一个行列式（包括最后总的矩阵）的行列式应该为1 （推荐养成检查的好习惯）
print("注1：任意行列式应当为1")
print("det(Mx):",np.linalg.det(Mx))
print("det(My):",np.linalg.det(My))

# 计算矩阵的迹 2cos(\theta)，注意必须小于2才是稳定解
print("注2：矩阵的迹应当小于2才是稳定解：")
print("Mx的迹：",Mx[0,0]+Mx[1,1])
print("My的迹：",My[0,0]+My[1,1])
```

```
Mx:
 [[-1.62512436  2.29407824  0.        ]
 [-1.92808155  2.10640491  0.        ]
 [ 0.          0.          1.        ]]
My:
 [[ 1.32959105  0.98661757  0.        ]
 [-1.78724743 -0.57410864  0.        ]
 [ 0.          0.          1.        ]]
注1：任意行列式应当为1
det(Mx): 1.0000000000000013
det(My): 1.0000000000000002
```

注2：矩阵的迹应当小于2才是稳定解：
Mx的迹： 0.48128055182485285
My的迹： 0.7554824132816707

In [ ]:

```
#2. 计算Twiss参数并作图
# 一般最大最小值在四极磁铁的中点，因此对四极磁铁进行切分并排成对称布局，但用作tracking时没必
latx=[QF(0.4*0.5,4),D(0.5),QD(0.4*0.5,-3.8),QD(0.4*0.5,-3.8),D(0.5),QF(0.4*0.5,4)]
laty=[QD(0.4*0.5,-4),D(0.5),QF(0.4*0.5,3.8),QF(0.4*0.5,3.8),D(0.5),QD(0.4*0.5,-4)]
Mx=Mdot(latx)
My=Mdot(laty)
# 输出两个方向的总矩阵确认和后续对比
print("传输矩阵：\n")
print("Mx:\n",Mx)
print("My:\n",My)

print("注1：任意行列式应当为1")
print("det(Mx):",np.linalg.det(Mx))
print("det(My):",np.linalg.det(My))

print("注2：矩阵的迹应当小于2才是稳定解：")
print("Trace Mx",Mx[0,0]+Mx[1,1])
print("Trace My",My[0,0]+My[1,1])

print("注3：同一个FODO，选择的起点不同，矩阵的形式不同（对应不同的twiss参数和相空间）")

# 输出twiss参数
print("βx,αx,γx: ",twiss_from_m(Mx))
print("βy,αy,γy: ",twiss_from_m(My))

# 获取以任意原件为起点时的传输矩阵和twiss参数
for i,m in enumerate(latx):
    # print(i,m)
    lt = latx[i:]+latx[:i]
    print("Twiss参数：X:\n",i,twiss_from_m(Mdot(lt)))

for i,m in enumerate(laty):
    # print(i,m)
    lt = laty[i:]+laty[:i]
    print("Twiss参数：Y:\n",i,twiss_from_m(Mdot(lt)))
```

传输矩阵：

```
Mx:
 [[ 0.24064028  2.68849456  0.        ]
 [-0.35041628  0.24064028  0.        ]
 [ 0.          0.          1.        ]]
My:
 [[ 0.37774121  0.80579039  0.        ]
 [-1.0639387   0.37774121  0.        ]
 [ 0.          0.          1.        ]]
```

注1：任意行列式应当为1
det(Mx): 0.9999999999999996
det(My): 1.0000000000000004
注2：矩阵的迹应当小于2才是稳定解：
Trace Mx 0.48128055182485285
Trace My 0.7554824132816704
注3：同一个FODO，选择的起点不同，矩阵的形式不同（对应不同的twiss参数和相空间）
βx,αx,γx: [2.7698895079255754, 0.0, 0.36102523120097996]
βy,αy,γy: [0.8702674450516203, -1.1990599151012757e-16, 1.149072053293554]
Twiss参数：X:
0 [2.7698895079255754, 0.0, 0.36102523120097996]
Twiss参数：X:
1 [2.363532122421277, 1.9222511974864858, 1.986454773218174]

```
Twiss参数: X:
 2 [0.9378946182393347, 0.9290238108773984, 1.986454773218174]
Twiss参数: X:
 3 [0.7609647579248855, 1.429794181848416e-16, 1.3141213040232667]
Twiss参数: X:
 4 [0.9378946182393346, -0.9290238108773984, 1.986454773218174]
Twiss参数: X:
 5 [2.3635321224212764, -1.9222511974864855, 1.986454773218174]
Twiss参数: Y:
 0 [0.8702674450516203, -1.1990599151012757e-16, 1.149072053293554]
Twiss参数: Y:
 1 [1.0655639026729824, -1.0280141651771373, 1.9302578837790036]
Twiss参数: Y:
 2 [2.5761425387948704, -1.993143107066639, 1.9302578837790036]
Twiss参数: Y:
 3 [2.9962768420005688, 5.995299575506378e-17, 0.33374753159735265]
Twiss参数: Y:
 4 [2.576142538794871, 1.9931431070666394, 1.930257883779004]
Twiss参数: Y:
 5 [1.0655639026729824, 1.0280141651771375, 1.9302578837790039]
```

In [ ]:

```python
#3. 一般情况：现在我们增加两个函数，让上述定义的函数更有用和具有一般性

# 根据输入参数返回相应的传输矩阵，注意相应的处理要与lattice元件的定义格式相对应，可自行整理
def element_matrix(*args):
    if args[0] == 'D':
        l=args[1]
        return [D(l),D(l),l]
    elif args[0] == 'Q':
        l=args[1]
        k=args[2]
        if k>0:
            return [QF(l,k),QD(l,-k),l]
        elif k<0:
            return [QD(l,k),QF(l,-k),l]
        else:
            return [D(l),D(l),l] # if k=0, 按漂移段处理
    elif args[0] == 'SBend':
        l=args[1]
        angle=args[2]
        rho=l/angle
        return [SBend(rho,angle),D(l),l]
    else:
        print('未定义元件类型!')
        return None

# 返回x方向，y方向的传输矩阵及（各元件出口）位置坐标 [type,a1,a2,a3]
def lat_matrices(lat):
    ml=[element_matrix(e[0],*e[1:]) for e in lat]
    mlx=list(map(lambda x: x[0],ml)) # 选择x矩阵列表，这里用了python的lambda函数，可以其
    mly=list(map(lambda x: x[1],ml)) # 选择y矩阵列表
    mls=list(map(lambda x: x[2],ml)) # 选择元件位置列表
    return [mlx,mly,mls]
```

In [ ]:

```python
# e.g 以前述FODO为例

# QF: QUAD, L = 0.4, K1 = 4.0;
# D01: DRIFT, L = 0.5;
# QD: QUAD, L= 0.4, K1 = -3.8; # choose different value so x- and y- directions are
# D02: DRIFT, L =0.5;

# 这里假设每个元件都由四个值定义，保持同样的格式便于从外部文件读取（如果元件列表参数是从文件
```

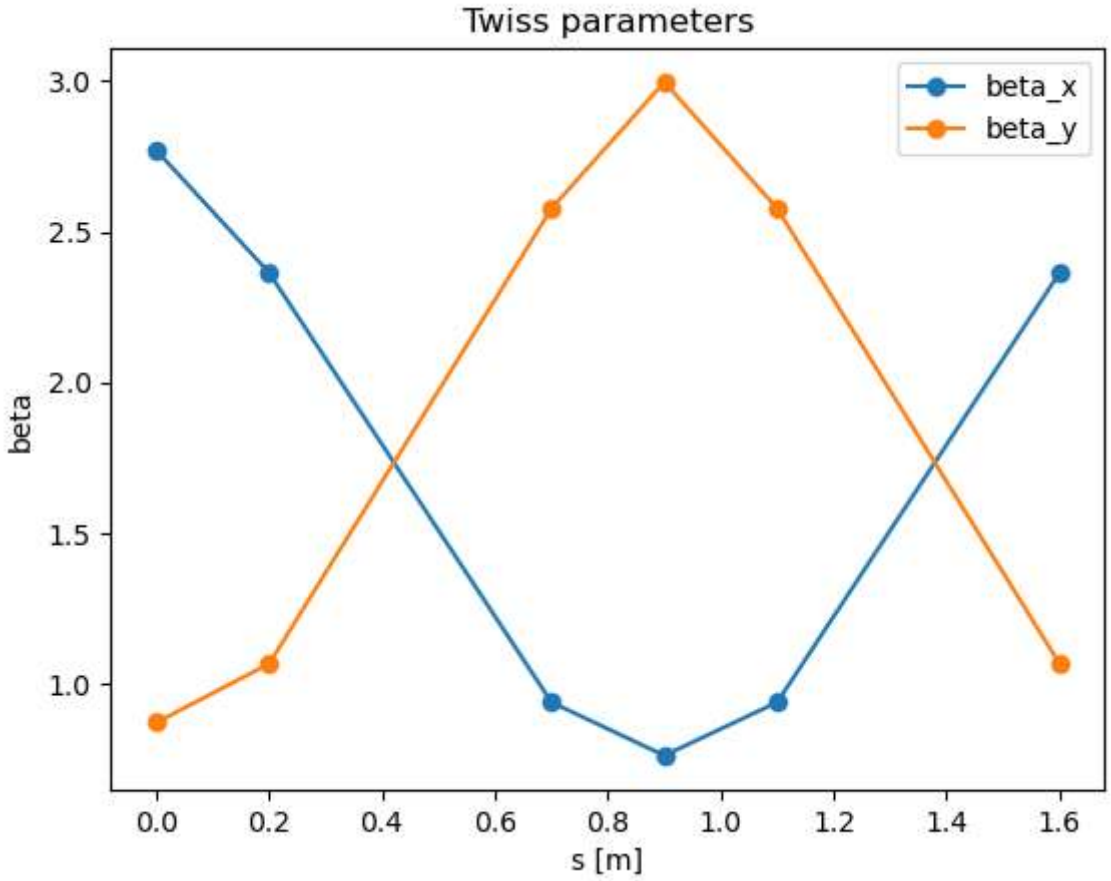
```
lat = [["Q",0.4*0.5,4.0],["D",0.5,0,0],["Q",0.4*0.5,-3.8],["Q",0.4*0.5,-3.8],["D",0.
# 计算FODO的矩阵和元件长度列表
[mlx,mly,mls]=lat_matrices(lat)
print("Mx:\n",Mdot(mlx),"\nMy:\n",Mdot(mly),mls)
```

```
Mx:
 [[ 0.24064028  2.68849456  0.        ]
 [-0.35041628  0.24064028  0.        ]
 [ 0.          0.          1.        ]]
My:
 [[ 0.37774121  0.80579039  0.        ]
 [-1.0639387   0.37774121  0.        ]
 [ 0.          0.          1.        ]] [0.2, 0.5, 0.2, 0.2, 0.5, 0.2]
```

In [ ]:

```
# 依次以各元件为起点计算传输矩阵，再由矩阵对比的方式获得Twiss参数
twiss_x=[]
twiss_y=[]
# 通过按格式定义的元件列表返回x，y方向的传输矩阵列表和元件长度列表
[mlx,mly,mls]=lat_matrices(lat) # 实际这里我们只是为了存mls的值计算一下初始的排布各元件位
sl=np.cumsum(mls) #可以写个简单的循环计算元件位置坐标，这里用了numpy内置函数
for i,l in enumerate(mls):
    sl[i]-=l # 修正sl为元件入口处位置
print("sl:\n",sl)

for i,e in enumerate(lat):
    lat_temp=lat[i:]+lat[:i] # 重新排元件顺序，以第i个元件为起始
    # print(lat_temp)
    [mlx,mly,mls]=lat_matrices(lat_temp)
    [b,a,r]=twiss_from_m(Mdot(mlx))
    twiss_x.append([sl[i],b,a,r])
    [b,a,r]=twiss_from_m(Mdot(mly))
    twiss_y.append([sl[i],b,a,r])

# 通过作图确认twiss参数的beta函数
import matplotlib.pyplot as plt

plt.plot(np.array(twiss_x)[:,0],np.array(twiss_x)[:,1],'-o')
plt.plot(np.array(twiss_y)[:,0],np.array(twiss_y)[:,1],'-o')
plt.xlabel('s [m]')
plt.ylabel('beta')
plt.title('Twiss parameters')
plt.legend(['beta_x','beta_y'])
plt.show()
```

```
sl:
 [0.  0.2 0.7 0.9 1.1 1.6]
```

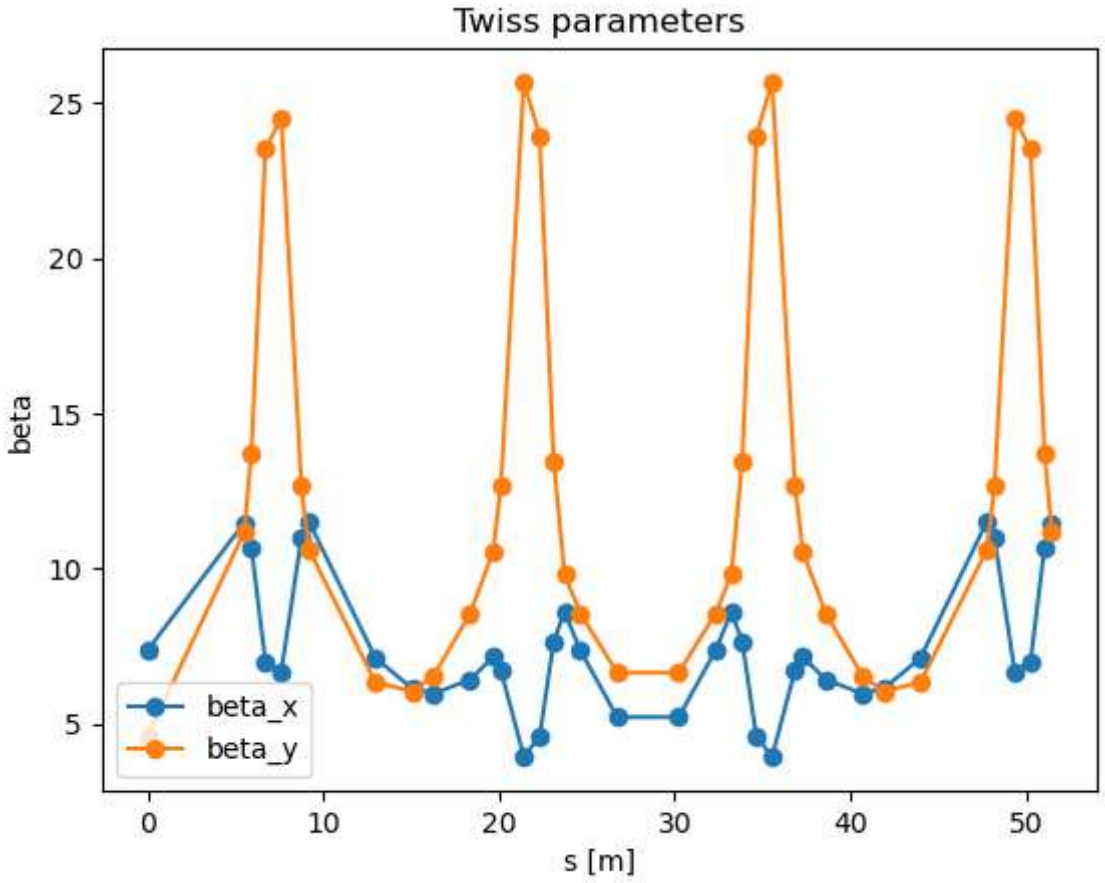


In [ ]:

```
#4. 计算RCS—R1段twiss参数
rcs_r1=[["D",5.5,0],
["Q",0.41,0.7406566],
["D",0.8,0],
["Q",0.9,-0.579364096],
["D",1.15,0],
["Q",0.41,0.664196402],
["D",3.8,0],
["SBend",2.1,0.261799388],
["D",1.2,0],
["SBend",2.1,0.261799388],
["D",1.3,0],
["Q",0.45,0.570558026],
["D",1.3,0],
["Q",0.9,-0.579364096],
["D",0.8,0],
["Q",0.62,0.608457761],
["D",0.9,0],
["SBend",2.1,0.261799388],
["D",3.5,0],
["SBend",2.1,0.261799388],
["D",0.9,0],
["Q",0.62,0.608457761],
["D",0.8,0],
["Q",0.9,-0.579364096],
["D",1.3,0],
["Q",0.45,0.570558026],
["D",1.3,0],
["SBend",2.1,0.261799388],
["D",1.2,0],
["SBend",2.1,0.261799388],
["D",3.8,0],
["Q",0.41,0.664196402],
["D",1.15,0],
```

```
["Q",0.9,-0.579364096],
["D",0.8,0],
["Q",0.41,0.7406566],
["D",5.5,0]]

# 依次以各元件为起点计算传输矩阵，再由矩阵对比的方式获得Twiss参数
twiss_x=[]
twiss_y=[]
# caculate acuumulated values for mls, you can do it with a small loop if not with p
[mlx,mly,mls]=lat_matrices(rcs_r1) # only mls would be use here

sl=np.cumsum(mls) #可以写个简单的循环计算元件位置坐标，这里用了numpy内置函数
for i,l in enumerate(mls):
    sl[i]-=l # 修正sl为元件入口处位置
print("sl:\n",sl)

for i,e in enumerate(rcs_r1):
    lat_temp=rcs_r1[i:]+rcs_r1[:i] # 重新排元件顺序，以第i个元件为起始
    # print(lat_temp)
    [mlx,mly,mls]=lat_matrices(lat_temp)
    [b,a,r]=twiss_from_m(Mdot(mlx))
    twiss_x.append([sl[i],b,a,r])
    [b,a,r]=twiss_from_m(Mdot(mly))
    twiss_y.append([sl[i],b,a,r])

# 通过作图确认twiss参数的beta函数
import matplotlib.pyplot as plt

plt.plot(np.array(twiss_x)[:,0],np.array(twiss_x)[:,1],'-o')
plt.plot(np.array(twiss_y)[:,0],np.array(twiss_y)[:,1],'-o')
plt.xlabel('s [m]')
plt.ylabel('beta')
plt.title('Twiss parameters')
plt.legend(['beta_x','beta_y'])
plt.show()
```

```
sl:
 [ 0.    5.5   5.91  6.71  7.61  8.76  9.17 12.97 15.07 16.27 18.37 19.67
 20.12 21.42 22.32 23.12 23.74 24.64 26.74 30.24 32.34 33.24 33.86 34.66
 35.56 36.86 37.31 38.61 40.71 41.91 44.01 47.81 48.22 49.37 50.27 51.07
 51.48]
```


Twiss parameters
beta
s [m]
beta_x
beta_y
25
20
15
10
5
0
10
20
30
40
50